

# Trebank Refinement

Optimising Representations of Syntactic Analyses  
for Probabilistic Context-Free Parsing

von

Tylman Ule



Philosophische Dissertation  
angenommen von der Neuphilologischen Fakultät  
der Universität Tübingen

am 12. Juni 2006

München

2007



Gedruckt mit Genehmigung der Neuphilologischen Fakultät  
der Universität Tübingen

Hauptberichterstatter: Prof. Dr. Erhard Hinrichs  
Mitberichterstatter: Prof. Dr. Uwe Mönnich  
Dekan: Prof. Dr. Joachim Knappe



# Acknowledgements

I am thankful to all the people that have supported me during the years that I spent working on this thesis.

The one person who was most directly involved in stopping me from diverging any longer from the main goal of finishing this thesis was Jorn Veenstra. He was critical about my ideas and ever ready to comment on them and to discuss the questions that I often had. But he also helped me to understand that at some point, you simply have to stop producing new ideas, and that you have to restrict yourself to writing down the most interesting consequences of the best ideas that you have come up with so far. A lengthy talk in his own walls abroad laid the basis for this thesis. I did not change it much lateron.

Frank Müller helped me to start writing papers at all. He convinced me that our work was interesting enough to be published, and the reviewers agreed. Frank and I and our Espresso maker spent many years together, and I will miss his Cappuccino.

All the time I have been working in Tübingen, my supervisor Erhard Hinrichs took the time to listen to my ideas. He supported my plans for the outline of my thesis and gave me enough freedom to implement them. He was always friendly and supportive to me and expanded my knowledge of central topics in numerous courses.

A thesis should have a central idea that connects all of its parts. Kiril Simov helped me to find it when he invited me to his laboratory in Sofia. Working and being with him has always been very pleasant, but this time I was especially lucky, because when reading in his library, I discovered a paper that would give the central theme to my thesis: Grammar Refinement.

Going on, I was encouraged by most interesting discussions with Detlef Prescher and Helmut Schmid to stay on my track. Helmut offered me to modify the source code of his parsers, which made some of the experiments reported here possible in the first place.

I have been part of the SFB in Tübingen, and felt encouraged by my colleagues there. Holger Wunsch commented my plans quietly but precisely

and helped me keep up my spirit. I was amazed that Heike Telljohann was always kind and willing to explain the intricacies of TüBa-D/Z to me. A big thank you also to the members of my thesis committee, Fritz Hamm, Uwe Mönnich, and Frank Richter.

Before I joined the SFB, I worked in a project with Wolfgang Lezius and Esther König, who helped me believe that we are doing a good job despite of some problems. Without Lothar Lemnitzer I probably would have done something completely different, because he drew me to Tübingen and has been pleasant to work with all the time that I know him. Jochen Saile never let me down when I had another technical problem.

Thanks also goes to the student staff that helped me build my data and programs, including Maureen Dunne, Aisling Fleming, Steffen Frömel, Katrina Keogh, Wolfgang Maier, Nicole Maruschka and Sigrid Schmitt.

I found refuge for writing up my book at Wolf Paprotté's Arbeitsbereich Linguistik in Münster – Thank you! The support I had from his staff was tremendous, including Hendrik Cyrus, Lea Cyrus, Robert Memering and Johannes Schwall. They did not mind to delve into whatever topic I had to write down. It was a nice year, thanks!

Last, but not least comes my family. My mother seems to believe in me even if I am not sure myself that I am doing the right thing. She helps me to question my ways, and to calmly solve all problems ahead. My grandmother has always shown me that it's fun to learn and ask, and that you can understand about everything. And what is more, she let me explain her so many things that I became confident I had understood them. There is much more family that helped me complete this thesis, because I could always rely on them. Thanks to you, Astrid, Christof, Dominique and Joëlle. Britta and Jan, I am not sure whether you realise how much you supported me with your visits and invitations. And your sense for life, even outside University.

Nadja, I am so happy that we have gone through this together!

You took me through these years – Thanks!

*meiner Großmutter*





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>I</b>	<b>Parsing and Parser Evaluation</b>	<b>5</b>
<b>2</b>	<b>Mostly Context-Free Probabilistic Parsing</b>	<b>7</b>
2.1	Context-Free Grammars . . . . .	7
2.2	Probabilistic Context-Free Grammars . . . . .	9
2.3	Parsing with Context-Free Grammars . . . . .	11
2.4	Parsing with Probabilistic Context-Free Grammars . . . . .	12
2.5	Efficient Viterbi Parsing . . . . .	14
2.6	Alternative Approaches to Probabilistic Parsing . . . . .	17
<b>3</b>	<b>Parser Evaluation</b>	<b>21</b>
3.1	Parseval . . . . .	24
3.2	Relational Evaluation . . . . .	28
3.3	Relative Differences in Information . . . . .	30
<b>II</b>	<b>Representations of Syntactic Analyses</b>	<b>35</b>
<b>4</b>	<b>A Treebank’s Choice</b>	<b>37</b>
4.1	Observations: Challenges for Proper Trees . . . . .	44
4.1.1	The Objects of Syntactic Annotation . . . . .	44
4.1.2	Unattached Elements . . . . .	46
4.1.3	Free Constituent Order . . . . .	47
4.1.4	Coordination . . . . .	54
4.1.5	Named Entities . . . . .	59
4.2	Representations of Relations . . . . .	65
4.2.1	Class, Sequence, Domination . . . . .	66
4.2.2	Arbitrary Relations and Parallel Structures . . . . .	67

---

4.3	Encoding the Analyses . . . . .	68
4.3.1	Proper Trees . . . . .	69
4.3.2	Crossing Edges, Co-Indexation, and Secondary Edges . . . . .	69
4.3.3	The Semantics of a Small Label Set . . . . .	71
4.4	Conclusion . . . . .	72
<b>5</b>	<b>PCFG Treebank Grammars</b>	<b>75</b>
5.1	Syntactic Bias of a PCFG Treebank Grammar . . . . .	76
5.1.1	Bias in TüBa-D/Z and negra . . . . .	78
5.1.2	Extending the Label Set . . . . .	82
5.1.3	Structural and Lexical Preferences . . . . .	85
5.2	PCFG-Parsing of TüBa-D/Z . . . . .	88
5.2.1	Parentheses and Punctuation . . . . .	90
5.2.2	Secondary Edges and Co-Indexation . . . . .	100
5.2.3	Train and Test Regimes . . . . .	100
5.2.4	POS-Tagging and Parsing . . . . .	103
5.3	Conclusion . . . . .	107
<b>6</b>	<b>Related Research</b>	<b>109</b>
6.1	Algorithmic Transformations . . . . .	109
6.2	Tuning a Treebank to a Parser . . . . .	112
6.3	Parsing German . . . . .	115
6.4	Conclusion . . . . .	117
<b>III</b>	<b>Treebank Refinement</b>	<b>119</b>
<b>7</b>	<b>Including Local Context into Nonterminals</b>	<b>121</b>
7.1	More Context in Context-Free Grammars . . . . .	122
7.1.1	More Context Required . . . . .	124
7.1.2	Formal Equivalence of Grammars . . . . .	132
7.1.3	Experimental Adequacy . . . . .	133
7.2	Context – Focus – Production . . . . .	135
7.3	Determining Deviant Distributions . . . . .	140
7.3.1	$\chi^2$ Goodness-of-Fit . . . . .	141
7.3.2	$\chi^2$ Merging Infrequent Classes . . . . .	143
7.3.3	Kolmogorov-Smirnov Goodness-of-Fit . . . . .	144
7.3.4	Kullback-Leibler Divergence . . . . .	146
7.3.5	Skew Divergence . . . . .	147
7.4	Iteratively Adding Local Context . . . . .	151
7.5	Unconditionally Spreading Parent Information . . . . .	157

---

7.6	Conclusion . . . . .	157
<b>8</b>	<b>Extended Local Context</b>	<b>159</b>
8.1	Relevant Context . . . . .	159
8.2	Extending Context . . . . .	162
8.2.1	Ancestor Context . . . . .	163
8.2.2	Descendant Context . . . . .	171
8.2.3	Attribute Context . . . . .	175
8.3	Lexical Context . . . . .	183
8.3.1	Unrestricted Lexical POS Splitting . . . . .	184
8.3.2	Restricted Lexical POS Splitting . . . . .	188
8.3.3	Selecting Lexicalised POS Before and During Parsing .	189
8.4	Recursiveness . . . . .	195
8.4.1	High-Frequent Node Labels . . . . .	198
8.4.2	MF, Recursively . . . . .	199
8.4.3	SIMPX, Recursively . . . . .	199
8.4.4	NX, Recursively . . . . .	204
8.4.5	The Impact of Lexicalisation in TüBa-D/Z and negra .	210
8.5	Varying Detail of Input and Output . . . . .	213
8.6	Evaluation on Final Data Sets . . . . .	218
8.7	Conclusion . . . . .	218
<b>9</b>	<b>Handling Sparse Data</b>	<b>221</b>
9.1	Merging Similar Nodes . . . . .	223
9.1.1	Similar Parents and Unmergeable Nodes . . . . .	224
9.1.2	Similarity of Split Nodes . . . . .	225
9.2	Splitting with Context History . . . . .	232
9.3	Conclusion . . . . .	237
<b>10</b>	<b>Treebank Refinement Aiding Supervised Annotation</b>	<b>239</b>
10.1	Methods and Data . . . . .	240
10.1.1	Unexpected Productions . . . . .	240
10.1.2	Ranking Error Candidates . . . . .	241
10.1.3	Early and Revised Annotation . . . . .	242
10.1.4	Evaluation via Artificial Errors . . . . .	243
10.2	Experiments and Results . . . . .	244
10.3	Discussion and Related Work . . . . .	248
10.4	Conclusion . . . . .	249
<b>11</b>	<b>Conclusions</b>	<b>251</b>

---

<b>A Label Sets used in negra and TüBa-D/Z</b>	<b>255</b>
A.1 STTS POS Tags . . . . .	255
A.2 negra Node and Edge Labels . . . . .	257
A.3 TüBa-D/Z Node and Edge Labels . . . . .	259
<b>B PP-Attachment in negra and TüBa-D/Z with Edges</b>	<b>263</b>
<b>Bibliography</b>	<b>269</b>

# Chapter 1

## Introduction

Syntactic parsing plays a central role in the automatic processing of natural language. Parsing identifies and relates the constituent parts of linear input, and any application that tries to find out who did what to whom for some novel input will need to determine the constituent parts of the input that correspond to who, what, and whom. Ever since the basis of formal language theory has been laid out by Chomsky (1956), the description of natural language in terms of context-free grammars was adopted by many parsing systems. While parsers were initially designed as hand-built rule systems, the advent of large syntactically annotated language resources in the 1990s, so-called treebanks, has triggered an interest in deriving parsers directly from these resources. Interestingly, context-free grammars have again served as the starting point for many of these approaches, because the long line of research has produced efficient algorithms for parsing with them.

The efficiency comes at some cost, though. It has been shown that context-free grammars cannot adequately describe certain phenomena in free word order languages, such as cross-serial dependencies or certain scrambling phenomena. They also show rather limited parsing performance when probabilistic versions of these grammars are derived straightforwardly from treebanks. The conditioning of probabilistic context-free grammars has consequently been extended to take into account more contextual information, i.e. parameters have been added to the model, resulting in high-performing parsers, which, however, depart from the original context-free view of language. One of the most influential extended probabilistic models has been proposed by Collins (1997) for English. This model has consequently been modified and applied to other languages, including Czech and German (Collins *et al.*, 1999; Dubey and Keller, 2003). It has become apparent that the adaptation was necessary because the languages differ, but also because the different treebanks adopt different representations of syntactic analyses.

We therefore investigate a different approach to finding a good parser for a given treebank here. Instead of considering the treebank as fix, so that a probabilistic model has to be tuned to it and the model has to be extended appropriately, we consider the grammar formalism as fix, and modify the treebank so that it represents the syntactic analyses in a way that is more appropriate for the grammar. This approach is motivated by previous research observing considerable variation in the performance of parsers based on probabilistic context-free grammars when the representation of syntactic analyses is modified. We do not intend to modify the analyses, which would change the parsing task, and therefore we require equivalence between the different representations. This equivalence can be easily guaranteed by using the original annotation found in the treebank as the standard against which any parser is evaluated. We also motivate this approach by noting that the shape of the annotation as we observe it in the treebank on which we perform our experiments is to some degree arbitrary, and primarily geared towards the developers and human users of the treebank. The same syntactic analyses can thus be expressed differently and more appropriately for probabilistic context-free parsing without losing any aspect of the original analyses. An automatic approach for this optimisation, which consistently improves parsing performance on a number of different data sets, is the main contribution of this thesis. Being automatic, this approach obviates the need for tedious manual adaptations that are otherwise necessary for different languages, and also for different representations of syntactic analyses for the same language.

The outline of the thesis is as follows. In the first part, we show how we parse sentences with a given context-free grammar, and that plain probabilistic versions of such a grammar are still attractive despite their mature age, because they can parse natural language efficiently (chapter 2). We also show how we evaluate the success of parsers, and that we rely on a combination of evaluation metrics in order to avoid being misled by weaknesses of the individual metrics (chapter 3).

The second part of this thesis shows how the shape of syntactic annotation in a German treebank has to meet diverse goals such as the faithful rendering of regularities in the language data, the conformance to existing annotation standards, and simplicity of the annotation scheme, so that both annotators and users of the treebank experience a steep learning curve. The representation of syntactic analyses in the treebank is carefully designed to meet these partly conflicting requirements (chapter 4). On the other hand, changes in the representation of syntactic analyses affect the performance of parsers based on probabilistic context-free grammars that are derived from the treebank. Context-free grammars are susceptible to changes in the representation of analyses because they determine the possible children of a

node exclusively from its label. For similar reasons, probabilistic context-free grammars are susceptible to changes in the representation of analyses, because they determine the probability of a subtree of the full tree that serves as syntactic analysis independent of the position of the subtree in the sentence, and independent of any words or structure outside the subtree. Parsing performance thus strongly depends on the choice of labels for such a grammar. This observation is especially important because a probabilistic context-free grammar can only be derived from most treebanks after they have been modified, and any modification may affect parsing performance. We therefore detail how we modify the German treebank that we use for most of our experiments (chapter 5). We conclude the second part of this thesis with a survey of related research that shows how modifications in a treebank can boost parser performance. These reported modifications are most effective when determined intellectually and not algorithmically (chapter 6).

The third part of this thesis gives our main contribution. First, we emphasise that while context-free grammars are not able to generate certain configurations that may occur in treebanks, such as cross-serial dependencies, they can still enumerate a limited number of these configurations by extending the finite set of labels and context-free rules. Evaluation then determines the degree of success of such an enumeration, and in this spirit we algorithmically modify the treebank and evaluate the success in the experiments of the remaining thesis. We first define a test inspired by Bockhorst and Craven (2001) that tries to assess whether the usage of the node labels in the treebank is faithfully captured by probabilistic context-free grammars. The test compares the distribution of expansions of nodes when more knowledge is available than just the node label, to the distribution when only the node label is available, which corresponds to the view of a probabilistic context-free grammar. We experimentally determine a successful instantiation of the test that uses an information-based metric and show that repeated applications of this test and corresponding modifications of the treebank improve parsing performance of plain probabilistic context-free grammars derived from it (chapter 7). These repeated tests and modifications are what we call Treebank Refinement.

We further present extensions to the test, where we determine the utility of more information about adjacent nodes for predicting structure from the perspective of a probabilistic context-free parser. The additional information includes lexical items, and also optional information that is not part of the target structure, such as fine-grained grammatical functions when we only seek to find bracketings without functional labels. A more detailed evaluation reveals that the extended information helps assigning more complex recursive structure, while its effect on node labels used for high-frequent flat structure is

less strong. We show that any kind of information considered for our method to test and modify the labels of treebanks improves parsing performance of probabilistic context-free grammars (chapter 8). A separate chapter is dedicated to handling sparse data problems that arise when many new node labels are introduced into a treebank (chapter 9).

Before we conclude, we show that Treebank Refinement can be tuned to detect rare and unexpected events, and that these events often turn out to be inappropriate labels originating from annotation errors. We show that these errors can be found reliably in treebanks for different languages and at different stages of completion by classifying the node labels modified by Treebank Refinement in a machine learning paradigm (chapter 10).



Part I

Parsing and Parser Evaluation



## Chapter 2

# Mostly Context-Free Probabilistic Parsing

Syntactic parsing of natural language determines the relations between the parts of utterances. Sophisticated parsing techniques continue to emerge, but the early notion of context-free grammars is still a point of reference for them, because it restricts the problems that are inevitably raised by natural language, such as ambiguity and unknown input, by a strict formal description of the shape of allowed structure. Context-free parsing also provides a heritage of algorithms for efficient implementations. In this chapter, we will introduce context-free grammars and their probabilistic version along with methods to parse with these grammars. We will also briefly relate methods for probabilistic context-free parsing to more powerful parsing methods.

### 2.1 Context-Free Grammars

A context-free grammar defines a formal language. A formal language is a possibly infinite set of strings of symbols. These strings are called sentences, and the atomic symbols that make up these sentences are the words of the language.<sup>1</sup> The grammar licenses those sentences that are part of the language, and at the same time assigns some structure to each sentence.

More formally, a context-free grammar (CFG)  $G$  is defined as a four-tuple consisting of a finite set of non-terminal symbols  $\mathcal{N}$  from which the structure is built, a finite set of terminal symbols  $\mathcal{T}$  with  $\mathcal{T} \cap \mathcal{N} = \emptyset$  that corresponds to the words of the language, a unique start symbol  $S \in \mathcal{N}$  that is the root of

---

<sup>1</sup>We prefer to call the strings *sentences* instead of the equally common term *word*, to stress the status of words (otherwise: letters) as atomic symbols throughout our experiments.

all structures assigned to the sentences, and finally a finite set of rules  $\mathcal{R}$  that expands some non-terminal node to a sequence of one or more non-terminal or terminal symbols:  $\mathcal{R} \subset \{A \rightarrow \alpha \mid A \in \mathcal{N}, \alpha \in \{\mathcal{N} \cup \mathcal{T}\}^*\}$ , in short:

$$G = (\mathcal{N}, S, \mathcal{T}, \mathcal{R})$$

A CFG can be used to both recognise and generate strings of a language. For generation, the start symbol  $S$  is expanded by one of the rules  $(S \rightarrow \alpha) \in \mathcal{R}$ , and the expansion continues recursively for all non-terminal symbols in the right-hand side of each applied rule, eventually generating a sequence of terminal symbols, i.e. a sentence.

The relation  $\overset{G}{\Rightarrow}$  specifies a context-free rewriting of one string into another along the lines of multiple rule application. We usually just write  $\Rightarrow$  when the grammar  $G$  can be inferred from the context. Given a rule

$$A \rightarrow \beta$$

from  $\mathcal{R}$ , and  $\alpha, \gamma \in \{\mathcal{N} \cup \mathcal{T}\}^*$ , then

$$\alpha A \gamma \Rightarrow \alpha \beta \gamma$$

means that the string  $\alpha A \gamma$  is *rewritten* into the string  $\alpha \beta \gamma$  using the rule  $A \rightarrow \beta$ . A finite sequence of these rewritings is called *derivation*, e.g.

$$\alpha_1 \Rightarrow \alpha_2, \quad \alpha_2 \Rightarrow \alpha_3, \quad \dots, \quad \alpha_{m-1} \Rightarrow \alpha_m$$

derives  $\alpha_m$  from  $\alpha_1$ .  $\overset{*}{\Rightarrow}$  is the reflexive and transitive closure of  $\Rightarrow$ . If  $\alpha_1 \in \{\mathcal{N} \cup \mathcal{T}\}^*$ , and all rewritings use rules from  $\mathcal{R}$ , then also  $\alpha_m \in \{\mathcal{N} \cup \mathcal{T}\}^*$ . Consequently, all derivations that start with the grammar's start symbol  $S$  and derive strings consisting only of terminals

$$S \overset{*}{\Rightarrow} \alpha, \quad \alpha \in \mathcal{T}^*$$

describe exactly the strings of the language that is defined by  $G$ .

Parsing means that for a given sentence, the combination of rules is determined that generates this sentence; or all distinct combinations, if there is more than one way to generate it. The CFG's output for such an input string is twofold. First, the CFG either succeeds, or fails to accept the input string, i.e. it recognises the input sentence as being part of the language defined by the grammar or not. Second, it assigns one or more parse trees to the sentence in case it is part of the language defined by the CFG, i.e. it analyses the structure of the input sentence. Parse trees are equivalent to derivations for CFGs when a unique order of rewriting is used, e.g. by always

expanding the left-most non-terminal symbol at the frontier of a partially completed parse; otherwise there could be several derivations that lead to the same parse tree.<sup>2</sup> Each production  $A \rightarrow \beta$  that takes part in a derivation by rewriting  $\alpha A \gamma \Rightarrow \alpha \beta \gamma$  corresponds to a *local subtree*.<sup>3</sup> The right-hand side  $\beta$  of the production is a sequence  $X_1, X_2, \dots, X_n$  of terminals and non-terminals. The resulting local subtree then consists of a top node labelled  $A$ , which dominates a sequence of children labelled  $X_1, X_2, \dots, X_n$ . The local subtrees are the building blocks of CFG parses, and it will become relevant later that these local subtrees can transport information to the rest of the parse tree only via the category of  $A$  (going up) and  $X_1, X_2, \dots, X_n$  (going down), all of which can only take one of a finite number of values.

Parse trees (or parses for short) are normally used to represent constituent structure in linguistic analyses, which is what we will do here as well. When  $\mathcal{T}$  is the set of words occurring in a natural language, and  $\mathcal{N}$  is the set of categories for syntactic constituents of that language, then the CFG provides syntactic analyses for sentences of this natural language. Each sentence  $w$  is assigned the set of parses  $\Psi_G(w)$  by the grammar  $G$ , which can have zero or more elements (again, we usually just write  $\Psi(w)$ ).

We do not give pre-terminal nodes a special status, which naturally map to parts of speech when terminal nodes correspond to the words of the natural language recorded in our treebank. Most of our languages use parts of speech as terminal symbols instead. Even when we assign parts of speech as part of the parsing process (i.e. as pre-terminal nodes), we do not handle them any different than other non-terminals. We do not use empty productions, which corresponds to the requirement that the sentences have at least one word, or that our treebank does not have sentences without annotation. These restrictions are arbitrary and do not diminish expressive power of conforming context-free grammars: avoiding  $\epsilon$ -productions does not cost any generative power, as long as the language does not contain the empty string (Hopcroft and Ullman, 1996, p. 97).

## 2.2 Probabilistic Context-Free Grammars

For more complex grammars, there will frequently be several parses for one input sentence, i.e. the sentence will be ambiguous with respect to the grammar. A *probabilistic* context-free grammar (PCFG) extends a CFG with probabilities assigned to each rule in  $\mathcal{R}$ . The probabilities  $P(A \rightarrow \beta | A)$  describe how probable it is that the left-hand side  $A$  of the rule expands to the right-

---

<sup>2</sup>The frontier is the set of nodes without any children.

<sup>3</sup>Rules are also often called *productions*. We will use both terms interchangeably here.

hand side  $\beta$ , given the left-hand side  $A$ . The probabilities of all rules with the same left-hand side in the set of rules  $(A \rightarrow \beta) \in \mathcal{R}$  have to sum up to one in order to form a proper probability distribution over all sentences generated by  $G$ :<sup>4</sup>

$$\sum_{A \rightarrow \beta, \beta \in \{\mathcal{N} \cup \mathcal{T}\}^*} P(A \rightarrow \beta) = 1$$

The probability of a parse generated by a PCFG is obtained by multiplying the probabilities of all rules involved in the parse. A derivation corresponds to a series of rule applications, i.e. a parse tree  $\psi$  is obtained by applying rules  $r_1, r_2, \dots, r_n$ . The probability of  $\psi$  is then obtained as the product of the probabilities of the rules in its derivation:

$$P(\psi) = P(r_1) P(r_2) \cdots P(r_n)$$

If the rules are grouped by their types, and  $Count_\psi(r)$  gives the number of times rule type  $r$  occurs in parse  $\psi$ , then the probability of the parse is:

$$P(\psi) = \prod_{r \in \mathcal{R}} P(r)^{Count_\psi(r)} \quad (2.1)$$

The assumptions that have to hold so that calculating probabilities in this way is justified have been summarised as three kinds of independence assumptions for the probabilities of subtrees (Manning and Schütze, 1999, p. 384). First, the probability of a subtree depends only on the string of words that it dominates, and not on where these words occur in a sentence (place invariance). Second, the probability of a subtree only depends on the words it dominates, and not on the words in the sentence that it does not dominate (context-freeness). Last, the probability of a subtree only depends on the non-terminal nodes inside of the subtree, and not on those outside of it (ancestor-freeness).

If there are several parses for an input sequence, then each parse is assigned a probability, which establishes an order among the parses. We interpret the probability as the quality of a parse, so that the most probable parse is the parse that describes syntactic structure of the input sentence best according to the PCFG.<sup>5</sup> We will see in section 5.1 how rule probabilities can be estimated from treebanks and assume for the moment that they are chosen appropriately. We always use plain PCFG grammars for our experiments

---

<sup>4</sup>In fact, an additional prerequisite is that parse trees of infinite depth must not occur (Prescher, 2002, p. 44). This constraint is always met by the treebank grammars which we consider for all our experiments (Chi, 1999).

<sup>5</sup>We will always ignore ties of several equally probable best parses and assume that our parser randomly chooses one of them.

and do not introduce lexicalisation as more complex conditioning on lexical heads that are marked explicitly in the rules of the grammar.<sup>6</sup>

The idea of allowing ambiguity in a PCFG model of natural language is, of course, that genuine ambiguity in natural language is reflected by ambiguity in the PCFG model, and that the probabilities in the PCFG model reflect the preferences for the different readings in their natural-language context. A PCFG conditions the probability of the expansion of a node only on the node's label, so that decisions for putting any local subtree in place rest on the subtree's top node. The question immediately arises whether the limited expressive power of a PCFG caused by these strong independence assumptions is sufficient to model relevant context in natural language. Consider e.g. the sentence *Die Schülerin kennt die Lehrerin* ('The pupil knows the teacher') that could also be translated as 'The teacher knows the pupil' if the pupil is contrasted with someone else that the teacher knows. The reason for this ambiguity is that constituent order is relatively free in German, and in the example both subject and object are ambiguous between accusative and nominative case. The default position of the subject is before all parts of the verb, however. The context that encodes this default position of the subject is different from the context that is necessary to trigger a non-default reading, which probably needs to reach beyond even the border of the example sentence. It becomes evident that the *context-freeness* of PCFGs may prevent the inclusion of the context that is necessary to parse correctly, given the more general kind of context that triggers the different reading. To what extent this is the case will be the main topic of this thesis.

## 2.3 Parsing with Context-Free Grammars

The actual task of determining the rules from a CFG that can be used to parse an input sentence, and to construct the corresponding parse, or parses, is performed by a *parser*. The Earley parser is a well-known and still widely adopted approach to parsing, using a chart of items that represent possible constituents of a parse, from which full parses can be read. The chart is filled bottom-up, restricted by a top-down filter that only allows items to be added which may eventually lead to a full parse (Earley, 1970). The Earley parser needs at most cubic time with respect to sentence length, and a chart with a size that is at most quadratic relative to sentence length (Grune and Jacobs, 1990, p. 155). This means that the time it needs to find all parses for a sentence grows less than exponentially with respect to the number of words in the input sentence. The time required by a straightforward parser that

---

<sup>6</sup>See however an alternative approach to lexicalisation in section 8.3.

builds a chart bottom-up without filtering would be exponential, matching all the rules' right-hand sides first against the terminals, and then against the sequences of terminals and non-terminals that result from the rules applied so far, until eventually a rule with the start symbol on the left-hand side has all remaining symbols on its right-hand side (e.g. the straightforward Unger parser, Grune and Jacobs, 1990, p. 82ff.). In practice, time complexity is often lower than cubic for the Earley parser, depending on the ambiguity of the grammar. There are also CFG parsers that have a guaranteed linear time complexity. They do, however, generally require the grammar to be unambiguous (among other restrictions on the grammar), which is unlikely for large-scale grammars as expected in the current context. The Earley parser needs at most quadratic time for such grammars, and linear time for those grammars where a linear-time method would be applicable. More sophisticated approaches exist that tend to achieve almost linear time more often, but they require more preprocessing, making them not necessarily more efficient and convenient to use when grammars change frequently (Grune and Jacobs, 1990, p. 250).

It is not known beforehand if the grammars that shall be used in the current context have certain properties that would allow the application of parsing methods that have a guaranteed worst case linear, or quadratic, time complexity. It is quite likely that the grammars will be ambiguous, so that parsing methods with linear complexity cannot be applied. A grammar derived from the Penn Treebank (Marcus *et al.*, 1993), e.g., is quite ambiguous. According to Klein and Manning (2001b), almost any non-terminal can be built over almost any span of terminals for such a grammar, so that a flexible yet rather efficient approach such as the Earley parser seems to be a good choice for parsing with such a grammar.

## 2.4 Parsing with Probabilistic Context-Free Grammars

Without imposing restrictions on the grammar, a parser cannot be guaranteed to enumerate all parses in less than cubic time relative to sentence length. In the current context, however, we focus on PCFG parsing, where our primary interest is to determine the best of all parses, namely the one with the highest probability. There is a standard algorithm to find such a parse more efficiently than by computing the probabilities for all parses generated by a CFG parser and then selecting the parse with the highest probability. The idea behind this algorithm is that each non-terminal dominating a sub-



tree in a derivation is the only link between this subtree and the rest of the tree, which is of course due to the context-freeness assumption on which the grammar is based. The maximum probability of the subtree can be calculated once and stored in a chart simply from the most probable combination of a rule expanding the subtree's mother node and the probabilities of subtrees that are dominated by its immediate children.

More formally, our goal is to determine the most probable parse  $\hat{\psi}$  from all parses  $\Psi(w)$  for a given string  $w$  licensed by our grammar  $G$ , i.e.

$$\hat{\psi} = \arg \max_{\psi \in \Psi(w)} P(\psi)$$

We will now assume a grammar in Chomsky Normal Form, i.e. a grammar with rules either of the form  $A \rightarrow BC$  or  $A \rightarrow a$ , with  $a \in \mathcal{T}$  and  $A, B, C \in \mathcal{N}$ .<sup>7</sup> We write  $w_{i,j}$  for a continuous substring of  $w$  and let  $i, j$  point between words, where  $i$  points into the gap before word  $i$ , so that  $w_{0,n+1}$  is equivalent to the full string  $w$ , when  $n$  is the index of the last word:  $n = |w| - 1$ .

The chart that is used for parsing consists of entries of the form  $[A, i, k]$ . Each entry holds a four-tuple of the form

$$(P(A \xrightarrow{*} w_{i,k}), B, C, j)$$

to remember the children of the non-terminal  $A$  that maximise  $P(A \xrightarrow{*} w_{i,k})$ , with  $0 \leq i < k \leq n$ , along with the actual probability of the whole subtree headed by  $A$ , and rooted in  $w_{i,k}$  (see figure 2.1). Chart entries for pre-terminal nodes only hold the probability that the pre-terminal expands to the terminal.

The algorithm first initialises the probabilities for the unary productions, which are simply the rule probabilities  $P(A \rightarrow w_{i,i+1})$  for each word,  $i = 0, \dots, n-1$ . Consequently, the chart entries  $[A, i, i+1]$  for each  $i = 0, \dots, n-1$  are filled with the probabilities

$$P(A \rightarrow w_{i,i+1})$$

An induction step then determines all other subtrees with highest  $P(A \xrightarrow{*} w_{i,k})$  bottom up, so that each entry of the chart is filled with the probability of the most probable subtree  $[A, i, k] = (P(A \xrightarrow{*} w_{i,k}), B, C, j)$  found via

$$\max_{(B,C,j)} P(A \rightarrow BC)P(B \xrightarrow{*} w_{i,j})P(C \xrightarrow{*} w_{j,k})$$

---

<sup>7</sup>Any context-free grammar has an equivalent context-free grammar in Chomsky Normal Form (Hopcroft and Ullman, 1996, p. 99).

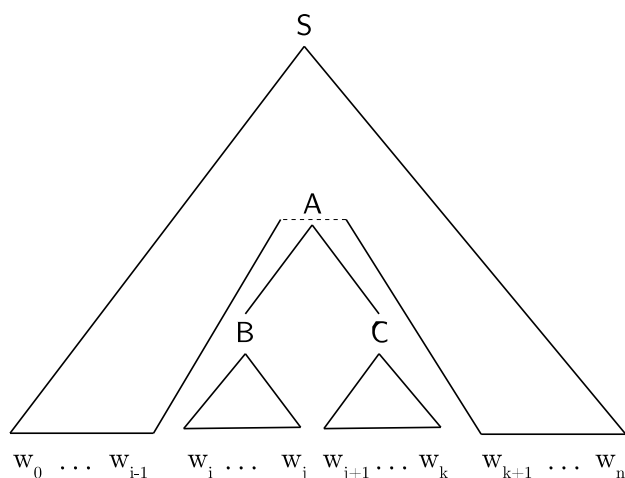


Figure 2.1: Best Subtree Spanning  $w_{i,k}$ :  $\max_A P(A \xrightarrow{*} w_{i,k})$

for the maximum probability and with:

$$\arg \max_{(B,C,j)} P(A \rightarrow BC) P(B \xrightarrow{*} w_{i,j}) P(C \xrightarrow{*} w_{j,k})$$

for the corresponding categories of the children  $B, C$  of  $A$ , plus the marker  $j$  for the words they span.

When the induction terminates, the most probable parse is read from the chart by starting with the cell that holds the grammar's start symbol  $S$  and that spans the whole sentence, i.e. entry  $(S, 0, n + 1)$ . You then read the children  $B, C$  from it, and the parts of the sentence they span, determined by  $j$ . The full most probable parse can be read from the chart by continuing with the children  $B$  and  $C$  along the same lines. The resulting parse is called the *Viterbi parse*. The algorithm is a variation of a dynamic-programming algorithm known as the *inside algorithm*. It determines the Viterbi parse with very small overhead on top of building the chart (Manning and Schütze, 1999, p. 396ff.).

## 2.5 Efficient Viterbi Parsing

A more efficient solution for finding just the most probable parse than we have just outlined is *A\* parsing*. The inside algorithm requires a full chart, because only when the root node is assigned its most probable dominated subtree, the set of edges (i.e. pointers between items) in the chart can be selected that represent this subtree, and all following subtrees that eventually lead to the most probable parse. This is so because the Viterbi parse does

not necessarily consist only of most probable local subtrees, because several suboptimal subtrees may yield a higher product of rule probabilities than a most probable subtree which can build a full parse only in combination with other low-probability subtrees. A\* parsing, in contrast, restricts the search for the most probable parse to a fraction of all edges, while it still guarantees that the Viterbi parse is found. An application of A\* parsing to English newspaper text shows that with only a minimal and fix additional time and space requirements, only 3% of the processing performed by a PCFG chart parser has to be carried out, resulting in parsing time of only a few seconds for an average-length sentence on current hardware (Klein and Manning, 2003a).

Implementations that derive the Viterbi parse via inside probabilities from a chart without guided search are not necessarily less efficient in terms of empirical run-time than those using A\* parsing. Schmid (2004) shows that choosing a compact representation for the edges of the chart can be similarly efficient, reducing the amount of time spent per edge instead of reducing the number of items processed. He also shows that PCFG Viterbi parsing can be implemented quite efficiently, and that it can be used to parse unrestricted natural language data in almost real time, also generating the most probable parses in only a few seconds on average.<sup>8</sup>

We are mainly interested in the impact of the representation of linguistic annotation on parsing performance when following the assumptions of probabilistic context-free parsing. We therefore employ straightforward PCFG estimation and Viterbi parsing, without any smoothing or backing off. This is not to say that both could not improve performance, but they would affect the impact of changes in representation on performance. A positive consequence is that the choice of parsers should not affect the results of our experiments. We use the implementation of `lopar` for determining the Viterbi parses in all of our experiments (Schmid, 2000). We also add evaluation on other consequences of the language models, most prominently the average per-word perplexity (section 3.3). `BitPar` is a parser that is extraordinarily efficient in calculating Viterbi parses, which is very similar to computing inside probabilities (Schmid, 2004) and does not require enumerating all parses. We therefore use `BitPar` to compute inside probabilities, which for the grammar start symbols equals the sentence probability. `BitPar` is a rather general CYK parser (Schmid, 2004), and `lopar` implements a left-corner parsing scheme (Schmid, 2000). Parsing performance is quite different for `lopar` and `BitPar` Viterbi parsing in terms of memory and time requirements. `lopar` manages to

---

<sup>8</sup>This contrasts sharply with one hour needed for nine words, as reported about a decade ago in an early PCFG treebank grammar experiment (Krotov & al., 1994).

parse all sentences of our smaller tune test data set with the amount of memory available on our test system.<sup>9</sup> It comes close to using all of it and needs about 15 seconds per sentence (depending on the grammar). `BitPar` uses less than 60MB of memory under the same conditions, and parses at more than one sentence per second (preliminary results for sentences of up to 40 words; more than two sentences per second for some grammars). These differences in parsing performance had an impact on our choice of parser for the different experiments. `BitPar` is used for the experiments on the larger final data sets, and also for all experiments on sentences of unrestricted length, which cannot be performed by `lopar` due to memory restrictions. Both `lopar` and `BitPar` originally only output the probabilities of the Viterbi parses when used to generate the parses, which is insufficient for calculating the corpus perplexity. We chose `BitPar` for a second go on all experiments in which we calculate corpus perplexity. Both `BitPar` and `lopar` had to be patched to compute the corpus perplexity and full number of parses, respectively (which often exceeds the domain of the standard C language's data type originally employed in `lopar`).<sup>10</sup>

The efficiency of Viterbi parsing comes at the price of limitations in descriptive power. (P)CFGs describe the class of context-free languages, which has been extensively studied in formal language theory. It includes all sentences of the regular languages, but cannot describe languages with unrestricted complexity. Natural languages do not strictly belong to the class of context-free languages. It has been shown that there are syntactic phenomena in some natural languages that cannot be adequately described by CFGs, which seems to render them poor candidates for describing the syntax of natural language.<sup>11</sup> In the current context we will take treebanks as the source of syntactic description for a language. CFGs are usually not capable of reproducing all information in a treebank straightforwardly either, but they do seem to be able to describe adequately the majority of it when appropriately applied, which is indicated by experiments performed e.g. by Klein and Manning (2003b) and by ourselves below. The advantage of restricting expressiveness is that PCFGs allow for efficient implementations that provide most probable parses of unrestricted natural language data, which we hope has become apparent by now. We will discuss in chapter 5 how a treebank can be used to determine the probabilities of the rules of a PCFG model.

---

<sup>9</sup>A 32bit Debian Linux Opteron 246 with 8GB memory, which has slightly more than 3GB of memory available per process. See section 5.2.3 for a description of the data sets. We do not use cut-offs for low-frequency rules, so that we always use all rules.

<sup>10</sup>Thanks to Helmut Schmid for making the source code of both parsers available, along with helpful comments. We use `lopar` version 3.0 and a `BitPar` version of July 29, 2004.

<sup>11</sup>Most notably cross-serial dependencies exceed context-free power (see section 7.1).

## 2.6 Alternative Approaches to Probabilistic Parsing

A considerable amount of research has focused on approaches to probabilistic parsing that do not limit the expressiveness of the parsing model in advance, thus exceeding context-free power. None of them meets the efficiency of PCFG parsing, while many practically deployable models have appeared recently.

Malouf and van Noord (2004) describe a parser that applies a hand-built attribute-value grammar to unrestricted text. It employs the maximum-entropy framework that allows specifying arbitrary overlapping and non-local features on the parses. The model is not trained directly on the manually annotated parses of a treebank, because the kind of parses available in their treebank cannot be cast directly into parses of their grammar. An exhaustive search through all parses generated by the model is too inefficient, either, so that the model is trained on a subset of parses that impact the model parameters depending on how well the grammar's parses map to the treebank parses. When applying the resulting model for parsing, Malouf and van Noord employ beam search over components of parses in order to deliver parses within a few seconds. In the end, they show that parsing with more expressive probabilistic grammar formalisms is indeed possible by applying optimisations to the training and parsing stages, which reduce time and memory requirements of large-coverage parsing with only a small impact on accuracy.

Other more powerful approaches like Data Oriented Parsing (DOP) also require optimisations in order to handle the large number of parses that natural language grammars often have to handle. Sima'an (1996) shows that determining the most probable parse for such a stochastic tree substitution grammar is NP-hard, i.e. there is no guaranteed upper bound on the number of parses, which is smaller than all parses, that you need to search until you find the most probable parse. The standard way to generate the probability of a parse according to the DOP model is to sum over all derivations leading to the parse, and DOP considers exponentially many of them. As a result, parsing requires time that is exponential in sentence length. As the number of different subtrees also grows exponentially with the length of the training corpus, memory requirements are also exponential. There have been several approaches to reducing the complexity of the DOP model in practice, most notably the approximate sampling methods employed by Bod (1995) to determine the most probable parse, or using the most probable derivation as the best parse, which can be found using the Viterbi algorithm (though with some impact on accuracy; Bod, 2000). Even more interesting in the

current context is the approach of Goodman (1996a, 2003) to recast a DOP model as a PCFG that assigns the same sets of parse trees with the same probabilities as the corresponding tree substitution grammar, but which only grows linearly in size with the training data. Goodman (2003) also proposes a new parse selection criterion that reduces the search complexity for finding the best parse to  $O(n^3)$ , i.e. to the same complexity as is generally required for PCFGs. The equivalence between an appropriately chosen PCFG and the DOP model seems to be counter-intuitive at first, given the power of a model that allows the combination of arbitrary subtrees from training data to assemble parses. The equivalence highlights a PCFG's ability to take considerable, even though theoretically limited context into account. It also shows how PCFG techniques find their way into more powerful parsing models.

The efficiency of PCFG Viterbi parsing thus also appeals where the main objective is to apply more powerful language models. Such language models allow the definition of many non-local features that are not necessarily independent, without the burden of encoding them in a generative probabilistic model. As a downside, they are not efficient enough to determine the best parse from a large number of parses. Less complex models, however, usually assign all their parses a rank, and it has been observed that you often find the best parse among a small fraction of parses with high ranks, even though these models fail to select the best parse reliably (Collins, 2000; Ratnaparkhi, 1999). While only Collins (2000) is based on a generative model that vaguely resembles a plain PCFG (that from Collins, 1999), one can hope that adequate plain PCFG models may be employed along the same lines when the number of parses becomes too large, and efficient pre-processing is indispensable.

Taskar *et al.* (2004) even show that feature-based discriminative methods may consider the full set of parses in a similar framework as used by Collins (2000) to rerank only the pre-selected top fraction of all parses. They use Klein and Manning's (2003b) plain PCFG model as a baseline, and observe very similar performance when both models have the same kind of limited access to lexical information. The discriminative model only improves considerably on this score when additional features are specified that are not trivial to incorporate into generative models, like the word preceding or following a constituent. Specifically, the final model also scores above Collins (1999) on the chosen data. It performs parsing in cubic time, and only has a more costly training phase as a major downside. We would like to note that the plain PCFG developed in Klein and Manning (2003b) has been chosen as a starting point. In the present context we rather focus on ways to improve the structure of the underlying grammar along the lines of Klein and Manning than on the parser that is used to apply it, as do Taskar *et al.* We restrict

---

ourselves to plain PCFG parsing for the evaluation of our transformations. The fact that the powerful discriminative method introduced by Taskar & al. performs similar to a PCFG model under comparable conditions does not mean, of course, that it will suffer in the same way as a plain PCFG does when transformations that optimise PCFG performance along the lines of Klein and Manning are absent. We expect, however, that it may be useful for more powerful disambiguation methods, too, to look for a grammar that represents the parsing problem most adequately. This problem of finding adequate grammars is our central question.

Plain PCFGs, albeit limited in power, will serve as our parsing model, because it seems to be rather clear what their main deficiencies are, and we hope it has become clear that their specific combination of power and efficiency still makes them attractive even for current approaches to parsing.





## Chapter 3

# Parser Evaluation

Parser Evaluation tries to assess the performance of a parser. It is at the core of parser development, because only via evaluation is it possible to compare alternative approaches to parsing, or to ensure that changes in a parser during development do not degrade performance. Evaluating the performance of a parser, or equivalently of a PCFG's language model, is not trivial. Ideally, such a model should be evaluated by a method that states in absolute terms how good it is. It is, however, not known what exactly constitutes language performance, and therefore such an evaluation method does not exist. It should be feasible, though, to assess the performance of a language model relative to a perfect model, i.e. that of the human mind. This idea is the basis of the Turing Test, which proposes to assign the role of a human being to the model to be evaluated, so that our model can be tested via its performance relative to another (perfect) model, i.e. relative to a human being (Turing, 1950). This test setup, however, assumes that we have a language model at hand that is capable of a lot more than just performing syntactic analyses. A syntactic parser does not attempt to provide full analyses of utterances, including semantic and pragmatic interpretations (although those aspects may turn out to be relevant for best performance). A syntactic parser only attempts to assign syntactic structure to utterances. Lacking a lot that makes it a fully workable partner in a dialogue, a syntactic parser cannot be evaluated directly along the lines of the Turing Test.

The assumption underlying the definition of the task of syntactic analysis is that there is such a thing as syntactic structure. Given this assumption, it is therefore appropriate to evaluate the ability of a syntactic parser to assign syntactic structure. Strings of symbols generated by a perfect model (i.e. the mind) are available in collections of utterances, i.e. corpora, but syntactic structure is not. This is, of course, where treebanks fit in, which represent assumed syntactic structure for observed utterances. The evaluation task thus

has become considerably easier: instead of looking for a function that measures the absolute performance of a parser, we are now looking for a function that compares syntactic structure that a parser assigns to given utterances with the structure that is assumed to properly represent the syntactic analyses of these utterances. The latter is brought to us in the *gold* annotation found in treebanks.

Existing treebanks differ in the kind of syntactic structure that they specify. The structural information that they assume always seems to be some kind of directed acyclic graph with labelled nodes (Bird and Liberman, 2001). Often the graphs are further constrained, e.g. to represent constituent structure. Not all treebanks base their analyses on constituents, however, which imply intermediate nodes that do not correspond directly to items observed in the utterances recorded in corpora. Syntactic analyses encoded strictly according to dependency analyses of syntax (like the PDT, Böhmová & al., 2003) do not assume those intermediate nodes. Other treebanks still follow some of the ideas of dependency analyses, but do so using intermediate nodes (like negra, Skut & al., 1998). Still others primarily focus on constituent structure, but still allow for a dependency view on their analyses, by employing heuristics that map one kind of analysis to another (including TüBa-D/Z and the Penn Treebank; Bosco and Lombardo, 2004; Kübler and Telljohann, 2002).

Given the variation in assumed analyses in treebanks, the question arises how structure assigned by a parser can be compared to structure given in a treebank if the underlying representation of syntactic analyses differs between them (e.g. comparing constituent-based parses against the analyses in a dependency treebank). We assume that this question does not only appear when parser and treebank are incompatible in the first place. Even when parser and treebank provide similar (e.g. constituent) analyses, the different views on syntax (e.g. assuming constituency or dependency) are equally important. Even more so because each view on syntactic analyses has its own set of evaluation metrics with its flaws and merits, and because there seems to be no single metric that reliably represents the quality of a syntactic analysis. The observation that a parser can be tuned to optimise certain metrics, but usually only at the expense of performing worse at others (Goodman, 1996b) suggests that applying several well-motivated metrics may come closer to assessing the actual goodness of a parser, which is a quality we cannot judge directly.

A scaled-down version of modelling all aspects of linguistic performance requested above could be the degree in which the parser helps solving another problem for which better evaluation metrics are available. Such an evaluation *in vivo* is quite convincing, because it measures the amount of success of a

certain application when the parser supports this application. We are more interested, though, in a parser that is not yet specialised, and seek to optimise it in the most general way. We therefore stick to evaluation *in vitro* and choose to apply a number of metrics to evaluate the general quality of parses provided by the parser.

### Perfect Match and Coverage

The *full match* (*fm*) or *keywordperfect match* metric, and the *number of unparsed sentences*, *failed parser coverage* are rather straightforward and uncontroversial, but they are also rather coarse metrics. If the syntactic analysis of a parser for a given utterance corresponds fully to the analysis given in the treebank, then there is no doubt that the analysis is perfect. The full match metric gives the proportion of perfect analyses provided by the parser relative to all tested sentences. Given  $\psi_T(w)$  is a gold parse for a sentence, and  $\psi_P(w)$  is the parse proposed by the parser, then the full match ratio for the strings  $\mathcal{W}$  in the treebank is

$$fm = \frac{|\{w \mid \psi_T(w) = \psi_P(w), w \in \mathcal{W}\}|}{|\mathcal{W}|}$$

Obviously, this metric is rather coarse, because it does not consider the degree in which the parses are incorrect, if they are not fully correct. If the parser returns parses that all are minimally different from the gold parses, the full match ratio would return its worst verdict of 0% success. The *failed parses* (*failed*) metric analyses the other end of perfection. There may be utterances where the parser not only presents a wrong parse, but where it is not able to present any parse at all. The number of those cases is recorded with this *failed* metric.<sup>1</sup> It corresponds to the coverage, which specifies the fraction of all sentences for which the parser provides some analysis.

If the analysis provided by the parser is not identical to the analysis presented in the treebank, then there is usually still some overlap between them. The major question that remains to be answered is thus how big the penalty should be for deviations from the gold parse. This question turns out to be rather difficult to answer. A penalty on the quality of the parse may be zero if the alternative analysis offered by the parser is equally acceptable, e.g. because the utterance is inherently ambiguous, or the ambiguity does not matter for the sake of the application that requests the parse. A penalty may be very large if the wrong reading has been chosen for a sentence with two very different readings.

---

<sup>1</sup>We choose to use absolute numbers instead of the standard ratio, because most models that we examine fail only on very few sentences.

The metrics we will use to evaluate the gradual overlap between the analysis returned by the parser and the gold parse focus on either constituent, or dependency structure. A third kind of metric takes advantage of PCFGs being probabilistic language models, and assesses differences between what the model predicts and what is found in a corpus. We also report other details on experiments that we think do not need extensive discussion, most prominently the number of rules in a specific grammar. We have pointed out already that generally, fewer rules seem preferable, but it may also be useful to have more rules in a CFG if they encode more global constraints (Yoshinaga & al., 2003).

### 3.1 Parseval

With the advent of treebanks, it was for the first time possible to compare automatically the output of syntactic parsers with what would be considered as the perfect syntactic analyses for the same input, constituting an ideal resource for comparing different parsers, or measuring the impact of changes in a single parser. Parsers before had of course been developed independently of these new resources, making such comparisons rather difficult, because identical analyses would often be expressed by different tree structures, and identical categories by different labels. Black & al. (1991) therefore proposed to first strip the labels off the tree, and then transform parser output, but also gold data, so that both agree for identical analyses. They further specified metrics that would give the degree of correspondence between both if they would not be identical, based on the sets of tuples of start and end positions of each constituent in the gold data (the key  $\mathcal{K}$ ) and the parser's output (the answer  $\mathcal{A}$ ). *Recall* is then called the fraction of the number of correct constituents returned by the parser divided by the number of truly correct constituents ( $\frac{|\mathcal{K} \cap \mathcal{A}|}{|\mathcal{K}|}$ ). If the parser returns constituents not in gold data, then they could either be compatible with the gold parse, i.e. they could both be part of a single proper tree, or they could be incompatible. The *crossing brackets metric* ( $CB$ ) counts those constituents that are incompatible. Given a constituent in gold data is represented by the tuple  $(f, l)$  representing the position of the first word that it dominates, and the position of the last word, and some constituent in the parser output is represented accordingly by  $(f', l')$ , then these two constituents represent a pair of crossing brackets if for any  $(f', l')$  there is some  $(f, l)$ , so that  $f < f' \leq l < l'$ . Harrison & al. (1991) present an implementation of these metrics in the Parseval program and propose to measure also the *precision* of the parser output, which is the proportion of correct constituents from all constituents in the parser's

output,  $\frac{|A \cap K|}{|A|}$ . Additionally, they propose to normalise the absolute number of crossing brackets as two values: the ratio of all sentences that do not contain crossing brackets at all, called *zero crossing brackets* (0 *CB*), and the average number of crossing brackets per sentence (the *CB* rate).

The idea behind using unlabelled constituents is that a parser that has been developed independently of the resource against which it should be tested often would not use the same labels to express the same categories. Grishman & al. (1992) address this problem by specifying tree transformations that undo the systematic differences between the structure assumed in the treebank and the structure assumed by the creators of the parser. Their goal is to keep the manually devised data unchanged, so that it represents a proper (unmodified) gold standard that stays the same for all parsers that use it, and they indeed show that their parser can incorporate alternative preferences that make it perform better on the Parseval metrics, and they also add robustness rules that allow parsing increasing amounts of gold data without diminishing performance.

Parseval metrics in this last incarnation see a treebank as a set of unconnected nodes. They understand each node in a treebank’s annotation as a triple of start position, end position, and label. A triple that is found in gold annotation and in the parser’s output is correct, and incorrect if not found otherwise. Given the set of all triples in gold annotation is  $K_L$  and the set of triples provided by the parser is  $A_L$ , then Parseval also defines labelled precision and recall on these sets:

$$prec_{lab} = \frac{|A_L \cap K_L|}{|A_L|}$$

$$rec_{lab} = \frac{|A_L \cup K_L|}{|K_L|}$$

Precision and recall are also combined in a geometric mean, where a weighting factor  $\alpha$  is standardly set to 1/2 so that both have equal impact on the overall so-called F-measure (presented by van Rijsbergen, 1979, p. 134, as *effectiveness measure*  $E_\alpha = 1 - F_\alpha$ ):

$$F_{\alpha=1/2} = \frac{1}{\alpha \frac{1}{precision} + (1 - \alpha) \frac{1}{recall}} = \frac{2 precision \times recall}{precision + recall}$$

so that our labelled F-measure is

$$F_{lab} = \frac{2 prec_{lab} \times rec_{lab}}{prec_{lab} + rec_{lab}}$$

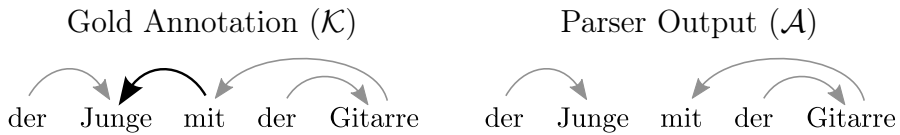
We do not see a reason to prefer either precision or recall, so that we always use  $F_{\alpha=1/2}$ , or  $F$  for short. An alternative formulation uses a parameter  $\beta$ ,

where  $F_{\beta=1} = F_{\alpha=1/2}$ . The same set of metrics for precision, recall, and weighted average are also specified for the unlabelled start and end position tuples recorded in the gold annotation set  $K_U$  and the parser annotation set  $A_U$ , resulting in:

$$\begin{aligned} prec_{unl} &= \frac{|A_U \cap K_U|}{|A_U|} \\ rec_{unl} &= \frac{|A_U \cup K_U|}{|K_U|} \\ F_{unl} &= \frac{2 prec_{unl} \times rec_{unl}}{prec_{unl} + rec_{unl}} \end{aligned}$$

Certain requirements should hold for such a combined metric, e.g. that for any given value of precision, lower values of recall should always give lower values of the composite measure. Van Rijsbergen (1979) shows that  $F$  is such a metric. It is the harmonic mean of precision and recall, so that low values of both are punished, but also large differences between them, so that performance that does not neglect any of both is rewarded most.

When parsers were first evaluated with Parseval, they usually had not been developed with the annotation scheme of the gold parses in mind, so that dropping the labels, and generally transforming the parser’s output to comply with the gold data were useful, if not necessary steps. We do not require any mapping of labels or structure, because we are free to adopt the parser so that it produces parses that agree with our treebank’s annotation scheme. Even though our parser can be compared directly with test data, there are weaknesses of the constituent-based view on syntactic structure. Parseval metrics are based on constituents, which can vary in number without major impact on their interpretation. While their absolute number is hidden by the normalised precision and recall ratios, it can nonetheless influence evaluation. Depending on the number of constituents that are used to represent constituent structure, a single incorrect attachment decision can have varying impact on the metrics. Figure 3.1 shows the noun phrase *der Junge mit der Gitarre* (‘the boy with the guitar’), where the prepositional phrase is meant to modify the initial noun phrase *der Junge* as the correct solution ( $\mathcal{K}$ ), but a parser delivers a wrong parse and does not attach it ( $\mathcal{A}$ ). We assume two annotation schemes that take a different approach on representing embedded structure, and compare the impact of these representational differences on the Parseval scores. The first scheme uses one constituent per phrasal head (*Junge* and *Gitarre*), and lets the constituent extend and include all dependent constituents (figure 3.1(b)). This is an approach similar to the flat attachment strategy adopted in the negra treebank (Skut & al., 1998). The second scheme adds a new constituent for the combination of the



(a) idealised representation

$$\begin{array}{ll} \text{[der Junge [mit der Gitarre]]} & \text{[der Junge] [mit der Gitarre]} \\ \mathcal{K} = \{(1, 5), (3, 5)\} & \mathcal{A} = \{(1, 2), (3, 5)\} \end{array}$$

$$prec_{unl} = \frac{|\{(3,5)\}|}{|\{(1,2),(3,5)\}|} = 1/2$$

$$rec_{unl} = \frac{|\{(3,5)\}|}{|\{(1,5),(3,5)\}|} = 1/2$$

$$F_{unl} = 1/2$$

(b) negra style bracketing

$$\begin{array}{ll} \text{[[der Junge] [mit [der Gitarre]]]} & \text{[der Junge] [mit [der Gitarre]]} \\ \mathcal{K} = \{(1, 5), (1, 2), (3, 5), (4, 5)\} & \mathcal{A} = \{(1, 2), (3, 5), (4, 5)\} \end{array}$$

$$prec_{unl} = \frac{|\{(1,2),(3,5),(4,5)\}|}{|\{(1,2),(3,5),(4,5)\}|} = 1$$

$$rec_{unl} = \frac{|\{(1,2),(3,5),(4,5)\}|}{|\{(1,5),(1,2),(3,5),(4,5)\}|} = 3/4$$

$$F_{unl} = 6/7$$

(c) TüBa-D/Z style bracketing

Figure 3.1: Parseval is Sensitive to Bracketing Style

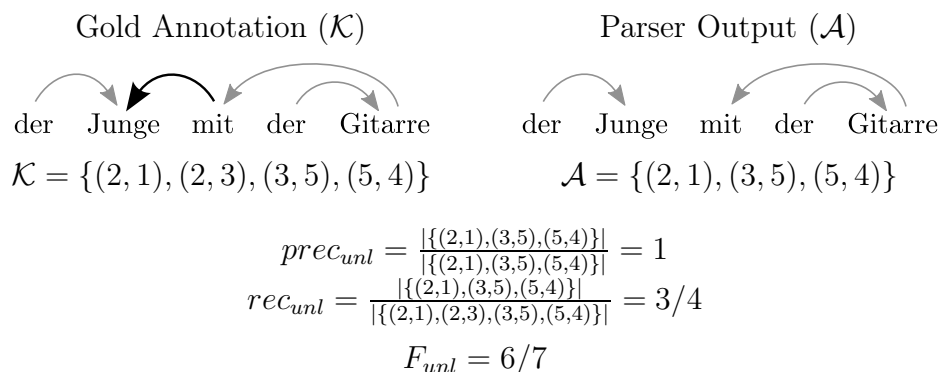
two phrasal heads instead (figure 3.1(c)), which is similar to the approach adopted in the treebank TüBa-D/Z (Telljohann *et al.*, 2004). The status of the constituents is obviously different. While the upper representation only shows maximal projections, the lower also shows intermediate projections. Failing to attach the prepositional phrase to the noun phrase has different impact on the Parseval evaluation of the parser error in the analyses on the two sides. In the negra style representation, the error is considered more severe than in the TüBa-D/Z style representation ( $F_{unl} = 0.5$  vs.  $F_{unl} = 0.86$ ). The unlabelled crossing brackets metric is similarly unbalanced: it counts a crossing bracket only in figure 3.1(b) ((1, 5) vs. (1, 2)). What is more, the absolute number of brackets is lower in the first case, so that a single crossing bracket has a relatively higher impact. The idealised representation given in figure 3.1(a) focuses on the single erroneous attachment decision. It shows directed arcs that link the heads of the constituents, which, however, are partly determined by the labels of constituents and edges, so that it is based on more information than is available in the unlabelled bracketings. It adopts a representation similar to dependency analysis of syntax.

## 3.2 Relational Evaluation

A weakness of the constituent-based Parseval metrics is that a single incorrect attachment decision can cause many brackets to cross. Given the above observation of impact that varies with the chosen (identically meaningful) representation, such differences seem to be based on rather unrelated characteristics of the data. Instead, an annotation scheme that focuses on the meaningful differences and represents them in a normalised way would be desirable. Lin (1995) proposes such a scheme, which is based on syntactic dependencies between words, i.e. on syntactic analyses that can be represented as trees without non-terminal nodes. The key and answer sets of parses are then expressed by tuples representing the positions of head and modifier ( $h, m$ ), where standardly a word can be a head to several modifiers, but a modifier only to a single head (an assumption broken in e.g. certain representations of coordinations, see below). The dependency metric is calculated on the basis of head/modifier tuples in much the same way as the Parseval metrics are derived from tuples representing bracketings. Figure 3.2 shows that the key and answer sets are equally intersected to determine recall and precision, and finally the weighted mean is computed, which we call the  $F$  score for dependency evaluation.

The major challenge when implementing such a relational evaluation scheme, of course, is the transformation of constituent-based analyses that



Figure 3.2: Computing the  $F$  score for dependency evaluation

are found in a treebank into dependency analyses, and the choice of the label set that is used to label the relations. Technically, the method is rather straightforward (Lin, 1995). Each non-terminal is assigned a head from one of its children, so that heads percolate up from the lexical level. Wherever a head does not percolate further up, because a sister head is selected as the head of its mother node, the lexical item representing the dominated node is a modifier to the lexical item of the dominating node, and a dependency link is added between the lexical items representing both heads accordingly. Thus determining the heads of constituents, and choosing a label for the resulting dependencies will determine the resulting dependency annotation. Fortunately, the treebank that we use for most of our experiments, TüBa-D/Z, provides a wealth of information concerning heads of constituents, so that rather few heuristics are necessary.<sup>2</sup>

Using a dependency representation of TüBa-D/Z annotation for evaluation is rather appealing, because of the advantages of dependency annotation over the Parseval metrics, and because the dependency representation can be deduced from TüBa-D/Z annotation rather straightforwardly. However, we cannot use dependency annotation to evaluate all our experiments, because only full annotation of TüBa-D/Z including edge labels defines the unambiguous mapping outlined above. It is not given for less rich models of TüBa-D/Z constituent structure that do not include edge labels. We will therefore only apply dependency representation to evaluate some experiments.

<sup>2</sup>Please see Kübler and Telljohann (2002) for more details on these heuristics.

### 3.3 Relative Differences in Information

PCFGs define a language model: for each sentence that is part of the language that they define, they specify how likely it is to occur. What makes probabilities useful for evaluation is that a language model should have better knowledge of the language when it assigns higher probabilities to sentences that are likely to occur in the language, and lower probabilities to those that are unlikely to occur. Given that we are talking about a treebank of sufficient size, we can hope that a PCFG derived from it comes close to modelling the natural language that it represents, i.e. that it assigns probabilities equally reliably to an unseen sentence taken from the treebank as to any unseen sentence drawn from a source other than the treebank.

A language that has only few different sequences of strings that occur very frequently is easier to predict than one in which all words occur completely at random. There is a bound on predictability, though: if each sentence in a language were completely predictable, then it would not convey any information. Thus even a very good language model (e.g. that of a human being) will not be able to predict perfectly the next sentence uttered by someone else, or otherwise the listener will be bored. The lower bound on the difficulty of predicting the next part of an utterance, or on the surprise when faced with it, is called the *entropy* of the language.<sup>3</sup> A model usually performs worse than perfect, and will add to this lower bound that is inherent in the language. The less entropy a language has according to a model, the better the model approaches this lower bound. This notion is captured in the *relative entropy* between a language model and a language, which we will use for evaluation, and for which we will show below how it can be computed between a PCFG and a treebank.

The entropy can be interpreted as the average number of yes/no-questions you would have to ask on average to guess the next corpus position (or the number of *bits* to encode the answer). The entropy  $H$  has a closely related measure called perplexity:

$$perp = 2^H$$

The perplexity thus gives you the average number of equally likely choices per corpus position, both times assuming a perfectly informed guesser that asks the minimal number of questions, i.e. always asks first for the choice that it thinks is most likely to occur. As an advantage over entropy, the notion of perplexity does not depend on a base of 2 that is arbitrarily chosen.

The cross-perplexity between a corpus  $c$  and a language model  $m$ , is given

---

<sup>3</sup>We base our overview on Prescher (2002) and Cover and Thomas (1991).

as follows:

$$\text{perp}_{\text{cross}}(c, m) = \frac{1}{\sqrt[|c|]{P(c)}}$$

where  $|c|$  is the length of the corpus in words, and  $P(c)$  the probability assigned to the corpus by the language model. As the cross-perplexity specifies the average number of equally likely alternatives per corpus position, it averages in two ways: First, not every word in a sentence will be equally difficult to guess, i.e. perplexity averages over the different corpus positions. Second, the probabilities of the different alternatives at an individual corpus position are unlikely to be distributed evenly over all alternatives, i.e. perplexity, when interpreted as the number of different alternatives, averages over the alternatives. Evidently, it will be more difficult to predict a sentence given the preceding sentence than to predict a word given the preceding word, which in turn will be more difficult on average than to predict the letter following a given letter. It is thus necessary to normalise the entropy with respect to the size of the predicted unit. Words offer a relatively well-defined unit which we will use whenever we talk about entropy from now on, especially because we have data sets which differ in average sentence length. It is also evident that the difficulty to predict the next word depends on the position in the utterance. It will be harder, e.g., to predict the first word of a sentence than to predict what follows as the dots in *sagte heute Bundeskanzler . . .* ('said chancellor . . . today') in a German newspaper article from early 2005, where chances are that *Schröder* will be uttered next. We are therefore determining the *average* per-word cross-entropy between a language and a language model.

As an example, a corpus consisting exclusively of the words  $a, b, c, d$  that are evenly distributed according to figure 3.3(a) faces you with four equally likely words at each corpus position. A corpus consisting of the same words,

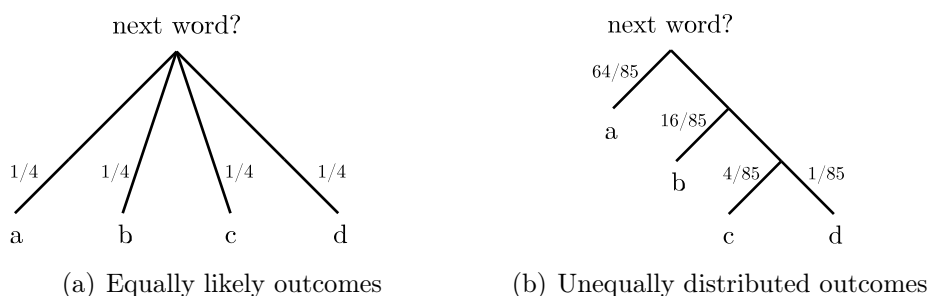


Figure 3.3: Decision Trees for Guessing the Next Word

but distributed as shown in figure 3.3(b), is easier to predict. A language

model that knows about this distribution and that is asked to guess each word will first ask for word  $a$ , and then for  $b$ ,  $c$ , and  $d$ , in this order. On average, this strategy makes the second corpus about as hard to predict as a corpus with two equally likely alternatives, corresponding to a cross-perplexity between the model just outlined and the corpus of 2.06.

The cross-perplexity between a language model and a corpus is determined by the two aspects already mentioned for the cross-entropy: first, how well the model fits the corpus, and second, how predictable the corpus is. Given a corpus consisting only of sentences  $abcd$ , you would never have to choose which word comes next – if you knew about it, which would also require a more powerful model than one that handles each word independently like the one shown in figure 3.3. A model that perfectly knows about the corpus would amount to a corpus perplexity of 1, i.e. the corpus being totally predictable, and also to a cross-perplexity of 1 between model and corpus. If the model’s assumption is that all words are equally likely (a language model corresponding to figure 3.3(a)), then the cross-perplexity of the model and this corpus is 4. If you assume an uneven distribution as in figure 3.3(b), then the cross-perplexity between this model and a corpus with evenly distributed words like in figure 3.3(a) increases to over 10. Language is unlikely to be totally predictable, because its purpose is to convey an informative message from speaker to listener. On the other hand, if there is something known about the corpus in advance, it becomes more predictable. All experiments partly know about the data to be predicted, because the training set which we use to predict the test set is quite similar to it, as both are taken from the same newspaper, and the same month of the year. For most experiments even more is known about the test data: the vocabulary is a part-of-speech tag set instead of the full range of possible German words. As also the size of the vocabulary determines corpus perplexity, we can use the cross-perplexity between the corpus and the language model only to compare models that share the same vocabulary. Among these comparable models, we know that the one with a lower cross-perplexity is better at predicting the data.

Having briefly discussed cross-perplexity as a metric that measures how well a model fits some data, and its dependence on the complexity of the data, we will now turn to the actual computation of the cross-perplexity between a PCFG model and a corpus. A PCFG assigns zero or more parses to a given sentence, each with a probability equal to the product of the probabilities of all rules used within the parse. The sum of the probabilities of all parses of the sentence is the total probability that the PCFG assigns to the sentence, because the parses represent all ways in which the PCFG can generate the sentence. Syntactic annotation in the treebanks considers sentences as separate units, which is also the assumption under PCFGs. The

probability of the whole corpus is consequently the product of all sentence probabilities.

The probability of a sentence can be calculated via the inside algorithm almost exactly as the most likely parse (see section 2.4). The only difference is that instead of storing spans of most likely constituents per chart entry (determined in equation 2.4), each entry  $[A, i, k]$  just holds the sum of probabilities of all subderivations of constituent  $A$  spanning  $w_{i,k}$ :

$$[A, i, k] = \sum_{j=i+1}^{k-1} \sum_{A \rightarrow BC \in \mathcal{R}} P(A \rightarrow BC) P(B \xrightarrow{*} w_{i,j}) P(C \xrightarrow{*} w_{j,k})$$

so that when the chart is filled, the entry for the start symbol spanning the whole input string holds the probability of the input string according to the PCFG:

$$[S, 0, |w|] = P(S \xrightarrow{*} w_{0,|w|})$$

Sentences that fail to parse are considered in our evaluation, in that the sum of their lengths is subtracted from the number of words used to compute the perplexity.

Inside probabilities cannot be easily calculated for cyclic grammars, because they give part of the probability mass to parses of infinite depth.<sup>4</sup> There exist algorithms to calculate it, but they are rather expensive (Klein and Manning, 2001a). Cyclic grammars occur only rarely in our experiments, because symbols generating cycles also cause massive ambiguity and are usually among the first to be removed by our transformations. The parser we use has been extended by us to deliver only a simple approximation of sentence probability based on parses with the minimal number of applications of rules forming cycles. Calculating probabilities along these lines ignores possible derivations (those with repeated rule applications) and thus determines sentence probabilities that are too low. We nonetheless stick to the simple sentence probability estimates also for those grammars that are cyclic and accept that resulting probability estimates may be too low. This results in perplexity measures that are too high (see equation 3.1 below).

We will now briefly give the remaining steps to derive the cross-perplexity from sentence probabilities. We assume that sentences  $w$  in our corpus  $\mathcal{W}$  are probabilistically independent:

$$\begin{aligned} P(\mathcal{W}) &= P(w_1, w_2, \dots, w_{|\mathcal{W}|}) \\ &= \prod_{w \in \mathcal{W}} P(w) \end{aligned}$$

---

<sup>4</sup>In a cyclic grammar, you can start from a node and arrive at the same node again by traversing edges in a single direction.

where each sentence probability equals the inside probability of the node with the grammar’s start symbol covering the whole sentence in our PCFG model:

$$P(w) = P(S \xRightarrow{*} w_{0,|w|})$$

We are interested in the average per-word perplexity of a corpus, which for us are the sentences in the treebank  $\mathcal{W}$ , given a model, which is our grammar  $G$ . The average per-word perplexity of a corpus is then defined as:

$$\text{perp}_{\text{cross}}(G; \mathcal{W}) = 2^{H_{\text{cross}}(\mathcal{W}; G)}$$

where  $H_{\text{cross}}(\mathcal{W}; G)$  is the cross-entropy between model and corpus, which equals the negative log-likelihood of the corpus  $L(\mathcal{W}; G) = \frac{1}{|\mathcal{W}|} \log_2 P(\mathcal{W})$ :

$$\begin{aligned} \text{perp}_{\text{cross}}(G; \mathcal{W}) &= 2^{-L(\mathcal{W}; G)} \\ &= 2^{\frac{1}{|\mathcal{W}|} \log_2 P(\mathcal{W})} \\ &= \frac{1}{\sqrt[|\mathcal{W}|]{P(\mathcal{W})}} \end{aligned} \tag{3.1}$$

where our grammar gives us  $P(\cdot)$ , and  $|\mathcal{W}|$  is the length of the corpus in words (Prescher, 2002, p. 51ff.). We will refer to the cross-perplexity  $\text{perp}_{\text{cross}}(G; \mathcal{W})$  as *perp* from now on.

Information-based metrics, relational evaluation and the Parseval metrics are widely applied, but there are a number of other metrics that we do not consider. Carroll & al. (1998), e.g., propose to map parses directly to predicate-argument structure, which is expected to represent the decisions better that are relevant for the semantic interpretation of a parse. A drawback is that it is not immediately obvious how to extract the predicate-argument structure from gold data. The leaf-ancestor metric is another distinct evaluation metric that is equally applicable to constituents and dependency-based analyses and thus does not require mappings between both types of analyses (Sampson and Babarczy, 2003). It reacts less nervously to attachment errors than Parseval and is not equally distracted by increasing the proportion of trivial annotation at the low end of syntactic trees, because annotation higher in the tree has more impact on it (Sampson, 2000, p. 66). The main reason why we do not employ the leaf-ancestor metric is that the other metrics have been applied more widely, and thus allow for better comparison, although, admittedly, numbers on different data sets can hardly be compared; just the behaviour of the metrics is better known. We nonetheless acknowledge that it would be useful to include the leaf-ancestor metric into the set of standardly applied metrics in the future.

Part II

Representations of Syntactic  
Analyses





# Chapter 4

## A Treebank's Choice

We will frequently experiment with grammar models that are derived from syntactically annotated language data. Collections of such data are called treebanks, because the annotation found in them usually resembles some kind of tree. These trees express linguistic analyses that correspond to regularities observed in the data, i.e. they express generalisations that we refer to as syntactic phenomena. Treebanks exist for a variety of languages, including German (Abeillé, 2003). For written German, there are two such resources. First, the *negra* treebank, which is enlarged and continued as the *Tiger* treebank (Brants & al., 2002; Skut & al., 1998). Second, the *Tübinger Baumbank des Deutschen/Schriftsprache* (TüBa-D/Z; Telljohann & al., 2004). *Tiger* and TüBa-D/Z are similar in that they both comprise data from major newspapers of the 1990s. Both follow the same scheme to assign parts of speech to words (the Stuttgart-Tübingen Tagset: STTS; Schiller & al., 1995), and their main goal is to provide syntactic analyses for individual sentences. They rely on the same tool for annotation, which uses graphs to express the analyses, so that their analyses also share some formal properties. While the scope of the annotated phenomena is similar and includes parts of speech, constituents, and predicate-argument structure, the individual solutions to encode the analyses are different.

Having two similar treebanks at hand, it is tempting to examine their differences, and to judge how these differences affect treebank users. Even within a single treebank, though, similar generalisations may be represented in different ways. For the major part of this chapter, we will restrict ourselves to examining alternative representations within a single treebank, because our main interest is not so much to discuss the differences between treebanks, but rather the consequences of certain ways to represent syntactic analyses when there is no obvious representation, or when there are several equally plausible representations of a syntactic phenomenon.

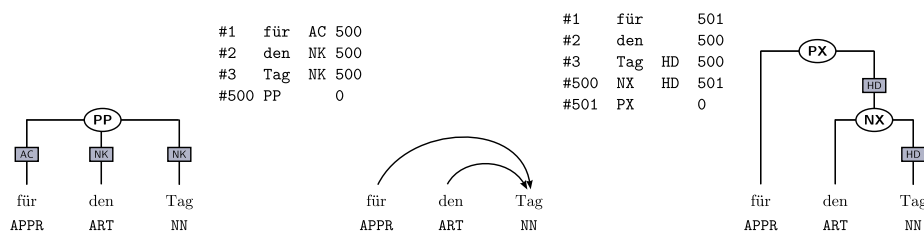


Figure 4.1: Representations of Relations within a Prepositional Phrase

Figure 4.1 shows representational variants of analyses of a prepositional phrase as an example.<sup>1</sup> TüBa-D/Z and Tiger go different ways to annotate them (see extreme right and left). The graphs correspond to an underlying representation, which is shown as tables of labelled nodes and edges resembling the export format, which is discussed in more detail below. It is sufficient for us here that terminal and nonterminal nodes correspond to lines, which are numbered, and which also encode the node label and a (possibly labelled) edge. The representations as elements of a graph and as entries in a table are *equivalent* in the sense that one can be derived from the other by simple mappings between a line in the table and a node plus its upwards edge in the graph. Another representation of syntactic relations is the dependency representation, which is shown in the middle of the figure. This representation does not provide information about intermediate nodes and their labels, and is thus not equivalent to the other representations, while it still can serve to represent syntactic relations that hold between words that make up prepositional phrases. Different representations thus do exist for a single phenomenon in a language, but also within a single treebank, when we accept that the dependency representation and the constituent representation, and also the table representation, express the same relations. The terms that we will use to discuss these variants are *phenomenon*, *representation*, and *encoding*. A *phenomenon* is a generalisation that corresponds to patterns observed in the language data recorded in a treebank, such as a prepositional phrase. This generalisation is expressed in some *representation*, which can highlight certain aspects like constituency or dependency. The *encoding* is the specific representation in which the treebank is originally delivered (i.e. the tables in figure 4.1 for both treebanks). We assign this specific representation a special status, because its choice is not arbitrary, and its shape most likely has consequences for the users of a treebank.

<sup>1</sup>See appendix A.2 and appendix A.3 for a description of all labels in negra and TüBa-D/Z. We do not treat the newer Tiger annotation scheme differently, because for our purposes both schemes behave identically (Brants and Hansen, 2002).

We focus on TüBa-D/Z as an example and discuss the choice of representation for selected phenomena in this chapter. Before we turn exclusively to TüBa-D/Z, however, we will briefly survey some differences between TüBa-D/Z and Tiger. TüBa-D/Z strives to build annotation around a context-free backbone, while Tiger employs crossing edges from the start to cope with the relatively free constituent order in German. In Tiger, the clausal node is the mother of the finite verb and of the verb phrase that groups the remaining non-finite verbal parts and all their complements and adjuncts. As the elements of the verb phrase may move around rather freely within a German sentence, this leads to crossing edges whenever they are realised both before and after the subject or the finite verb (see figure 4.2(a)). TüBa-D/Z does not

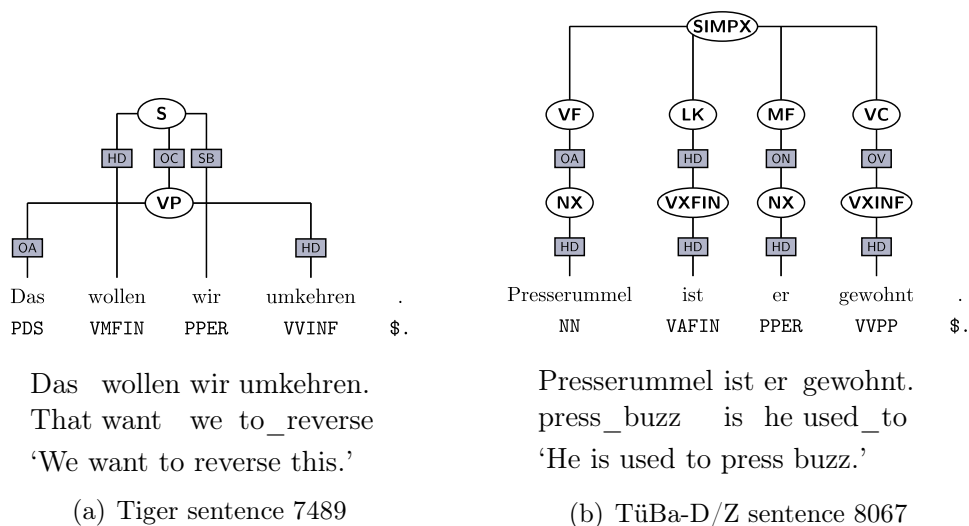


Figure 4.2: Crossing Edges in Tiger vs. Co-Indexation in TüBa-D/Z

allow crossing edges, but groups constituents into so-called topological fields, which always occur in linear order. The context-free backbone in TüBa-D/Z encodes the fields rather than the relations between the constituents. Relations between the verbal parts and their complements and adjuncts are also recorded, but via edge labels expressing the grammatical functions instead of via potentially crossing edges. These edge labels are co-indexed in the sense that they form groups, such as the complements including accusative object (OA), dative object (OD), genitive object (OG), and the subject (ON), which all depend on the verbal parts of the clause (see figure 4.2(b)). The verbal parts of a clause are always found in the same clause, but not necessarily adjacent to the complements. A similar group of edge labels covers modifiers of these complements (e.g. the modifiers of the accusative object OA-MOD, of the dative object OD-MOD, and of the genitive object OG-MOD). Each modi-

fier of a constituent with a certain grammatical function is thus co-indexed with the constituent via the naming scheme of the edge labels marking the respective constituents. Co-indexation is rather powerful, because it is just a different means to express relations between arbitrary nodes in the graph that carry labels from a predefined subset of all labels.

Formally, the difference between Tiger and TüBa-D/Z still holds, of course, that the former employs crossing edges while the latter does not. It is not the case, though, that Tiger prefers graphs to express syntactic structure on all levels of the analyses. Tiger encodes syntactic categories in node labels, and grammatical functions in edge labels. The tree (or rather graph, as it includes crossing edges) defines argument structure. Internal structure of constituents is consequently mainly expressed via edge labels, and not in the graph, so that the resulting annotation graphs are rather flat. TüBa-D/Z adds topological fields as a separate descriptive level, and it does not allow edges to cross (except for a separate class of so-called *secondary edges*, which are also present in Tiger). TüBa-D/Z uses the graph to encode structural information inside constituents more extensively than Tiger, resulting in analyses that consist of more nodes per word on average.

The primary goal of a treebank is to annotate syntactic phenomena in language data. It is not clear whether the choice of crossing edges, or co-indexation, offers the best support for this goal. We will not try to answer this question here. We will instead concentrate on a single treebank and try to show why its designers choose a certain kind of graph to express the syntactic analyses. Comparing different representations (e.g. crossing edges vs. co-indexation) of a specific phenomenon (free constituent order) between treebanks as outlined above aims at showing which encoding is more appropriate for a certain phenomenon. Instead, we propose to concentrate on the annotation in TüBa-D/Z, which most likely tries to capture the same syntactic phenomena as the annotation in Tiger. The initial goal of a treebank is to identify these phenomena (such as constituents, or the relations between verbs and objects), analyse them, and store descriptions of the analyses. The ultimate goal of treebank designers is that a user will understand this analysis and identify the corresponding phenomenon in the same way as originally intended by the treebank annotators. The analyses are stored by means of a graph for both Tiger and TüBa-D/Z, i.e. both use a data model for storage that is strictly defined in terms of labelled nodes and edges. These graphs usually take the form of a tree, which seems to be a natural choice to represent hierarchical structure such as the structure of constituents or clauses. Other phenomena, such as ambiguous analyses, are not similarly easy to represent as trees, though.

There seem to be obvious causal connections between the phenomena for

---

which analyses are recorded in a treebank, their encoding as graphs, and their tree-like graphical representation. Our main interest in this chapter is to highlight the constraints that interact and determine the individual choice of encoding and representation of linguistic phenomena. Their choice determines whether a treebank user understands the annotation in the same way as the treebank annotators. We will consequently consider phenomena, encoding, and representation independently. We will try to show that different constraints determine their choice, and that they can be partly optimised independently. The influence of the variation of representations for a different application will be our topic in the next chapter, where we will focus on how different representations of syntactic analyses affect the way that PCFG parsers capture probabilistic dependencies between the parts of an analysis. Before we start, we give some more details on TüBa-D/Z and the *export* format.

## A Treebank of German Newspaper Texts: TüBa-D/Z

TüBa-D/Z consists of 15 260 sentences.<sup>2</sup> Texts in TüBa-D/Z come from the daily newspaper *die tageszeitung* (taz), covering May 3–7, 1999. Texts have been automatically segmented into words and sentences with some manual post-editing. Annotation continues semi-automatically with the support of the *annotate* tool (Plaehn, 1998), which aids in annotating and editing parts of speech and syntactic analyses.

Part of speech (POS) annotation follows the distinctions of classes given in Schiller & al. (1995). Each word is annotated with a single POS label. Words are grouped into constituents, and sequences of constituents are grouped into fields, and sequences of fields are grouped into clauses. Constituents may contain other constituents, and clauses may be part of fields and constituents, but constituents are never direct children of clauses, so that annotation always consists of layers of constituents, fields, and clauses that are stacked in a strict order.

The main principles of syntactic annotation are given in the manual of the treebank (Telljohann & al., 2003), along with detailed guidelines for the annotation of the individual types of phrases, fields and clauses. All information about the annotation of TüBa-D/Z that we refer to here is drawn from this manual or the data itself. Annotation in TüBa-D/Z follows three main principles: the *flat clustering* principle, the *longest match* principle, and the *high attachment* principle. Both the flat clustering and longest match principles express a preference to encode syntactic analyses with fewer nodes

---

<sup>2</sup>We always refer to the first release of TüBa-D/Z published in December 2003.

when there is a choice to either introduce additional nodes or to add more branches below existing nodes. Annotation only connects fragments of sentences when there is some kind of dependency relation between them. The high attachment principle states that when the scope of a modifier cannot be determined, then it is attached at the lowest position in the tree that includes all possible interpretations. It is called high attachment principle because when it is unclear whether a modifier has a less ambiguous reading that is encoded by lower attachment or a more ambiguous reading encoded by a higher attachment, then the higher attachment site is chosen.

A declared general goal of TüBa-D/Z is easy reuse of the treebank. The annotation should not be influenced by a commitment to a specific linguistic theory. Instead, the annotation scheme should be based on uncontroversial assumptions about syntactic structure, yet reflect specific properties of the annotated language, so that as many potential users as possible can take advantage of the treebank (Telljohann & al., 2003, p. 8). The treebank manual has the important function to explain the choice of encoding, because there are situations where the actual analysis of a phenomenon cannot be easily determined from the general principles outlined above, when there is no unique obvious way to express an analysis in a tree structure with the given inventory of labels for labelled nodes and edges.

### The *export* data model

There are some formal restrictions on the design of the annotation which come from the chosen data model. You are in principle free to choose any data structure to encode the annotation of your choice. Rather simple graphs are sufficient to encode a wide variety of (if not all) existing kinds of syntactic annotation (Bird and Liberman, 2001). In practice, however, you will reuse an existing tool to automate annotation, because such a tool speeds up annotation, enforces consistency, but is costly to create. The annotation effort of TüBa-D/Z reuses the tool *annotate*, which has originally been produced to support the annotation of *negra*, and also aids annotation of *Tiger* (highlighting another feature common to these treebanks).

The data model of *annotate* thus gives a hard constraint on the encoding of syntactic annotation in TüBa-D/Z. The data model is rather flexible, though, as it is capable to express more than proper trees (henceforth the *export* model; Brants, 1997). It describes sets of unconnected sentences. Each word has exactly one parent node, which is either a normal non-terminal node, or the special non-terminal called the *virtual root node* (VROOT) marking the root of each sentence. Being connected to VROOT means being unattached to any syntactic annotation, because VROOT and all edges lead-

---

ing to VROOT are not shown in the standard representation of the *export* data model. Non-terminal nodes also have exactly one parent node, which is again either a normal non-terminal node or VROOT. Only the descendants of a non-terminal are excluded from the set of its allowed parents in the *export* model, so that edges may cross, but cycles do not occur. There is a separate type of dependencies called *secondary edges*, though, which can be used to represent also cycles. Each non-terminal node, and each word, may have an unlimited number of secondary edges, which point to an arbitrary node in the graph that belongs to the same sentence. Each word, non-terminal node, edge, and secondary edge also has a finite set of attributes with values that are selected from a finite set of different labels, so that formally, each of these elements can be seen to have an atomic label from the set of node labels times the set of attribute labels (see Brants, 1997, for the inventory of attributes).

Secondary edges, and node labels that are connected by their definition in the treebank manual (i.e. via co-indexation), essentially express relations between arbitrary nodes in the annotation graph. The difference between them is formal rather than semantic. Secondary edges connect two nodes explicitly, and they have a label from a finite set of labels. Co-indexation also relies on a finite set of labels, but has no other means beside the label to point to another node. Co-indexation thus has to enumerate every single relation in the set of labels, because if the label were re-used, it would become ambiguous: given that a relation of type  $a$  is always expressed via the two labels  $(a_1, a_2)$  that mark two nodes in a sentence, then no other two nodes in this sentence could be co-indexed by the same labels, because it would not be clear anymore which  $a_1$  belongs to which  $a_2$ . Co-indexation may be more appropriate than secondary edges in situations where an encoding of source and destination of an edge would be redundant, e.g. when the definition of co-indexed labels makes it clear that only daughters of the same mother may be related via the labels.

A graph consists of nodes and edges between the nodes. A rooted tree is a special kind of graph with a single root node and directed edges that point away from the root node and connect each mother node with its daughter nodes. Each node except for the root node in a rooted tree has exactly one mother node. The root node does not have a mother node. The edges of a tree do not form cycles. Words are the terminal nodes in the *export* model. When the daughters of each non-terminal node are ordered so that they occur in the same order as the words that they dominate directly or indirectly, then we call the tree a *proper tree*. Proper trees do not have any crossing edges. Node labels serve to distinguish different types of nodes.

The *export* data model that is used to encode syntactic analyses in TüBa-D/Z thus has many features in common with proper trees. TüBa-D/Z further

restricts expressiveness of the *export* data model in that daughter nodes are always ordered in the same sequence as the terminal nodes that they dominate, i.e. edges do not cross. The definition of the *export* format does not rule out non-terminals that do not dominate any word, but this feature is not used in either TüBa-D/Z or Tiger.

## 4.1 Observations: Challenges for Proper Trees

Proper trees are a natural choice to represent constituency, and labels distinguish categories, so that we seem to have the proper tools at hand now to encode syntactic analyses. There are some aspects of annotation in TüBa-D/Z, however, where it is not clear how to employ proper trees to represent a syntactic analysis. This section presents some of the more problematic cases in TüBa-D/Z, where we focus on the syntactic phenomena, and we compare how they have been identified and annotated in the treebank, and how the annotation should be understood by the users of the treebank.

### 4.1.1 The Objects of Syntactic Annotation

A graph representing a syntactic analysis connects at most as many words as there are in a sentence. Breaking up the input text into sentences, and identifying words, thus determines important features of the individual graphs, and the principles of tokenisation and sentence segmentation determine which elements in a sentence are related syntactically (i.e. words, and not morphemes), and which elements cannot be related (i.e. the words of different sentences). Information about morphemes may be accessible when atomic POS tags encode them, but this depends on the choice of POS tags, and different morphemes in a word still would have to share a single parent.

#### Segmentation

Word and sentence segmentation in TüBa-D/Z is originally based on the automatic segmentation procedures developed for the DEREKO corpus (Ule, 2002). Automatic segmentation has been edited manually when there were obvious printing errors. Yet the spelling itself was generally not altered (see also Telljohann & al., 2003, section 3.4.4). Tokenisation and sentence segmentation are revised manually and they therefore only pose problems where largest units connected by syntactic dependencies exceed the size of sentences, or smallest units collide with the requirements of POS tags.

Sentential arguments can only be attached properly when all words making up the main and the subordinate clauses are segmented as a single sen-



tence. Newspaper texts as found in TüBa-D/Z contain many quotations in direct speech, which often consist of several clauses. Examples range from single quoted words to several fully quoted sentences in a row. Quoted material is sometimes considered as an adjunct or argument of the matrix clause (usually for smaller fragments), but sometimes not (usually for larger segments that make up full sentences). In TüBa-D/Z, at most a single sentence in direct speech that is introduced or finished by indirect speech is segmented as one sentence. When direct speech consists of more than one sentence, then only indirect speech preceding the first sentence and connected with a comma (when introduced by indirect speech), or following the last sentence (when finished by indirect speech) is segmented into the same sentence as the direct speech. Relations between direct and indirect speech therefore cannot always be expressed when sentences are the largest descriptive units.

### Parts of Speech

Part of speech tags are atomic labels of the terminal nodes of the annotation graph. They label the individual words that have been identified during segmentation. Some pairs of function words in German frequently co-occur and are blended. High-frequent articles and prepositions form a single word which is found more frequently in written language than the combination of two words, including *zum* ('to the'), *zur* ('to the'), *im* ('in the') (Eisenberg & al., 2005, p. 622). The POS tag set provides tags for these merged forms, so that the function of a preposition and an article is either expressed by one or by two words. As a result, syntactic analyses of prepositional phrases contain noun phrases where the article is either realised within the noun phrase or is part of the dominating prepositional phrase, as opposed to Tiger, where this distinction is less explicit (see figure 4.1). Figure 4.3(a) shows two coordinated prepositional phrases (PX) where the first noun phrase (NX) contains the article (ART), while the second article is merged with the preposition (APPRART) and is part of the PX instead.<sup>3</sup> Other words can also be merged, but do not receive dedicated POS tags, as the combinations are much less frequent. The subject in figure 4.3(b) is not visible as part of the word *isser*/VAFIN (blending *ist*/VAFIN and *er*/PPER), because the verb was considered more important, and either a POS tag for a verb or for a pronoun could be assigned to the word *isser*.<sup>4</sup> The secondary edge *refint* that marks the dependency to only part of a constituent shows that *unser Lou* refers to part of *isser*/VAFIN, but it is not obvious that it refers to the personal pronoun *er* ('he').

The inventory of a POS tagset could provide a large number of labels

---

<sup>3</sup>Part of TüBa-D/Z sentence 2368.

<sup>4</sup>Part of TüBa-D/Z sentence 6974.

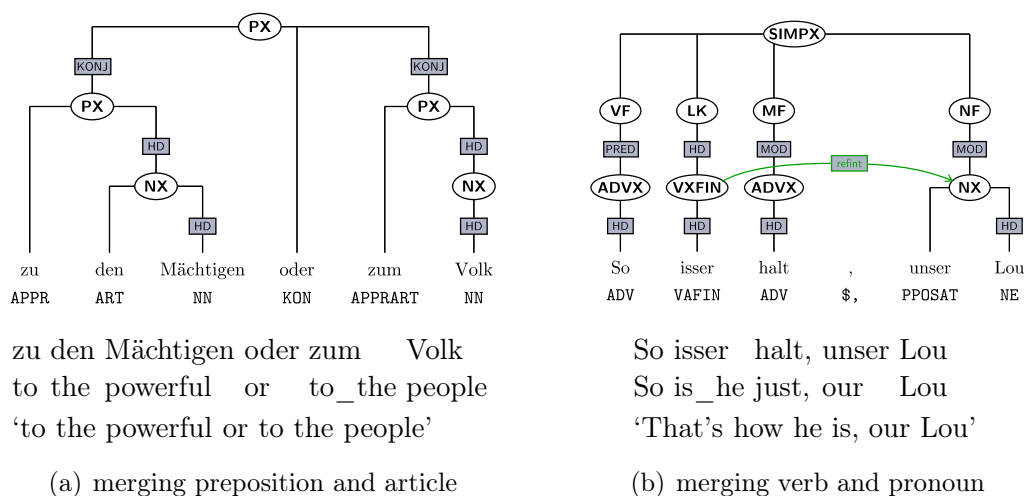


Figure 4.3: Segmentation Determines Syntactic Analyses

with separate labels for all combinations, as for `APPRART` in figure 4.3(a). Alternatively, segmentation could split *isser* in figure 4.3(b) into two separate words, which could be tagged individually and which could be referred to subsequently as separate words in syntactic annotation. There is no dedicated mechanism in the *export* model that shows whether two words are separated by whitespace or not, and TüBa-D/Z also has no means to encode this distinction. As a result, the shape of noun phrases embedded in prepositional phrases may vary depending on where the article is realised, as in figure 4.3(a), and information about the elements of the syntactic analysis may be hard to recover, as in figure 4.3(b).

## 4.1.2 Unattached Elements

In a proper tree, each word and each node is attached to a mother node, and there is one special root node. TüBa-D/Z, however, also includes words that are not attached to any node. Recall that *not attached* is an interpretation of the encoding of an edge pointing to `VROOT`, which is not shown by default. Unattached nodes fall into two classes: they are either terminal nodes representing punctuation, or non-terminal nodes representing parentheses.<sup>5</sup>

Unattached elements cannot be encoded by proper trees, because they do not form a connected graph. As being unattached only refers to the representation (i.e. invisible), and not to the encoding (i.e. an edge to `VROOT`), the problem with unattached elements is rather that it is not guaranteed

<sup>5</sup>Note that punctuation with “semantic meaning” (Telljohann & al., 2003, p. 22) is attached.

that there are no crossing edges, and not that there is an unconnected graph. While unattached elements in TüBa-D/Z may turn the annotation into a graph that is not a proper tree, the unattached elements are properly nested, i.e. there is always a single parent that dominates the words preceding and following the parenthesis, or VROOT if there is no sister at one side. When a new edge is added between this mother and the unattached element, then this edge and the edges of the unattached element never need to cross the edges of the remaining graph.

Parentheses in TüBa-D/Z thus can be modelled by proper trees by connecting them to the rest of the annotation graph by an edge. When this edge is assigned a special status via a dedicated label, then the new representation as a proper tree can be mapped unambiguously to the original encoding, so that both can be considered to be equivalent.

### 4.1.3 Free Constituent Order

German is a language with a rather free constituent order. Any optional or obligatory arguments of a verb may choose their position relative to the verbal parts of a sentence with great freedom (Eisenberg & al., 2005, p. 881ff.). The topological field model for German takes care of this freedom, as the linear series of *topological fields* in a clause always groups consecutive words. The topological fields are named according to their position relative to the *verbal bracket*, which consists of two fields that contain the verbal parts of a sentence: the left bracket (*linke Klammer*, LK) and the verbal complex (VC). The initial field (*Vorfeld*, VF) precedes the LK, the final field (*Nachfeld*, NF) follows the VC, and the middle field (*Mittelfeld*, MF) is enclosed by the verbal bracket. We show a slightly simplified general schema of topological field sequences that occur in German in table 4.1, where bold field names indicate obligatory fields, and all other fields may also be empty. The

clause type	topological fields						
VL	KOORD	LV	<b>C</b>	MF	<b>VC</b>	NF	
V1	KOORD	LV	<b>LK</b>	MF	VC	NF	
V2	KOORD/PARORD	LV	<b>VF</b>	<b>LK</b>	MF	VC	NF

Table 4.1: Topological Field Sequences in German Clause Types

KOORD/PARORD fields are occupied by coordinating or non-coordinating particles, and the LV field may contain resumptive constructions (*Linksversetzung*). The scheme follows the naming scheme of TüBa-D/Z (which in turn adopts Höhle, 1985). The three clause types are verb-last (VL), including all

introduced subclauses, verb-initial (V1), including yes/no-questions and finite imperative clauses, and verb-second (V2) clauses, including affirmative clauses. We give an example for each clause type in figure 4.4.<sup>6</sup> It is easy to

	..., [C die ] [MF groß ] [VC sind ] [NF wie Tennisplätze ]
VL:	..., that large are as tennis_courts
	‘..., that are as large as tennis courts’
	[LK Veruntreute ] [MF die AWO Spendengeld ] ?
V1:	Did_misappropriate the AWO donations ?
	‘Did the AWO misappropriate donations?’
	[VF Die Zeit der Dürre ] [LK ist ] [MF vorbei ]
V2:	The time of drought is over
	‘The time of drought is over’

Figure 4.4: Examples for German Clause Types

see that the names *verb-last*, *-initial*, and *-second* refer to the position of the finite verb within the obligatory fields of a sentence. In a VL clause, the VC is the last obligatory field, in which we consequently find the finite verb *sind*, and we similarly determine the position of the finite verb in the other two sentences.

The topological field model is descriptive rather than explanatory, i.e. it does not explain clausal structure as part of a restrictive universal theory of syntax (Höhle, 1985, p. 339). It nonetheless is a theory, even though only descriptive, in that it describes the sequence of fields in German sentences reliably and in that it is compatible with many other accounts of the ordering of verbal and non-verbal parts of German clauses.<sup>7</sup>

Constituents may choose a position rather freely in the initial field, in the middle field, or in the final field. The topological field model does not have strict implications for the position of constituents in a clause in general, except for the constraint that the VF may only contain a single constituent, so that e.g. *der Dürre* in the V2 clause in figure 4.4 is not ambiguous between a complement to the verb and a modifier to *Die Zeit*, but can only modify

<sup>6</sup>Showing parts of TüBa-D/Z sentence 2051, sentence 1, and sentence 1486. Please see appendix A.3 for a description of all field labels.

<sup>7</sup>Höhle (1985) explicitly compares the current theory with earlier accounts in Herling (1821), Erdmann (1886), Drach (1937), and Engel (1970).

*Die Zeit*. Except for this restriction, the characterisations of the order in which constituents occur in the fields of a clause describe preferences rather than hard constraints. Constituent order depends on grammatical features of the constituents like case or definiteness, but also on semantic features like animacy or information content (Eisenberg & al., 2005, pp. 882, 889, 1867).

In TüBa-D/Z, fields are encoded as labelled nodes in exactly the same way as constituents and clauses. In figure 4.5(a) we see the original annotation of a sentence in TüBa-D/Z.<sup>8</sup> When we express the relations between the constituents within the fields via dependencies, we see that the fields simply group these constituents, and that the constituents always depend on the finite part of the verb in the verbal bracket, no matter in which field they occur (figure 4.5(b))<sup>9</sup>. We see that constituents correspond to all words dominated directly or indirectly by the head of each constituent (i.e. all parts of  $[_{NX}$  Eine solche Veranstaltung ] depend on *Veranstaltung*/NN), and that similarly clauses correspond to all nodes dominated directly or indirectly by the inflected part of the clause’s verbal bracket (here: *werden*/VAFIN).

Fields, however, do not appear similarly in the dependency graph. They mark the areas in which the constituents can appear, but they do not directly specify syntactic relations. The representation of fields as the same type of nodes as constituents and clauses can therefore be misleading, because it suggests that both classes of nodes behave identically. There is no need to adhere strictly to the original representation of TüBa-D/Z to represent the encoded information. A conversion to a different representation normally implies that parts of the information are lost (e.g. field information in figure 4.5(b)), and that the default reading of an analysis is altered, e.g. in figure 4.5(b) the relations between the inflected verb and the objects are much more pronounced than in figure 4.5(a). Such a conversion thus leads to a different interpretation of the data. We would like to stress that also before the conversion, a similar kind of interpretation takes place, e.g. the fields in figure 4.5(a) are more prominent than in figure 4.5(b).

We will use a different representation of nodes representing fields in the current chapter, which is shown in figure 4.5(c). The verbal parts are placed against a grey background and provide a frame to the constituents, which is confined by the clause and breaks it up into fields. We think that a representation that stresses the special status of the subclasses of node labels representing fields may help transport the observations underlying the original annotation of TüBa-D/Z. Grammatical functions of maximal constituents

---

<sup>8</sup>TüBa-D/Z sentence 5230.

<sup>9</sup>Thanks to Matthias Trautner Kromann for his DGgraph drawing tool that we have used to render dependency analyses.



are contrasted with functions of the verbal parts inside the verbal brackets by moving their function labels up against the bracket. The advantage of this representation, which highlights the status of topological fields, becomes more apparent when secondary edges cross field boundaries, or in complex sentences or coordinations (see below).

The types of relations between the maximal constituents in the fields and the verbal parts of the sentence are encoded in the edge labels that link the maximal constituents to the fields. Most dependency relations hold between the maximal constituents and the parts of the verb, so that it is sufficient to indicate the type of the relation and the dependent, because the head is given via the annotation of the verbal bracket. There are also dependencies that link a maximal constituent to another maximal constituent, or to an element that is neither a maximal constituent nor part of the verb bracket. Dependencies between maximal constituents are not encoded via a dominance relation but via a naming scheme of the edge labels. Constituents with an edge labelled  $X$  are the head of a dependency connecting them to the constituent labelled  $X$ -MOD. Given a label set that contains an  $X$ -MOD for each  $X$ , a single dependent can be specified for each maximal constituent with function  $X$ . This naming scheme is sufficient to express most relations between constituents and verbs in one clause. There are still situations, though, where other relations have to be encoded. In these situations, the naming scheme is augmented by secondary edges.

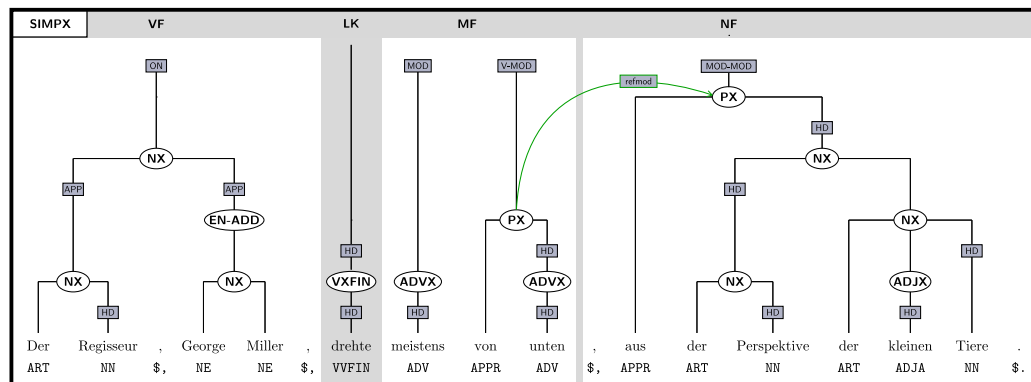
When there is more than one constituent with function  $X$  in the same clause, then  $X$ -MOD becomes ambiguous, and a *refmod* secondary edge resolves this ambiguity. Figure 4.6 shows such a sentence where the prepositional phrase in the final field (NF) could modify either the adverbial or the prepositional phrase in the middle field, because they both carry modifier edge labels (MOD and V-MOD), and the edge label MOD-MOD of the prepositional phrase in the final field generally refers to any modifier.<sup>10</sup> The secondary edge resolves this ambiguity, pointing from the head to the dependent MOD-MOD.

There are also relations that do not hold between maximal constituents below fields, as was still the case for *refmod*. These relations link either phrase-internal or clause-external nodes to a constituent directly dominated by a field node. As the set of candidate nodes for these relations is much larger than for two daughters of field nodes, they are expressed exclusively via secondary edges. Relative clauses often depend on phrase-internal nodes and are thus often connected to these nodes by a secondary edge labelled *refint* for dependencies on phrase-internal nodes (see figure 4.7(a)<sup>11</sup>). Complements

---

<sup>10</sup>TüBa-D/Z sentence 3720, without trailing double quotes.

<sup>11</sup>TüBa-D/Z sentence 2791.



Der Regisseur, George Miller, drehte meistens von unten, aus der  
 The director, George Miller, filmed mostly from below, from the  
 Perspektive der kleinen Tiere.  
 perspective of\_the small animals.

‘The director, George Miller, filmed from below most of the time, from  
 the perspective of the small animals.’

Figure 4.6: Secondary Edge Disambiguates Co-indexed Modifiers

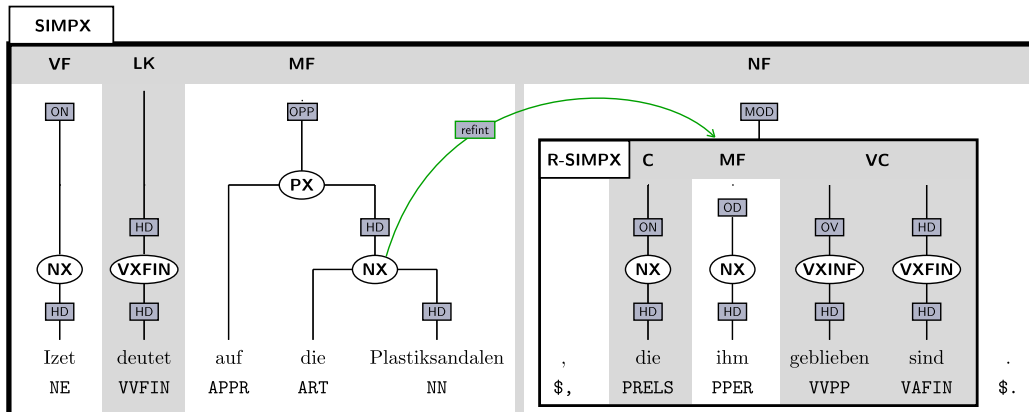
of verbs in the sentential objects of control verbs are sometimes outside the sentential object. The edge label *refcontr* expresses this type of dependencies (figure 4.7(b)<sup>12</sup>).

Secondary edges are also used inside the verbal complex to complement the edge label set that distinguishes between the head of the verbal complex (HD) and the verbal objects (OV) that are sister nodes. For up to two elements, these two edge labels specify exactly the dependency relation between the parts of the verbal complex. When there are three or more elements, all elements but the head are related via secondary edges with a *refvc* label pointing from selecting to depending verbal object. There is no example in TüBa-D/Z where this order deviates from the standard right-to-left order, so that a general rule of the ordering of verbal objects in the verbal complex would be sufficient to describe existing data.

The mechanisms that are used to encode relations between the elements of a German sentence in TüBa-D/Z instead of crossing edges are thus mainly co-indexation of edge labels and topological groupings into fields. Where this is not sufficient, (crossing) secondary edges are used.

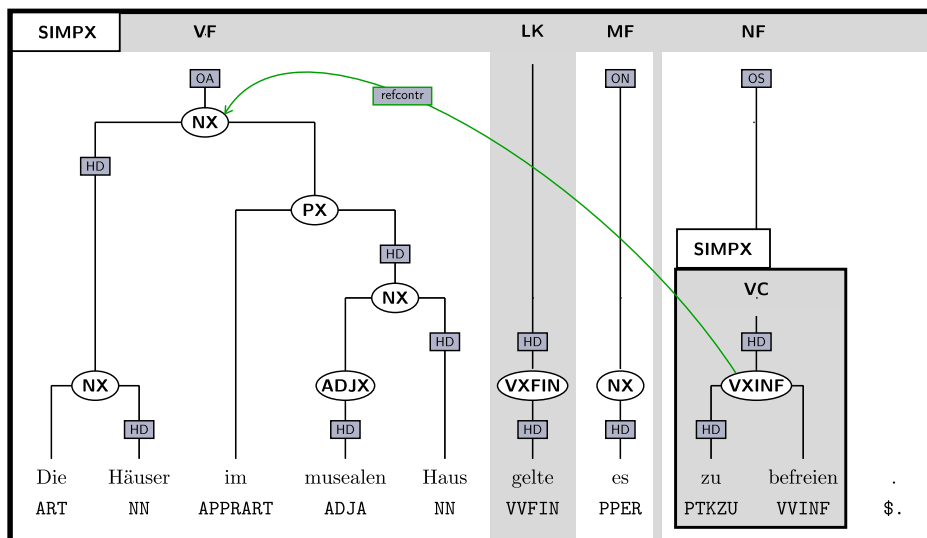
<sup>12</sup>TüBa-D/Z sentence 1895.





Izet deutet auf die Plastiksandalen, die ihm geblieben sind.  
 Izet points at the plastic\_sandals, which him left were.  
 ‘Izet points to the plastic sandals, which were left to him.’

(a) phrase-internal modification: refint



Die Häuser im musealen Haus gelte es zu befreien.  
 The houses in\_the museum house imperative\_is it to free.  
 ‘It is necessary to free the houses inside the house of the museum.’

(b) clause-external modification: refcontr

Figure 4.7: Clause-external and Phrase-internal Modification

### 4.1.4 Coordination

A coordination in a syntactic analysis groups elements with a similar status into a single new element. Coordinations can group all three major types of nodes in TüBa-D/Z, i.e. clauses, fields and constituents (or words). Coordination generally assumes close similarity between the coordinated elements. Only nodes of the same major type can be coordinated in TüBa-D/Z accordingly, so that a single new element of the same major type emerges, where such a *major type* is equivalent to a predefined subset of all node labels. When two or more clauses are coordinated, the two clausal conjuncts are thus dominated by a new node that also has a label of a clausal type, and equally fields form a new field node, and constituents a new constituent node. We will refer to the coordinated elements as *conjuncts*, to the whole new construction as *coordination*, and to the words connecting conjuncts as *conjunctions*.

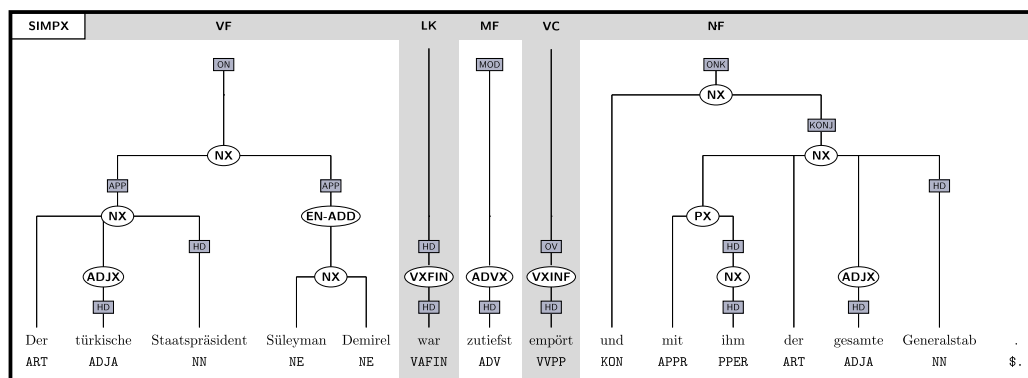
#### Linear Order

The encoding of a coordination as immediately dominating nodes of the same major type can be easily expressed in a proper tree as long as the nodes immediately follow each other, they all have the same node label, and are all individually coordinated. When one of these requirements is not met, it is harder to encode the relation among the nodes, and of the whole coordination to the rest of the clause, in a proper tree.

The elements of coordinations do not always precede each other linearly. Conjunctions often appear between coordinate clauses, fields and constituents, so that a single mother node representing the full coordination is accompanied by a KONJ edge label that is assigned to the conjuncts in order to distinguish them from the conjunctions. This edge label has been introduced in TüBa-D/Z, because in some configurations conjuncts otherwise can be confused with conjunctions (Telljohann & al., 2003, p. 94; Ule and Kübler, 2004).

Coordinations that coordinate constituents which are separated not only by conjunctions, but that are realised in more than one field, pose a special problem, because field structure and constituent structure do not both fit easily into a single proper tree. These so-called *split-up coordinations* are therefore represented by a tree for the field structure, and additionally by co-indexed edge labels that connect the coordinated constituents to represent the full coordination. Only the edge labels representing grammatical functions need to be duplicated for this purpose, because only maximal projections of constituents below fields are split between fields, and these maximal

projections always have an edge that describes their grammatical function. Figure 4.8 shows a subject that consists of two coordinated nominal constituents.<sup>13</sup> Their edge labels ON and ONK encode this connection, where



Der türkische Staatspräsident Süleyman Demirel war zutiefst  
 The Turkish president Süleyman Demirel was deeply  
 empört und mit ihm der gesamte Generalstab.  
 outraged and with him the whole General\_Staff.  
 ‘The Turkish president Süleyman Demirel was outraged, as well as  
 the whole General Staff.’

Figure 4.8: Co-Indexation of Split-Up Conjuncts

ONK marks the second conjunct and indicates at the same time that the constituent marked by ON is not the full subject. The fact that ONK can only refer to the ON in the same clause, and not to any other ON (e.g. in an embedded clause) is encoded in their attachment to the same clause. No constituent is split into more than two elements in TüBa-D/Z, so that a pair of node labels per type of relation is sufficient to connect all conjuncts, avoiding an excessive proliferation of categories. There are currently eight additional edge labels for split-up conjuncts: ONK, ODK, OAK, FOPPK, OADVPK, PREDK, MODK, V-MODK; please see appendix A.3 for a detailed description.

There are thus two methods to link the elements of a coordination, depending on their positions in a sentence: edges and a common parent, or co-indexed edge labels.

<sup>13</sup>TüBa-D/Z sentence 10884.

## Underspecification

Coordinations generally group conjuncts of the same or, at least, of similar type. It is thus straightforward to choose a label for the whole coordination that includes information about the type of the conjuncts by choosing the same label for the coordination that labels the conjuncts, if this information should be available at this level at all. If the information about the type of the conjuncts should not be duplicated and percolated up to the label of the whole coordination, then an underspecified label can be used instead. An underspecified label also allows to group conjuncts that differ slightly, but that still belong to the same major group. Both strategies of duplication and underspecification are employed in TüBa-D/Z.

Fields that are coordinated are grouped by a node with a dedicated label for field coordinations (FKOORD). Thus only the information that some fields are coordinated percolates up to the node representing the full coordination, but the node is underspecified with respect to the type of coordinated fields. Most of the time, FKOORD nodes are not direct mothers to field nodes, because the conjuncts normally consist of sequences of field nodes, which are grouped by the specialised node label FKONJ (see figure 4.9(a)<sup>14</sup>). When there is only a single field node, it is directly dominated by the FKOORD node. Single nodes with a field label and sequences of them can also be coordinated. This scheme of underspecified conjuncts for sequences of field labels has been introduced in TüBa-D/Z. In TüBa-D/S (Stegmann & al., 2000), from which originates the annotation scheme of TüBa-D/Z, each field conjunct enumerates the label sequence of its daughters instead of using an underspecified FKONJ label. While this was feasible for spontaneous speech, it turned out to be not useful for the arguably more complex sentence structure in TüBa-D/Z, where a larger number of different combinations of field labels make up the conjuncts, and thus a much larger number of labels was required.

The constituent representing coordinated clauses also has a clause label. It is a label for a simplex clause (SIMPX) when simplex clauses are coordinated. It is a label for paratactic construction of simplex clauses (P-SIMPX) when one of the simplex clauses (usually the second) has a conjunction that occupies the PARORD field for non-coordinating particles (e.g. *denn*, *weil*:

<sup>14</sup>TüBa-D/Z sentence 110.

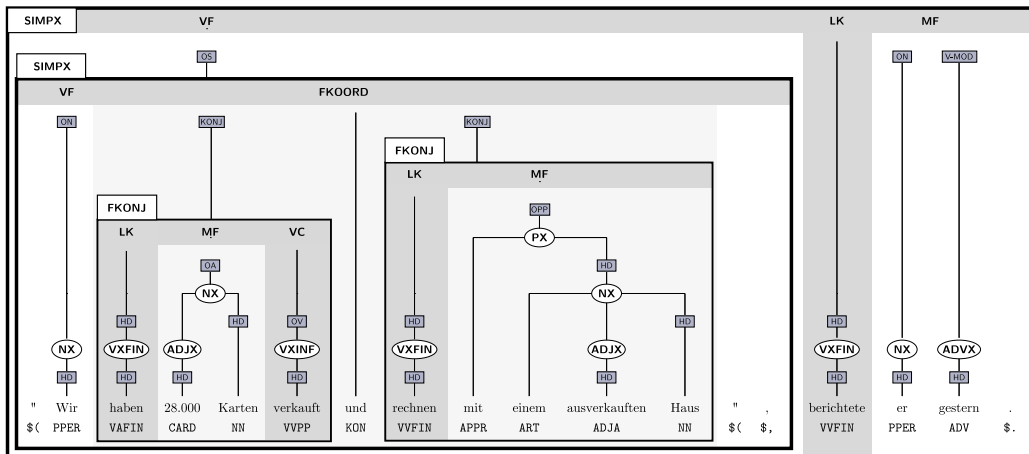
"Wir haben 28.000 Karten verkauft und rechnen mit einem ausverkauften

"We have 28.000 tickets sold and expect with a sold\_out

Haus", berichtete er gestern.

house", reported he yesterday.

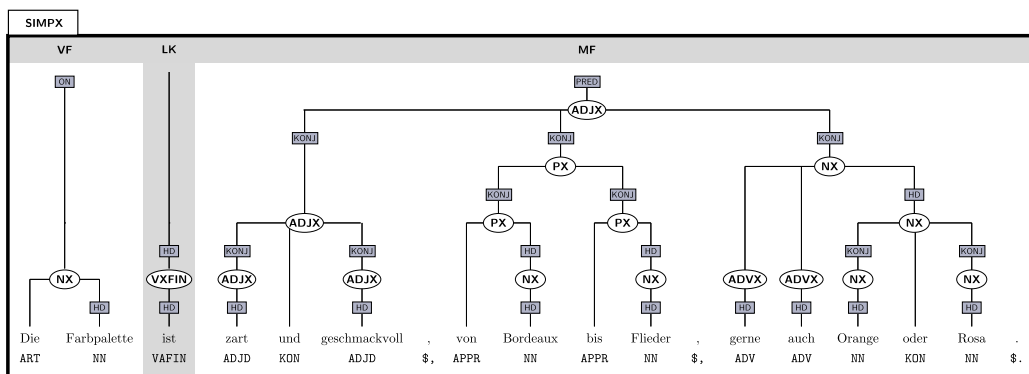
'"We have sold 28.000 tickets and expect to be sold out", he reported yesterday.'



(a) coordinated sequences of fields

Zur Zeit entwickelt er ein Modell, [R-SIMPX  
 [R-SIMPX nach dem er mit einer Agentin zusammenarbeitet ] und  
 [SIMPX damit die bisher blinde Schnittstelle zwischen Groß- und Kleinverlag  
 lukrativ zu machen hofft ] ].

(b) coordinated clauses of different types



(c) coordinated constituents of different types

Figure 4.9: Coordination of Clauses, Fields, and Constituents

‘because’). Relative clauses (R-SIMPX) can also be coordinated, and receive the same label for coordination and conjuncts. These cases represent the vast majority of clausal coordination in TüBa-D/Z. There are cases, however, where a simplex clause is coordinated with a P-SIMPX, or a simplex clause is coordinated with a relative clause. There is no specialised label for this purpose that underspecifies clausal coordination, or that expresses some features of the coordinated sentences when they differ slightly (as does P-SIMPX). Instead, another strategy is adopted, and the label of the first conjunct is used as the label of the whole coordination. Such a coordination of unequal coordinate terms can be observed in the sentence shown in figure 4.9(b), where the label R-SIMPX is neither underspecified as FKOORD was above, nor does it give information that applies only to the full coordination as P-SIMPX.<sup>15</sup>

The same strategy is adopted for constituents. Most of the time, coordinated constituents have exactly the same label, so that the label of the whole coordination is the same as for each conjunct. There are cases, however, where different types of constituents are coordinated (see table 4.2).<sup>16</sup> The high number of non-NX conjuncts is somewhat misleading, because 300 of the 320 labels are EN-ADD labels, which have an arguably nominal flavour (Telljohann & al., 2003, p. 47). However, ten percent of the conjuncts following an adverbial phrase (ADVX) in the same coordination are also not adverbial phrases. The solution to select the label of the first conjunct as the label of the whole coordination neither underspecifies the children's types, nor does it percolate up additional information about mixed-type coordination reliably (see figure 4.9(c) for an example<sup>17</sup>).

Words are rarely direct children of nodes representing coordinations, because normally, each word is projected to the phrase level first. Words are only coordinated directly when they are truncated, or when they directly follow a truncated word in a coordination. A truncated word is POS-tagged with the tag TRUNC, which applies to words ending in a dash, which replaces a part of another word after a conjunction (Schiller & al., 1999, p. 74).

<sup>15</sup>TüBa-D/Z sentence 8522.

Zur Zeit entwickelt er ein Modell, nach dem er mit einer Agentin zusammenarbeitet und damit die bisher blinde Schnittstelle zwischen Groß- und Kleinverlag lukrativ zu machen hofft.  
 At\_the moment develops he a model, according\_to which he with an agent works\_together and so the so\_far blind interface between large and small\_publishing\_houses lucrative to make hopes.

‘At the moment he is developing a model, according to which he works with an agent and thereby hopes to render the gap between large and small publishing houses more lucrative.’

<sup>16</sup>Numbers determined from the first 5000 sentences of TüBa-D/Z.

<sup>17</sup>TüBa-D/Z sentence 2300.

following	first			
	NX	PX	ADJX	ADVX
ADJX	7		558	3
ADVX		1	2	64
EN-ADD	300			
FX	1			
NX	2498	7	4	3
PX	4	312	12	2
VXINF	1			
same	2498	312	558	64
different	320	8	18	8

Table 4.2: Types of First and Following Coordinated Constituents

In summary, coordination is normally encoded via domination, but in certain situations also via co-indexation. Labels of the whole coordination are normally either underspecified or identical to the conjuncts' labels, but they may also be more specific (P-SIMPX), or unrelated to the conjuncts (for coordinations of different types of constituents). The conjuncts are normally nodes, but may also be words when the coordination includes a truncated word.

#### 4.1.5 Named Entities

Individual objects in the world can have names that unambiguously refer to them. These *named entities* are a prominent feature of newspaper articles. Newspapers report new facts about the entities in the world around us, and they use names to identify the entities to which these facts relate. Accordingly, the number of named entities that are found in TüBa-D/Z is rather high: about twenty percent of all noun phrases (NX) in TüBa-D/Z are named entities. The term named entity may also be used to refer to other types of entities that exist only once in the world, e.g. time and date, which are not

---

Die Farbpalette ist zart und geschmackvoll, von Bordeaux bis Flieder, gerne  
The colour\_palette is soft and aesthetic, from Bordeaux to lilac, fine  
auch Orange oder Rosa.  
also orange or pink.

‘The colour palette is soft and aesthetic, from Bordeaux to lilac, also orange or pink if you like.’

marked in TüBa-D/Z (Telljohann & al., 2003, p. 41).

Named entities cannot be contrasted with phrasal categories. Being a named entity is rather an attribute to a constituent that allows it to function as a nominal constituent, while having any phrasal or sentential category internally. Most named entities consist of noun phrases (NX) or single common or proper nouns (NN or NE) internally. Other internal types of proper names are about as frequent as you would expect by their overall frequency in the treebank, with the exception of foreign material (FX), which makes up named entities almost as frequently as single nouns, although the label is much less frequent otherwise. Named entities of a single internal type may serve as various subtypes of named entities, e.g. a prepositional phrase (PX) may serve as a person's name as in *von Beust*, as a street name as in *Unter den Linden*, or as a movie title as in *Gegen die Wand* ('Against the Wall'). Similarly, a single type of named entity may be created from various internal node types. A movie title which forms an entity may be formed from, e.g., an adjectival phrase (ADJX) like in *Schlaflos in Seattle* ('Sleepless in Seattle'), from a foreign language phrase (FX) as in *Shall we Dance*, from a simplex clause (SIMPX) as in *Ich weiß immer noch, was Du letzten Sommer getan hast* ('I still know what you did last summer'), or from a prepositional phrase as in *Gegen die Wand*.

The flexibility of internal structure obviously can result in syntactic analyses that would be hard to justify from a strictly syntactic perspective, where e.g. an adjectival phrase (ADJX) would not be expected to form the head of a prepositional phrase (PX). You would be forced to such kind of analysis without an integration of named entities, though, in the sentence [<sub>PX</sub> *Seit " [ADJX Schlaflos in Seattle ] "* ] *gelten Tom Hanks und Meg Ryan als Dream-Team des Biedersinns*.<sup>18</sup> Named entities are thus attractive as a mediator between constituents that have an opaque internal structure, and the rest of the sentence, for which the whole proper name is an atomic nominal unit (Telljohann & al., 2003, p. 47).

The *export* data model in TüBa-D/Z does not provide attributes to node labels. It only allows to specify atomic node or edge labels. We would like to create annotation that is interpreted to let any constituent function as

<sup>18</sup>TüBa-D/Z sentence 3512.

Seit "Schlaflos in Seattle" gelten Tom Hanks und Meg Ryan als  
 Since "Sleepless in Seattle", are \_considered Tom Hanks and Meg Ryan as \_the  
 Dream-Team des Biedersinns.  
 dream\_team of unsophisticatedness.  
 'Since Sleepless in Seattle, Tom Hanks and Meg Ryan are considered the dream  
 team for an upright couple.'



a named-entity noun phrase as a whole, yet keeping its internal analysis according to the usual syntactic analysis. Keeping the internal syntactic analysis may be helpful to understand inflection of part of the named entity, like the inflected surname *Bock* that corresponds to the function of a genitive pre-modifier in *Humor und Herz beweist* [<sub>NX</sub> *Henning Bocks*] *Inszenierung*. (‘Humour and heart is shown by Henning Bock’s production’)<sup>19</sup>. The solution adopted in TüBa-D/Z to represent a named entity attribute to a strictly syntactic internal analysis is to add a node with the label EN-ADD above the full named entity, where EN-ADD has as its only child a node representing the constituent according to its internal syntactic analysis. For one of the above example sentences, this yields the analysis shown in figure 4.10(a). The in-

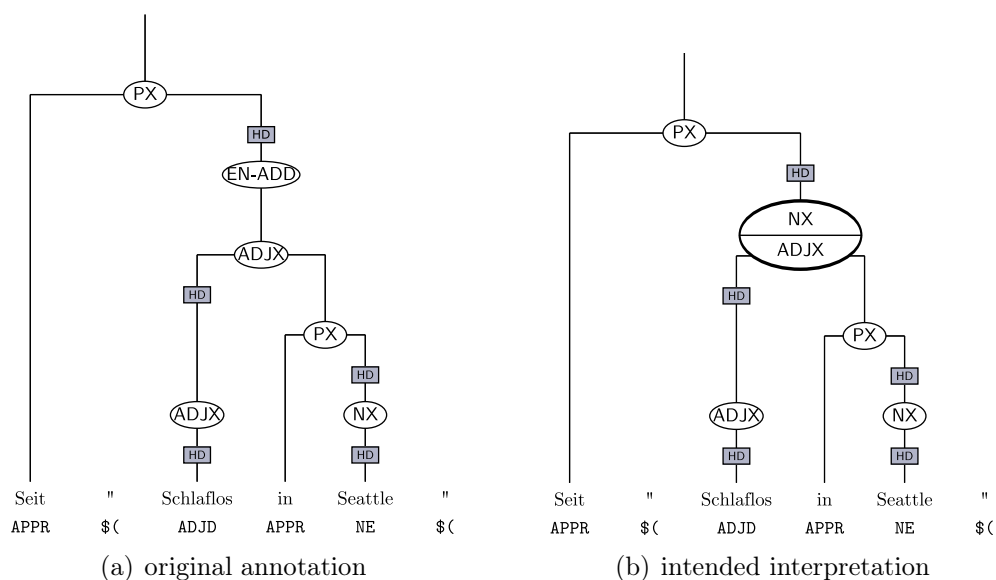


Figure 4.10: Named Entity Encapsulates Internal Analysis in TüBa-D/Z

tended interpretation is shown on the right-hand side in figure 4.10(b), which is underlined by the restrictions on the usage of EN-ADD nodes:

- EN-ADD only has a single child, which is the topmost node of the internal analysis.
- The edge between EN-ADD and its daughter does not have a label.
- The function of the whole named entity is encoded in the edge label connecting EN-ADD to its parent.

<sup>19</sup>TüBa-D/Z sentence 482

An alternative to this solution would be to duplicate all phrasal and clausal node labels, so that each label has a counterpart that encodes the additional function of a named entity just as the NX/ADJX node in figure 4.10(b). The disadvantage would be that many new node labels would have to be introduced, and that the internal syntactic analysis of a proper noun could not be expressed in the same way as the syntactic analysis of equivalent constituents outside proper names, because an NX/ADJX node, viewed atomically, has a different meaning than an ADJX node.<sup>20</sup>

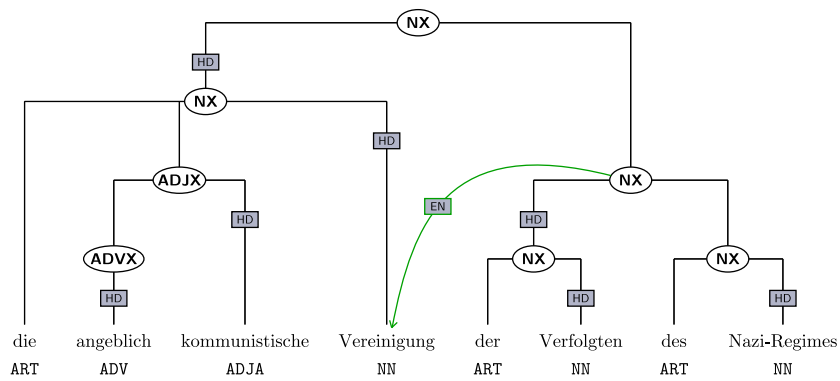
It became apparent rather early during the introduction of named entities into TüBa-D/Z that constituent structure does not always provide a single node that dominates all and only the words that belong to a single named entity. Consider e.g. the named entity *Vereinigung der Verfolgten des Nazi-Regimes* ('Association of the Victims of Nazi Persecution'), where the noun *Vereinigung* is the head of a complex noun phrase which is modified by *der Verfolgten des Nazi-Regimes* according to the annotation principles for post-nominal modification (see figure 4.11(a)<sup>21</sup>; for annotation principles of postmodification see Telljohann & al., 2003, p. 32). In order to encode the differences between the analysis as a named entity and according to purely syntactic principles, a secondary edge labelled EN is added. The edge points from the dependent part of the named entity to its head, so that we can construct an intended interpretation as shown in figure 4.11(b), where the double-sided NX/NX node label again represents a named entity with internal and external structure. This interpretation results in a different tree structure, so that a single proper tree will either lead to an interpretation along the lines of syntactic constituency as in figure 4.11(a), or alternatively to an interpretation where named entities are units with an opaque internal structure that takes precedence over a purely syntactic analysis as in figure 4.11(b), while both analyses can be recovered from the original annotation. An application that focuses on named entities, e.g. may benefit more from a representation as in figure 4.11(b).<sup>22</sup> Secondary edges that encode conflicting analyses always extend the dependent part of the named entity by an uninterrupted sequence of words until it includes the head of the named entity. The additional analysis expressed by the secondary edge labelled EN

---

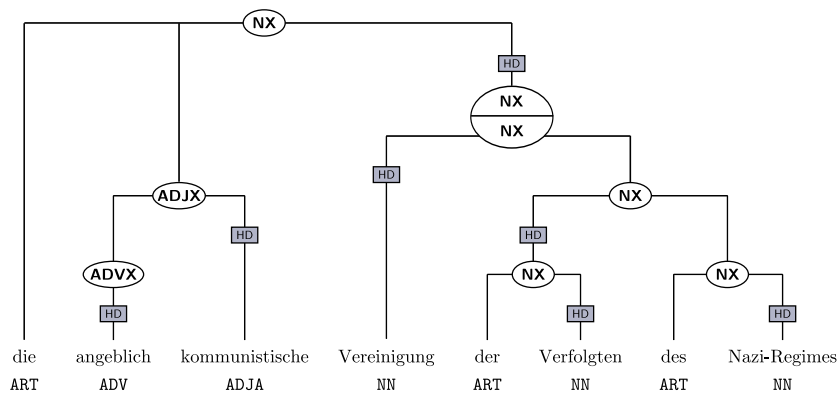
<sup>20</sup>When named entity annotation was introduced into TüBa-D/Z, it was not clear whether it could be integrated seamlessly, so that it was desirable to optionally remove all proper name information and recover the analyses according to the original scheme. The chosen encoding gives you this option.

<sup>21</sup>Part of TüBa-D/Z sentence 557.

<sup>22</sup>Figure 4.11(b) is not equivalent to figure 4.11(a) in that there is no information about how the original annotation can be recovered, which could easily be added, the point of the argument remaining that one analysis is easier to recover than the other.



(a) original annotation



(b) intended alternative interpretation as named entity

Figure 4.11: Conflicting Syntactic and Named Entity Analyses

can therefore always be encoded by a proper tree.

Not all named entities include either a node labelled EN-ADD or a secondary edge labelled EN. Single proper nouns have their own label NE in the STTS tag set (Schiller & al., 1999, p. 11). Compounds that consist of a proper noun and a common noun, however, are tagged as a common noun (NN). Street or place names like *Jannowitzbrücke*/NN are a common example. In order to highlight their status as a named entity despite their NN tag, these compounds are also annotated with an EN-ADD node (see figure 4.12<sup>23</sup>). When two or more adjacent words individually POS-tagged NE

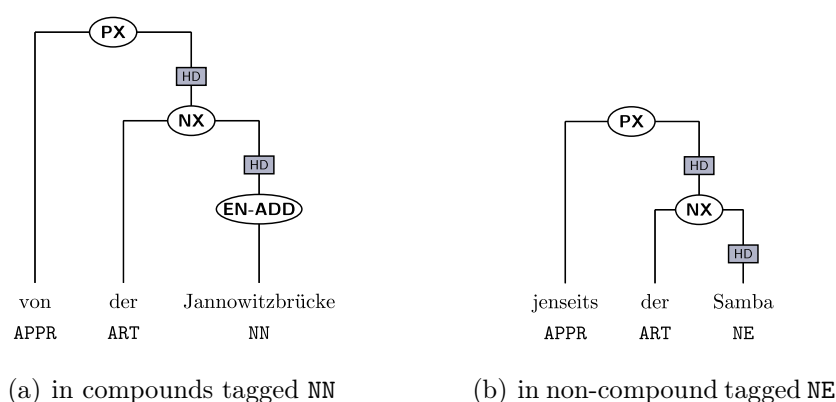


Figure 4.12: Proper Nouns

form a single named entity, then they are projected to a noun phrase (NX) first and only then grouped by EN-ADD. These exceptions arise from the distinction of parts of speech, and the desire to minimise redundancy.

The annotation of proper names thus consists of either a secondary edge EN or a single additional node EN-ADD, which is not given for single words POS-tagged NE, but is given for single proper nouns POS-tagged NN. It encodes a parallel analysis of constituency and named entities. The examples show that the different ways to encode named entities do not all reveal the intended alternative interpretation immediately, because they prefer the syntactic analysis when it conflicts with the alternative analysis as a named entity. The ways in which named entities are encoded in TüBa-D/Z can be understood better when we highlight such diverse constraints as the desire to maintain a context-free backbone, an established POS tag set, or the introduction of desirable, but conflicting alternative analyses. The encoding of named entities resembles the encoding of coordination and free constituent order in this respect.

<sup>23</sup>Part of TüBa-D/Z sentence 1805 and sentence 238.

## 4.2 Representations of Relations

We have just seen a number of phenomena that cannot be easily encoded in the proper trees that are the backbone of TüBa-D/Z's annotation.<sup>24</sup> Nonetheless, these phenomena are captured in the treebank by taking advantage of mechanisms offered by the *export* data model which go beyond the power of proper trees. Understanding the encoding of such phenomena in the treebank is complicated by the fact that the same phenomenon can be encoded in a number of different ways. Named entities, e.g., are encoded by secondary edges, part of speech labels, or additional non-terminal nodes. This means that these building blocks of graphs can have a number of different interpretations.

Nodes, edges, and their labels, which are the parts of the *export* data model, all express certain basic relations in a graph, e.g. the dominance relation between mothers and their daughters, or a categorial status. We have just pointed out that a single type of relation in a tree does not always correspond to a single type of phenomenon in TüBa-D/Z, so that for the parts of the graphs used to encode syntactic annotation, the natural interpretation does not necessarily correspond to the interpretation that is given in treebank documentation. We suggest that the difference between the natural and the suggested interpretation of the annotation graphs can make the interpretation of the analyses in the treebank considerably harder. In order to clarify the natural interpretation of the parts of a graph, we would like to outline the basic types of relations naturally expressed via trees in this section.

We have seen different encodings of the same phenomenon, but we have also seen different representations of a single encoding, e.g. when topological fields are either represented by nodes or by sequential areas in a rectangle as in figure 4.5, or when named entities are either represented by an additional EN-ADD node or by a single two-sided node as in figure 4.10. As we have already seen, these different representations are not fully determined by a certain encoding. This freedom can be used to represent the encoding provided by a treebank in such a way that the natural interpretation of this representation comes closest to the interpretation suggested in the treebank's manual. Different representations encompass small visual variants, like introducing colours to distinguish different parts of a graph, but we also intend to call more fundamental changes in the way that the same information is transported different representations of this information.

Different representations of a certain encoding are not necessarily equivalent. The dependency analysis shown in figure 4.5(b), e.g., does not include

---

<sup>24</sup>Context-free grammars generate proper trees (Telljohann & al., 2004).

the extent and the names of the fields. In a sentence as given in figure 4.6, where middle and final fields (MF, NF) directly follow each other, the fields cannot be recovered from the dependency representation without additional knowledge. We do not specify exactly what we mean by additional knowledge because we do not need any additional knowledge for our purposes, as all transformations that we perform in our experiments in later chapters can be trivially undone by merging given sets of node labels. The dependency representation, however, is not equivalent to the original encoding in this way. Representing topological fields as a sequence as in figure 4.5(c) is equivalent with the original annotation, though. Constituents are rendered as nodes and edges according to the original encoding. Fields are arranged in a sequence according to the sequence of daughters of the clausal node. The rectangular area below a field contains exactly the daughters of the field node in the original tree encoding. The clausal node label is given in the upper left corner of the sequence of fields, and corresponds to the mother node of all field daughter nodes in the original encoding. All terminal nodes that are unconnected by visible edges (e.g. the punctuation marks as in figure 4.5(c)), as well as the top clausal nodes, are connected to the VROOT node by unlabelled edges in the original encoding. The attachment of the punctuation marks to the fields and clauses is not specified in the original encoding. We group punctuation marks with clauses and parentheses according to heuristics described in detail later (see section 5.2.1). As long as the representation is equivalent with the original encoding, we are free to choose the representation that has an interpretation resembling best the intended interpretation of the original encoding according to the treebank manual.

### 4.2.1 Class, Sequence, Domination

Labels of nodes correspond naturally to categories of nodes, and edge labels to categories of relations. A hierarchical structure of categories, i.e. categories with subcategories, can be represented quite naturally by subsets of the full set of all categories. These subsets can not only be used interchangeably, e.g. when a constituent has an edge with a label from the subset of grammatical functions, e.g. either accusative or dative object. Relations can also hold between the members of different subsets, so that relations are expressed via class membership. When one set of nodes has edges with labels from the subclass of grammatical functions, and another set of nodes has labels from the subclass of fields that form the verbal bracket, then these two sets express the relation between the verbal parts of a clause and its arguments by virtue of their class memberships. Their relation can be naturally expressed by co-indexation as just outlined.

Labels are atomic. When a new class has to be defined that shares features with other classes, then this can only be accomplished with a new node label. In order to express e.g. underspecification or ambiguity, a new label has to be specified for each combination of subclasses when an ambiguity has to be represented explicitly, and a new label for each kind of underspecification is required, which limits the usefulness of labels to represent compositional information.

Sequences are naturally represented by a linear sequence of symbols. The words in a sentence are thus normally given in their linear order. Even unattached punctuation marks find their position in a sentence accordingly. Similarly, the fields in a clause always follow each other linearly, so that it seems natural to arrange them on a line as well. In TüBa-D/Z, fields are encoded as labelled nodes in exactly the same way as constituents and clauses. These three major classes of nodes are distinguished by their labels, which form disjunct classes. Despite their similar appearance, fields always appear in linear order, and not in hierarchical relations. They can thus be best rendered as a linear sequence as shown before, where linear precedence between the constituents is recorded as well as their position relative to the sentence bracket.

A rooted tree naturally represents domination, when the edges are all drawn into a single direction, which is usually from the root node at the top of the graph to the leaves at the bottom. A node that dominates another node is represented by edges connecting the node with a node below it in the tree. A node directly dominates another node when there is a single edge between them, and indirectly when a path of connected nodes and edges connects them. More restricted proper trees represent the structure where all words dominated either directly or indirectly by any node form a continuous sequence. Constituent structure and clause structure can be naturally represented by proper trees in most sentences in TüBa-D/Z.

### 4.2.2 Arbitrary Relations and Parallel Structures

Some relations in the syntactic analyses of TüBa-D/Z do not correspond naturally to dominance relations in proper trees. We have seen that free constituent order together with coordination can result in constituents that are split between two fields and are thus discontinuous. We have similarly seen that constituents can modify other constituents despite being realised rather freely and discontinuously at different positions in a sentence. As these relations cannot be expressed via the dominance relations in a proper tree, it is rather natural to represent them via less restricted edges, i.e. arbitrary connections between two nodes, or *secondary edges* in terms of the *export*

model. We have also seen that classes of labels can express arbitrary relations in a graph just as well. Both are used to represent arbitrary relations in the syntactic annotation of TüBa-D/Z.

Proper trees are also not well suited to represent alternative structural analyses as they arise e.g. from genuine ambiguity, or from the desire to express more than one analysis at once, where each analysis corresponds to a different tree. TüBa-D/Z employs different mechanisms to encode alternative analyses, such as secondary edges to encode named entities, or different places of attachment, where a higher attachment of a modifier encodes a more ambiguous modification (Telljohann & al., 2004, p. 80ff.). While alternative and conflicting full tree structures are hard to represent in a single graph, we think that it is rather natural to interpret the height of the attachment of a single node to represent an ambiguous scope.

The difference between what we term a representation and an encoding is that an encoding is given by the way in which a treebank is delivered, while there is more than one representation that can be derived from it. All have a default interpretation, such as a node representing a part of a hierarchy, and a label representing a class. A representation may differ from the encoding, by e.g. representing a node as a part of a linear sequence, or a label as a part of a subclass of labels (as we have proposed for fields). We argue that it is useful to consider the default interpretation of a representation, in order to choose the most suitable representation for the type of relation expressed by part of an annotation graph, as there is some freedom to choose different representations for a single encoding.

### 4.3 Encoding the Analyses

Different representations of a single encoding of syntactic analyses suggest different interpretations, so that more useful representations can be preferred. Also the original encoding of the analyses in TüBa-D/Z that follow the *export* data model is only one of a number of possible ways to encode the analyses. The choice of encodings is restricted in a different way than the choice of representations of a single encoding, though. First, the tool that is used to produce syntactic annotation is the *annotate* tool, which includes its own style to represent graphs. This style is the same as that shown in figure 4.5(a), where all nodes are rendered in the same way, and similarly all edges, secondary edges, and edge and node labels do not have visually distinct subtypes. Subclasses that are defined via subsets of labels of a single type of element in a graph cannot be represented in a distinct way by *annotate*. Second, the chosen encoding targets treebank annotators rather



than treebank users. Annotators will usually have better command of the treebank manual than the average user, and will therefore usually be able to handle more complicated encodings of certain phenomena, even though the representation provided by *annotate* may not be the most natural choice for the given phenomenon. Annotators will also prefer encodings that take best advantage of the automation offered by the annotation tool. When e.g. an arbitrary relation between two elements in a sentence can be encoded either by a secondary edge or by co-indexed edge labels, and the labels are assigned more reliably automatically than the secondary edges, then annotation will be faster due to better automation when edge labels are used to encode the relation.

### 4.3.1 Proper Trees

Generally, the elements of a graph used to encode linguistic analyses have default interpretations as outlined above. Proper trees express hierarchical relations between nodes, and node and edge labels imply that there are classes of nodes and edges. As the proper trees are the “backbone” (Telljohann & al., 2004) of TüBa-D/Z’s annotation, they seem to offer a useful representation of core facts of the treebank’s annotation, even though the default interpretation of trees as dominance relations does not seem perfect for e.g. sequences of topological fields. The encoding is supplemented by the treebank manual, though, so that nodes labels can depart from expressing just identity or difference, and can instead model subclasses of nodes. That is, instead of discriminating subtypes of nodes by their representation, they can be distinguished via the definition of classes of labels in the treebank manual. The meaning of a graph, which normally corresponds to the default interpretation of its visual representation, is then partly overridden by the treebank manual. Using the manual and the treebank side by side is still rather straightforward as long as the different levels of description in the annotation are expressed by different means, e.g. parts of speech via labels on terminal nodes, or coordination via dominance relations in a proper tree. While the former is always true in TüBa-D/Z, the latter is not, because syntactic relations such as coordinations can become so complex that they are hard to express via proper trees and are consequently modelled by other means.

### 4.3.2 Crossing Edges, Co-Indexation, and Secondary Edges

When other means than proper trees need to be used to express syntactic relations, then the *export* model offers more than one way to do that.

Crossing edges are ruled out by design in TüBa-D/Z, but secondary edges are subject to even less restrictions than crossing edges, and we have also seen that co-indexed labels can link arbitrary parts of a graph just as well, given an appropriate label set. Both co-indexation and secondary edges are used in TüBa-D/Z for different purposes that have been discussed separately above. We would like to recall here that co-indexation distinguishes subtypes of node labels, including constituents, fields, and clauses. The labels representing parts of speech on terminal nodes are also disjunct from all other labels. There are also subclasses of edge labels specifying heads and non-heads inside phrases, and those that mark different types of complements or modifiers. Finally, there is another subclass of edge labels for conjuncts of modifiers, and one more for conjuncts of complements. Co-indexation works with a modest amount of edge labels to link the complements and modifiers to a verb, and the conjuncts of a coordination, because the scope of nodes to which the edge labels point is restricted to those within the same class. That is, classes of node labels (for fields and clauses) interact with other classes of labels (edges for modifiers and complements) when they are in a certain dominance relation (edge labels of nodes within the fields of a clause). The semantics of this interaction is specified in the treebank manual. When classes of nodes are in similar relations as expressed by the edge labels, but their position in the syntactic annotation graph does not allow the application of co-indexation via edge labels, then secondary edges are used for the same purpose (the *refint/refcontr* edges for phrase internal/clause external modification). Secondary edges are also used to extend the number of distinctions in the label set for modifier edges (via *refmod*), or for the edge labels of verbal complements (*refvc*). Finally, secondary edges are used to express alternative analyses that assign named entities a more prominent status.

The idea to use a restricted inventory of relations, and to use more constrained graphs to describe more constrained phenomena, comes, of course, from the desire to express the observations about syntactic structure in the most accessible way. E.g., accusative objects occur alone more often than with a modifier, than with a modifier that has a modifier itself. A modifier to a modifier occurs even more rarely in situations where there is more than one other modifier in the same clause. It thus seems to be a straightforward solution to assign the accusative object its own edge label *OA*, and also to define an edge label *OA-MOD* for a modifier of an accusative object, and another edge label for a modifier of a modifier *MOD-MOD* that points to *OA-MOD*. Distinguishing ambiguous modifiers of modifiers explicitly via edge labels instead would result in many more edge labels like *OA-MOD-MOD*. The approach of TüBa-D/Z to use the secondary edge labelled *refmod* for these rather infrequent cases instead seems to be an elegant solution, even though

only the treebank manual shows the rather complex interaction between all these types of edges and edge labels.

Technically speaking, also unattached elements do have edges in the original encoding of the annotation of TüBa-D/Z, which link the unattached elements to VROOT. These edges often cross other edges, e.g. for commas that introduce a relative clause. These edges thus have a semantics that is transported well by their visual representation as an invisible edge. A representation that stays near the formal encoding, though, is less easy to interpret directly, because it is not immediately clear what it means to be connected directly to the root node of the sentence.

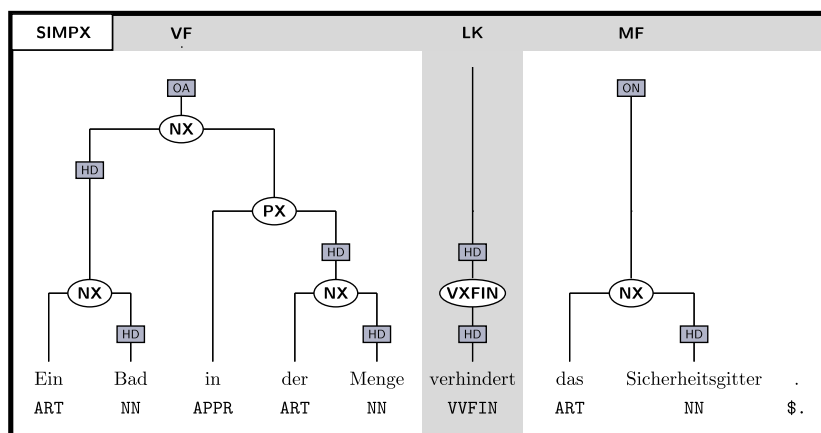
### 4.3.3 The Semantics of a Small Label Set

The desire to reduce the size of the label set and recurring to more powerful means in more complicated situations as outlined above may render the application of the labels more difficult. Some information about dependencies is only available via default interpretations given in the treebank manual. In verbal complexes with two or more parts, each part selects the part preceding it. When there are two parts, the last one is assigned a HD edge to mark it as a head. When there are three or more parts, these dependencies are individually given via secondary edges labelled *refvc*, so that dependencies are all given rather explicitly by changes in the annotation. The dependencies between pre-nominal modifiers, on the other hand, are not given equally explicitly in the annotation, but are specified only in the manual to hold between adjacent nodes from left to right (Telljohann & al., 2003, p. 30).

A definition of default readings via the manual can be even harder to interpret for the annotation of grammatical functions when the case of the complements is ambiguous. According to the manual, “if case assignment is ambiguous, we decide on the more plausible grammatical function respectively on the more plausible sequence of grammatical functions” (Telljohann & al., 2003, p. 80). As a consequence, the sentence in figure 4.13 is not distinguishable from a sentence where cases of the complements are not ambiguous.<sup>25</sup> The given sentence, of course, is semantically not ambiguous, so that the ambiguity of case in the articles can be resolved by the annotators nonetheless. It also seems that a sentence which is truly ambiguous and that cannot be resolved by the annotators by semantic means is rather a theoretical consideration, because such a sentence does not occur in the treebank, probably exactly because speakers try to prevent the ambiguity. The attachment site of a modifier also represents ambiguity of scope in a way

---

<sup>25</sup>Example from (Telljohann & al., 2003, p. 79).



Ein Bad in der Menge verhindert das Sicherheitsgitter.  
 A bath in the crowd prevents the security\_gate.  
 ‘The security gate prevents a bath in the crowd.’

Figure 4.13: Ambiguous Case Not Given in Edge Labels

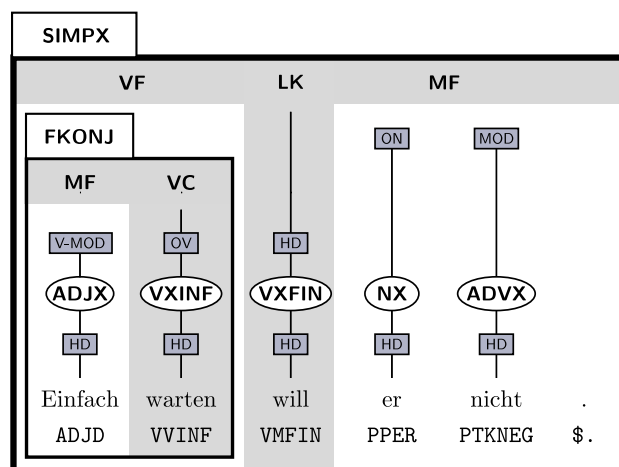
that can be misinterpreted. Higher attachment does not mean that all nodes c-commanded by the modifier are modified, but just that the modifier could modify any one node c-commanded by it.

Reluctantly adding new node labels in order to keep the number of different node labels manageable can also result in rather different meanings of a single label. The FKONJ label, which groups coordinated sequences of fields by default, is also used to group sequences of fields appearing in the initial field (VF) when parts of the sentence are topicalised. Figure 4.14 shows an example.

## 4.4 Conclusion

The shape of syntactic annotation as we find it in a treebank like TüBa-D/Z is determined by a number of factors that are not fully independent. It can be seen from different perspectives, and we think that it is useful to distinguish the encoding of the annotation in some kind of data structure, the visual representation of these data structures that follows some conventions, and finally the information that should be transported in the analyses. These different perspectives help to keep track of the different needs that a treebank is trying to serve, and to highlight the freedom to meet any of the needs independent of the others.

It will be paramount for a treebank to carry interesting information and



Einfach warten will er nicht.  
 Simply to\_wait wants he not.  
 ‘He does not want to simply wait.’

Figure 4.14: FKONJ Groups Sequences of Topicalised Fields

to be easy to use at the same time.<sup>26</sup> It should be kept in mind, though, that the degree in which this general goal is accomplished is usually judged by the interpretation of a certain representation of the analyses. What is more, the analyses are accompanied by a number of interacting basic principles and details about the application of these principles in less standard situations, that are only available via the treebank manual. In the end, what is reasonably simple and revealing depends on the individual needs of anyone who interacts with the treebank.

The annotators are primarily interested in consistency and speed. A small and possibly overloaded set of labels might best meet their needs, accompanied by the support of existing tools. They will also accept to reuse existing standards to achieve a steep learning curve. All of this may be at the expense of customised solutions and obvious representations for all analyses. Treebank users will be happy when their linguistic assumptions help them understand the rendering of the analyses, i.e. when what they have learned to be the linguistic interpretation of a certain graph corresponds to the interpretation suggested by the treebank manual. Finally, a machine that uses the annotated data will also need an understanding of the analyses, while its understanding of generalisations and dependencies may be again different,

<sup>26</sup>Syntactic analyses have long been requested to be “reasonably simple and revealing” (Chomsky, 1956).

making yet another representation of the analyses desirable.

Even when these representations are not equivalent, it can be useful to derive different representations from a single treebank (Nivre, 2003). Dipper & al. (2004) show that a similar strategy can also help adapting the treebank manual to the different needs of treebank users. In the following chapters, we will try to study how the freedom of representation of syntactic analyses can be used to let probabilistic context-free models of syntax make better use of the data provided in treebanks. In this goal we follow the designers of TüBa-D/Z, whose “annotation scheme tries to strike a balance between pragmatic requirements on the one hand and linguistic reality on the other hand” (Telljohann & al., 2003, p. 15).

# Chapter 5

## PCFG Treebank Grammars

In this chapter we introduce standard methods for deriving probabilistic context-free grammars from treebanks. Seemingly arbitrary choices of representations of syntactic structure can have considerable impact on the performance of parsers based on PCFG treebank grammars (section 5.1). With that in mind, we show how we extract a probabilistic context-free grammar from TüBa-D/Z, especially with respect to those characteristics of the original encoding of the treebank for which there is no straightforward representation that is compatible with a PCFG (section 5.2).

The analyses in a treebank all follow the same principles, which are laid out in a manual that accompanies the treebank. This manual can be understood as a descriptive grammar with an emphasis on example sentences collected in the treebank. This perspective suggests that the regularities of the syntactic analyses of the treebank form a kind of grammar by example: a *treebank grammar*. This term has been coined rather soon after the advent of the Penn Treebank, for PCFGs directly read off it (Charniak, 1996), and later came to be used for many kinds of computational (and mostly probabilistic) grammars derived from treebank data (Abeillé, 2003, gives several examples). A reason for early treebank grammars being mostly PCFGs may be that the encoding of the Penn Treebank uses proper trees with node labels, which can be translated into a PCFG by rather straightforwardly mapping the original annotation to the nodes and rules of a PCFG. In addition to the formal similarity between the encoding of treebank data and the structure assumed by the grammar, there is, however, also the question whether the semantics of node labels is captured well by the computational grammar, and whether the distribution of analyses in the treebank is faithfully captured by the derived grammar. We will mainly focus on the questions of probabilistic faithfulness and formal similarity in the following two sections.

The next section shows that the seemingly obvious translations from tree-

banks to PCFGs do not necessarily exhibit the desired probabilistic behaviour when used in a PCFG model, even if PCFGs are powerful enough to produce the annotation graphs found in the treebank.

## 5.1 Syntactic Bias of a PCFG Treebank Grammar

Syntactic annotation in a treebank is usually encoded as directed acyclic graphs with labelled nodes, which can be represented as a tree with words as leaves and possibly crossing edges. If each tree is rooted, and its edges do not cross, then the rules of the context-free grammar that are needed to license a sentence can be easily obtained by adding a rule  $A \rightarrow \beta$  to the CFG for each non-terminal node  $A$  in the tree, where  $\beta$  is the sequence of direct children of  $A$ , and by including all encountered non-terminal or terminal symbols into  $\mathcal{N}$  or  $\mathcal{T}$ . If there is a single type of root node for all sentences, then this is the start symbol  $S$  of the CFG; if there is more than one root node in the sentences, then a new nonterminal is added to  $\mathcal{N}$ , which will be the CFG's start symbol  $S$ , and for each type of sentence root in the treebank, a new rule is added to the set of rules  $\mathcal{R}$ , where the new rule's left-hand side is  $S$ , and the type of the sentence root node is the rule's right-hand side. The role of  $S$  is the same as that of the export model's special VROOT node label (Brants, 1997): it is the single root node which is the parent of all otherwise orphaned nodes in each sentence, where the connection of these nodes to VROOT is not shown in the standard graphical representation. The PCFG treebank grammars we derive from TüBa-D/Z all have VROOT as their start symbol. We will use VROOT and  $S$  interchangeably; the former for examples from TüBa-D/Z, the latter for more general examples. A terminological clarification seems to be in order here on node labels in an annotation graph, syntactic categories in a syntactic analysis, and (non-)terminal symbols in CFGs. The (non-)terminal symbols in CFGs cannot be separated from their type, while the nodes in an annotation graph, and in a syntactic analysis have separate labels, or separate categories, respectively. All three essentially mean the same when each node has an atomic non-empty type, so that we employ all three variants depending on the context.

When we build our grammar only from rules that we find in a treebank, then the frequency of a rule and of the non-terminal symbol on its left-hand side in the syntactic graphs of all sentences of the treebank can be used to determine the maximum-likelihood estimate of the probability that the non-terminal expands to this rule in a PCFG. Given that the rule  $A \rightarrow \beta$  occurs



$freq(A \rightarrow \beta)$  times in the treebank, and the non-terminal  $A$  occurs  $freq(A)$  times (possibly also in other rules), then the maximum-likelihood estimate of the expansion probability can be calculated as

$$P(A \rightarrow \beta|A) = \frac{freq(A \rightarrow \beta)}{freq(A)}$$

Obtaining a PCFG from a treebank is thus straightforward (Charniak, 1996), and has soon become the basis for high-performing parsers (e.g. on the Penn Treebank: Charniak, 1997; Collins, 1997). Simply deriving rules from the original annotation and using their relative frequencies as probabilities, estimates these probabilities according to the maximum likelihood principle, i.e. the model probabilities are chosen so that the probability of the data is maximised. This is the parameter setting that makes it most probable that the rule occurs as many times as it is actually observed, given the constraints of the probabilistic model. In particular, no probability mass is allocated for events not encountered, i.e. the resulting model assigns perfect probabilities to seen data, but it is deficient for events not encountered before, to which it assigns zero probability. This is a problem, because in the end, the treebank is only an incomplete sample of the language we seek to describe.

Maximum-likelihood estimation of a treebank grammar that is already present in the form of proper trees without crossing edges or other complications thus is remarkably straightforward. Treebanks are often not easy to map to a context-free representation in such an unambiguous way, though. They often contain information where several representations are possible, and none is explicitly preferred by the corpus providers. Before we turn to those representational issues that force us to choose a new representation for parts of the treebank, we will investigate the effect of seemingly equivalent choices for representing aspects of syntactic descriptions. We call them seemingly equivalent, because they can be represented directly by a CFG, but providers of different corpora choose different representations, with negligible consequences for the interpretation of the main syntactic observation when presented to a human user. We are in a position where two quite comparable representations of similar information are available for German, namely that of the negra and TüBa-D/Z treebanks. We use these two existing annotation schemes and consider a made-up example of prepositional phrase attachment. We contrast the representation of high and low attachment of the prepositional phrase in both schemes and examine the consequences of the different representations of the same contrast for the preferred parses in a derived PCFG treebank grammar. The observation that certain representations of syntactic phenomena have considerable impact on the interpretation under derived PCFG models is not new. Johnson (1998) has discussed theoretically

inspired examples for English. We follow up with examples resembling the annotation in two existing German corpora.

### 5.1.1 Bias in TüBa-D/Z and negra

PCFGs condition the expansion of rules only on the parent of the tree fragment corresponding to the rule: the probability  $P(A \rightarrow \beta|A)$  of a rule  $A \rightarrow \beta$  depends only on the type of  $A$ . A made-up example shows the consequences for assigning syntactic structure. The two sentences *Er ertappt den Dieb mit dem Fahrrad.* vs. *Er verfolgt den Dieb mit dem Fahrrad.* are intended to highlight the difference between the two verbs *verfolgen* (‘chase’) and *ertappen* (‘catch red-handed’), and the different requirements of the verbs for the attachment of the prepositional phrase *mit dem Fahrrad* (‘with the bike’) as part of the noun phrase *den Dieb mit dem Fahrrad* (‘the thief with the bike’) or as optional prepositional phrase to the verb *verfolgen* (see figure 5.1). We

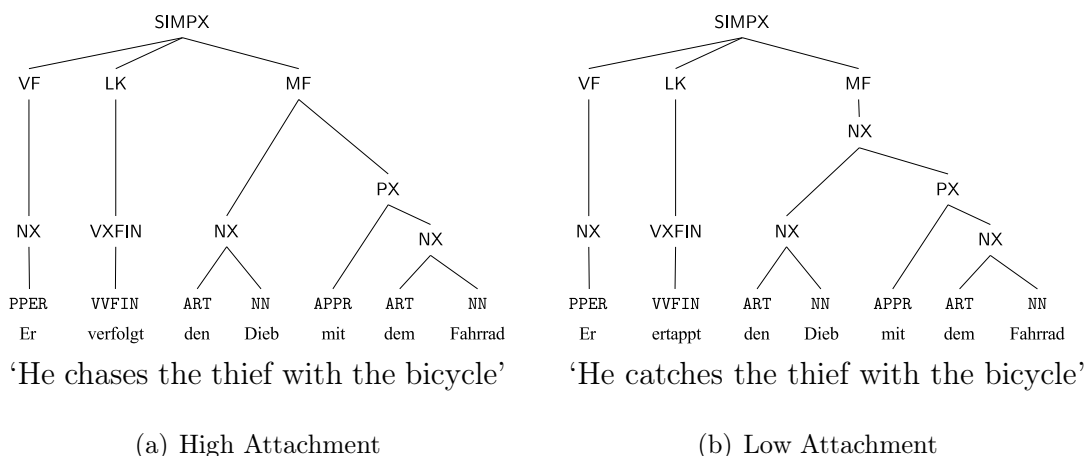


Figure 5.1: Artificial Example of German PP attachment in TüBa-D/Z

first show syntactic analyses for both sentences according to the TüBa-D/Z annotation scheme. For the moment we ignore edge labels, which carry information about headedness and grammatical functions (including optionality), and only consider the major clause, field, and constituent node labels for the sake of simplicity.

We label the two sentences (a) (*verfolgen*) and (b) (*ertappen*), and imagine a corpus with  $n$  sentences that is made up exclusively of  $h$  times sentence (a) and  $l$  times sentence (b) with at least one of each sentences ( $h, l \geq 1; h \neq l$ ). Rule probabilities that are read off the treebank by maximum

likelihood estimation are as follows:

$$\begin{array}{ll}
P(\text{SIMPX} \rightarrow \text{VF LK MF}) = 1 & P(\text{VVFIN} \rightarrow \text{verfolgt}) = \frac{h}{h+l} \\
P(\text{VF} \rightarrow \text{NX}) = 1 & P(\text{VVFIN} \rightarrow \text{ertappt}) = \frac{l}{h+l} \\
P(\text{PPER} \rightarrow \text{Er}) = 1 & P(\text{NX} \rightarrow \text{PPER}) = \frac{h+l}{3h+4l} \\
P(\text{LK} \rightarrow \text{VXFIN}) = 1 & P(\text{NX} \rightarrow \text{ART NN}) = \frac{2h+2l}{3h+4l} \\
P(\text{VXFIN} \rightarrow \text{VVFIN}) = 1 & P(\text{MF} \rightarrow \text{NX PX}) = \frac{h}{h+l} \\
P(\text{PX} \rightarrow \text{APPR NX}) = 1 & P(\text{MF} \rightarrow \text{NX}) = \frac{l}{h+l} \\
P(\text{ART} \rightarrow \text{den}) = 1/2 & P(\text{NX} \rightarrow \text{NX PX}) = \frac{l}{3h+4l} \\
P(\text{NN} \rightarrow \text{Dieb}) = 1/2 & \\
P(\text{APPR} \rightarrow \text{mit}) = 1 & \\
P(\text{ART} \rightarrow \text{dem}) = 1/2 & \\
P(\text{NN} \rightarrow \text{Fahrrad}) = 1/2 & 
\end{array}$$

The probability of the first parse (a) is the product of all rules in the corresponding derivation (only non-unity rules are given):

$$\begin{aligned}
P(\phi^a | w^a) &= P(\text{NX} \rightarrow \text{PPER}) P(\text{VVFIN} \rightarrow \text{verfolgt}) \\
&\quad P(\text{MF} \rightarrow \text{NX PX}) \\
&\quad P(\text{NX} \rightarrow \text{ART NN}) P(\text{ART} \rightarrow \text{den}) P(\text{NN} \rightarrow \text{Dieb}) \\
&\quad P(\text{NX} \rightarrow \text{ART NN}) P(\text{ART} \rightarrow \text{dem}) P(\text{NN} \rightarrow \text{Fahrrad})
\end{aligned}$$

The probability of the second parse (b) given the *first* sentence (a) as input is accordingly:

$$\begin{aligned}
P(\phi^b | w^a) &= P(\text{NX} \rightarrow \text{PPER}) P(\text{VVFIN} \rightarrow \text{verfolgt}) \\
&\quad P(\text{MF} \rightarrow \text{NX}) P(\text{NX} \rightarrow \text{NX PX}) \\
&\quad P(\text{NX} \rightarrow \text{ART NN}) P(\text{ART} \rightarrow \text{den}) P(\text{NN} \rightarrow \text{Dieb}) \\
&\quad P(\text{NX} \rightarrow \text{ART NN}) P(\text{ART} \rightarrow \text{dem}) P(\text{NN} \rightarrow \text{Fahrrad})
\end{aligned}$$

Parse (a) will be selected for sentence (a) when its probability is higher than

parse (b):

$$\begin{aligned}
& P(\phi^a|w^a) > P(\phi^b|w^a) \\
\Leftrightarrow & P(\text{MF} \rightarrow \text{NX PX}) > P(\text{MF} \rightarrow \text{NX}) P(\text{NX} \rightarrow \text{NX PX}) \\
\Leftrightarrow & \frac{h}{h+l} > \frac{l}{h+l} \frac{l}{3h+4l} \\
\Leftrightarrow & \frac{h(3h+4l)}{l^2} > 1 \tag{5.1}
\end{aligned}$$

Resolving equation 5.1 for  $h$  we have:

$$h > \frac{\sqrt{7}-2}{3} l$$

Calculating the ratio for sentence (b):

$$P(\phi^a|w^b) > P(\phi^b|w^b)$$

to find the more likely parse of sentence (b) yields exactly the same result, because  $P(\text{VVFİN} \rightarrow \text{verfolgt})$  is replaced by  $P(\text{VVFİN} \rightarrow \text{ertappt})$  on both sides of the inequation and can therefore be cancelled out:

$$\begin{aligned}
& P(\phi^a|w^b) > P(\phi^b|w^b) \\
\Leftrightarrow & P(\phi^a) > P(\phi^b) \\
\Leftrightarrow & h > \frac{\sqrt{7}-2}{3} l \\
\Leftrightarrow & h > 0.22 l \tag{5.2}
\end{aligned}$$

As a result, we will either always assign parse (a) or parse (b) to all input sentences, only depending on the ratio  $h/l$  in the corpus. In other words, the parse is selected without considering the impact of the verb on the syntactic structure, but only by comparing the likelihood of small disconnected tree fragments that are used in the PCFG models. This means that on top of ignoring *lexical preferences*, “PCFGs are also deficient on purely structural grounds” (Manning and Schütze, 1999, p. 419–421). PCFGs incorporate preferences for larger unlexicalised subtrees, but these *structural preferences* do not match the original data in our example.

The deficiency to model lexical preferences is partly expected for an unlexicalised grammar, and the deficiency to model structural preferences highlights the well-known independence assumptions, which shed a light on the importance of node labels, on which rest expansion probabilities. An undesirable consequence of unclear structural preferences is that the ratio of the

maximum-likelihood estimate of the probabilities of the two parses is different from the ratio of their relative frequencies in the corpus, i.e. the parse occurring more frequently in a corpus may be never selected: equation 5.2 is always satisfied for, e.g.,  $h = 1/3 l$ , and the less frequent parse (a) will always be selected in the resulting PCFG treebank grammar. We will return to discussing lexical and structural preferences of PCFGs in section 5.1.3 below after concluding with our examples.

We now compare the syntactic annotation in the TüBa-D/Z scheme with the annotation of the same pair of sentences in the negra annotation scheme, which is given in figure 5.2. The probabilities, again extracted from a corpus

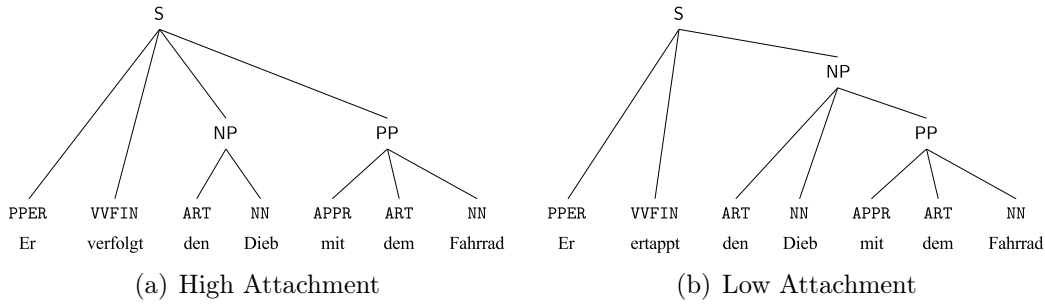


Figure 5.2: Artificial Example of German PP attachment in negra

consisting of  $h$  times sentence (a) and  $l$  times sentence (b) by maximum likelihood estimation are as follows:

$$\begin{aligned}
 P(\text{PPER} \rightarrow \text{Er}) &= 1 \\
 P(\text{ART} \rightarrow \text{den}) &= 1/2 \\
 P(\text{NN} \rightarrow \text{Dieb}) &= 1/2 \\
 P(\text{APPR} \rightarrow \text{mit}) &= 1 \\
 P(\text{ART} \rightarrow \text{dem}) &= 1/2 \\
 P(\text{NN} \rightarrow \text{Fahrrad}) &= 1/2 \\
 P(\text{PP} \rightarrow \text{APPR ART NN}) &= 1 \\
 P(\text{S} \rightarrow \text{PPER VVFIN NP PP}) &= \frac{h}{h+l} \\
 P(\text{VVFIN} \rightarrow \text{verfolgt}) &= \frac{h}{h+l} \\
 P(\text{NP} \rightarrow \text{ART NN}) &= \frac{h}{h+l} \\
 P(\text{S} \rightarrow \text{PPER VVFIN NP}) &= \frac{l}{h+l} \\
 P(\text{VVFIN} \rightarrow \text{ertappt}) &= \frac{l}{h+l} \\
 P(\text{NP} \rightarrow \text{ART NN PP}) &= \frac{l}{h+l}
 \end{aligned}$$

so that the parse (a) is preferred over parse (b) for sentence (a) if

$$\begin{aligned}
 & P(\phi^a|w^a) > P(\phi^b|w^a) \\
 \Leftrightarrow & P( S \rightarrow \text{PPER VVFIN NP PP} ) \\
 & P( \text{NP} \rightarrow \text{ART NN} ) > P( S \rightarrow \text{PPER VVFIN NP} ) \\
 & P( \text{NP} \rightarrow \text{ART NN PP} ) \\
 \Leftrightarrow & \frac{h}{h+l} \frac{h}{h+l} > \frac{l}{h+l} \frac{l}{l+h} \\
 \Leftrightarrow & h > l \tag{5.3}
 \end{aligned}$$

Again, the parse selection is independent of the input sentence, because the lexical probabilities of the verbs appear identically on both sides of the inequation and are cancelled out. Thus we prefer parse (a) over parse (b) for both sentences whenever sentence (a) is more frequent in our corpus than sentence (b).

The difference between equation 5.1 for TüBa-D/Z and equation 5.3 for negra is caused by differences in the encoding of high and low attachment of the prepositional phrase. TüBa-D/Z introduces a new noun phrase node on top of the existing noun and prepositional phrase nodes while negra detaches the edge connecting the prepositional phrase node and the sentence node in (a) and reattaches the edge to the noun phrase node in (b), keeping the number of nodes constant. The new node that TüBa-D/Z introduces in (b) instead has the same label as non-recursive noun phrases, effectively overloading the NX label to be used in a more general way.

### 5.1.2 Extending the Label Set

Another quite straightforward mapping of negra, or TüBa-D/Z annotation to a PCFG is to include edge labels as decoration to node labels, so that the edge label from the edge that links a node to its mother is appended to the node label. Appendix B shows that subcategorising all node labels accordingly changes the ratio between the probabilities of parse (a) and (b) to  $h/l$  also for TüBa-D/Z, which satisfies the obvious desideratum that the full parses should have the same probabilities according to the PCFG as the parses have in the data from which the PCFG was derived. Adding all edge information, however, also yields a higher number of node labels, and consequently a higher number of rule types (18 vs. 20 for using no vs. all edge label information in our toy example).

We can also try to achieve the same result by renaming just a single node in the TüBa-D/Z annotation: either the top NX of the complex noun phrase,

or the NX that directly dominates the head of the complex noun phrase, effectively splitting the node label according to its more specific function (recursive vs. non-recursive). In the first case, we percolate up the internal information about recursiveness to the top of the phrase, reading  $\text{NX\#1}$  as “NX dominating another NX”. The suffix “#1” is arbitrary, and we read it as “the first change”. In the second case we percolate it down nearer to the lexical head of the phrase ( $\text{NX\#1}$  is an “NX dominated by another NX”). In both cases, we add a single new non-terminal symbol to our grammar, where changing the upper NX does not increase the number of rules, and changing the lower NX increases it by one (see figure 5.3). Changing the

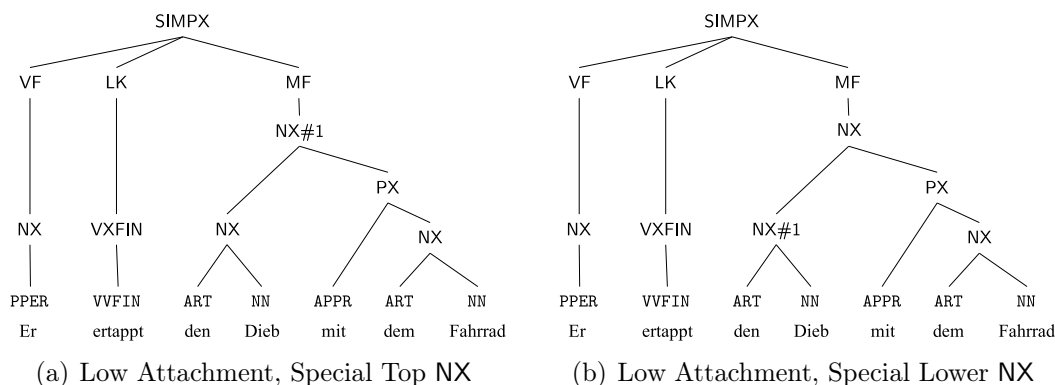


Figure 5.3: Modifications of the Low Attachment from figure 5.1

upper node label to a unique new name (figure 5.3(a)) yields the following rule probabilities:

$$\begin{aligned}
 P(\text{SIMPX} \rightarrow \text{VF LK MF}) &= 1 & P(\text{VVFIN} \rightarrow \text{verfolgt}) &= \frac{h}{h+l} \\
 P(\text{VF} \rightarrow \text{NX}) &= 1 & P(\text{VVFIN} \rightarrow \text{ertappt}) &= \frac{l}{h+l} \\
 P(\text{PPER} \rightarrow \text{Er}) &= 1 & P(\text{NX} \rightarrow \text{PPER}) &= \frac{h+l}{3h+3l} \\
 P(\text{LK} \rightarrow \text{VXFIN}) &= 1 & P(\text{NX} \rightarrow \text{ART NN}) &= \frac{2h+2l}{3h+3l} \\
 P(\text{VXFIN} \rightarrow \text{VVFIN}) &= 1 & P(\text{MF} \rightarrow \text{NX PX}) &= \frac{h}{h+l} \\
 P(\text{PX} \rightarrow \text{APPR NX}) &= 1 & P(\text{MF} \rightarrow \text{NX\#1}) &= \frac{l}{h+l} \\
 P(\text{ART} \rightarrow \text{den}) &= 1/2 & P(\text{NX\#1} \rightarrow \text{NX PX}) &= 1 \\
 P(\text{NN} \rightarrow \text{Dieb}) &= 1/2 & & \\
 P(\text{APPR} \rightarrow \text{mit}) &= 1 & & \\
 P(\text{ART} \rightarrow \text{dem}) &= 1/2 & & \\
 P(\text{NN} \rightarrow \text{Fahrrad}) &= 1/2 & &
 \end{aligned}$$

The relation between the probabilities of the two parses becomes the same

as the relation between the frequencies of each parse in the treebank again:

$$\begin{aligned}
& P(\phi^a|w^a) > P(\phi^b|w^a) \\
\Leftrightarrow & P( \text{MF} \rightarrow \text{NX PX} ) \\
& P( \text{NX} \rightarrow \text{ART NN} ) > P( \text{MF} \rightarrow \text{NX\#1} ) \\
& P( \text{NX\#1} \rightarrow \text{NX PX} ) \\
& P( \text{NX} \rightarrow \text{ART NN} ) \\
\Leftrightarrow & \frac{h}{h+l} \frac{2h+2l}{3h+3l} > \frac{l}{h+l} 1 \frac{2h+2l}{3h+3l} \\
\Leftrightarrow & h > l \tag{5.4}
\end{aligned}$$

The effect is the same as adding all edge information (see appendix B), but the number of new non-terminals is lower than when all edge information is added. Adding fewer non-terminals is generally preferable, because this generally results in fewer rule types that have to be estimated from the same amount of data.

The lower NX node cannot be assigned a new label to achieve the same result. Imagining a treebank consisting of  $l$  times the parse shown in figure 5.3(b), plus  $h$  times the parse from figure 5.1(a) yields the following rule probabilities:

$$\begin{array}{ll}
P( \text{SIMPX} \rightarrow \text{VF LK MF} ) = 1 & P( \text{VVFIN} \rightarrow \text{verfolgt} ) = \frac{h}{h+l} \\
P( \text{VF} \rightarrow \text{NX} ) = 1 & P( \text{VVFIN} \rightarrow \text{ertappt} ) = \frac{l}{h+l} \\
P( \text{PPER} \rightarrow \text{Er} ) = 1 & P( \text{NX} \rightarrow \text{PPER} ) = \frac{h+l}{3h+3l} \\
P( \text{LK} \rightarrow \text{VXFIN} ) = 1 & P( \text{NX} \rightarrow \text{ART NN} ) = \frac{2h+l}{3h+3l} \\
P( \text{VXFIN} \rightarrow \text{VVFIN} ) = 1 & P( \text{NX\#1} \rightarrow \text{ART NN} ) = 1 \\
P( \text{PX} \rightarrow \text{APPR NX} ) = 1 & P( \text{MF} \rightarrow \text{NX PX} ) = \frac{h}{h+l} \\
P( \text{ART} \rightarrow \text{den} ) = 1/2 & P( \text{MF} \rightarrow \text{NX} ) = \frac{l}{h+l} \\
P( \text{NN} \rightarrow \text{Dieb} ) = 1/2 & P( \text{NX} \rightarrow \text{NX\#1 PX} ) = \frac{l}{3h+3l} \\
P( \text{APPR} \rightarrow \text{mit} ) = 1 & \\
P( \text{ART} \rightarrow \text{dem} ) = 1/2 & \\
P( \text{NN} \rightarrow \text{Fahrrad} ) = 1/2 & 
\end{array}$$



The relation between the probabilities of the two parses then is:

$$\begin{aligned}
& P(\phi^a|w^a) > P(\phi^b|w^a) \\
\Leftrightarrow & P(\text{MF} \rightarrow \text{NX PX}) \\
& P(\text{NX} \rightarrow \text{ART NN}) \\
& P(\text{NX} \rightarrow \text{ART NN}) > P(\text{MF} \rightarrow \text{NX}) \\
& P(\text{NX} \rightarrow \text{NX\#1 PX}) \\
& P(\text{NX} \rightarrow \text{ART NN}) \\
\Leftrightarrow & \frac{h}{h+l} \frac{2h+l}{3h+3l} \frac{2h+l}{3h+3l} > \frac{l}{h+l} \frac{l}{3h+3l} \frac{2h+l}{3h+3l} \\
& \Leftrightarrow h(2h+l) > l^2 \\
\Leftrightarrow & h^2 + \frac{1}{2}hl - \frac{1}{2}l^2 > 0 \\
& \Leftrightarrow h > \left(\sqrt{\frac{5}{8}} - \frac{1}{4}\right)l \\
& \Leftrightarrow h > 0.54l \tag{5.5}
\end{aligned}$$

Assuming a standard PCFG model, it thus turns out to be possible to encode just the appropriate additional information in node labels that lets the resulting model assign different interpretations to unseen data just as often as these analyses were seen in the data on which it is based (here: low vs. high attachment). It is not immediately obvious, however, which node labels to change in order to find the best compromise between the number of newly introduced node labels and the faithfulness to structural preferences observed in the treebank.

### 5.1.3 Structural and Lexical Preferences

While the discussion so far has focused on structural preferences, it would be desirable to choose high or low attachment depending on the lexical form of the verb as well. In a PCFG framework this implies that the left-hand-side of the rule that is central for the attachment decision has to include information on the verb. In our example in figure 5.1, the full sequence of nodes (VVF<sub>FIN</sub>, VX<sub>FIN</sub>, LK, SIM<sub>PX</sub>, MF) separates the relevant left-hand sides from the verb. The appropriate parse will be more likely only when the lexical information of the verb is present on both ends of the sequence at the same time. In the end, the following should always hold without depending just on the structural preference characterised by  $\frac{h}{l}$ :

$$P(\phi^a|w^a) > P(\phi^b|w^a)$$

and

$$P(\phi^b|w^b) > P(\phi^a|w^b)$$

The number of nodes involved in the connection between a lexical item and the corresponding production thus seems to be higher than for the local structural preferences considered before, where changing just one node was sufficient.

This example shows the differences and the similarities between lexical and structural preferences from a probabilistic context-free point of view. Both need similar changes to the grammar to be reproduced faithfully by it, i.e. changes to node labels so that the contextual differences are within reach of PCFG conditioning. The difference lies mainly in the distance that needs to be bridged in the syntactic description, measured by the minimal number of nodes that need to be modified to represent the difference. Structural preferences for us will consequently be all preferences that can be captured in only few dedicated node labels, as opposed to lexical preferences, which typically link a node at the fringe of the syntactic description with another node higher up the tree, or even down the fringe again, crossing part of the tree. We know that this distinction is far from clear-cut, but as there is no fundamental difference between both kinds of preferences from our perspective, we use structural preferences just to start exploring the problem of encoding preferences in node labels in general. There are also cases where very frequent lexical items have strong impact on syntactic structure that is very near (e.g. commas starting relative clauses), which also surface as preferences in our experiments. We thus exclude the problem of lexical preferences for most of our experiments only as a simplifying restriction, and use the degree in which they can be reflected in the annotation as a measure for the sensitivity of our method.

Manning and Schütze (1999) discuss three different independence assumptions on which we rely when we calculate the probability of a parse as the product of the context-free rules in it (p. 384). Recall that we calculate the probability of a parse  $\psi$  simply as  $P(\psi) = \prod_{r \in \mathcal{R}} P(r)^{Count_{\psi}(r)}$  (equation 2.1 on p. 10), where the distributions of expansion probabilities of nonterminals are assumed to be

**place invariant** – a subtree has always the same probability independent of its position in the string,

**context-free** – the probability of a subtree does not depend on words that it does not dominate, and

**ancestor-free** – the probability of a subtree does not depend on any dominating non-terminals outside the subtree.

These assumptions obviously also underly the modelling of our examples on the previous pages. In the original annotation in figure 5.1, the probability that an `NX` expands into a pronoun, or a base, or a complex noun phrase, does not depend on whether it appears in the initial field or in the middle field (place invariance), and it does not depend on any terminal or non-terminal node outside the subtree that it dominates (context- and ancestor-freeness). The same assumptions still hold for all nodes in figure 5.3(a), of course, and in particular for the new node labelled `NX#1`. The difference between `NX` in figure 5.1 and `NX#1` in figure 5.3(a) is that `NX#1` only appears in the middle field, so that there is no other place where place invariance could apply, and similarly that the expansion of `NX#1` is ancestor-free, but it never appears but below `MF`, so that in effect, the probability of `NX#1` expanding into anything when it does not have the ancestor `MF` is zero.

We have seen by example that due to these independence assumptions inherent in PCFGs, the representation of linguistic information has direct impact on the grammar’s ability to select the correct parse, and context-freeness, ancestor-freeness and place invariance can have different impact. PCFG rule probabilities that are estimated via likelihoods being read directly off a treebank do not necessarily prefer full parses that are more frequent in the treebank. In the first TüBa-D/Z representation of prepositional phrase attachment, e.g., the low attachment has to be more than four times as frequent in the treebank than the high attachment to be chosen at all. Small modifications in the representation are sufficient to achieve better estimation, because e.g. information about an ancestor that would otherwise be outside of reach of PCFG conditioning according to the ancestor freeness assumption is moved into reach of conditioning via labels that carry the relevant information. However, not all modifications lead to optimal results, and the modifications are dependent on the treebank and on the amount of information included in the PCFG representation from the original parses. More information generally leads to more complex models, but may also represent syntactic preferences more adequately. Even though the original encoding of syntactic information seems to suggest a certain context-free representation (e.g. either with all or without any edge labels), none of these obvious choices may be optimal for our goal of capturing linguistically relevant observations in a PCFG. What surfaces here is the effect of different representations of treebank annotation. The default representation may be perfectly reasonable for those who produce and use the treebank, because it is a well-established solution to label noun phrases of different complexity with the same label (an occasional bar added to the label just highlighting their status). On the other hand, a PCFG may be perfectly able to learn the prevalence of a complex configuration from a corpus when certain labels are modified, as we

have shown above. Both representations do not have to coincide, and it is actually quite unlikely that they do, given the different capabilities and assumptions of human beings and PCFGs about the semantics of node labels. Both assume a semantics of labels and structure, but PCFGs do so just on the basis of treebank evidence and focusing on very local context for any particular label, allowing links that connect nodes to their grandparents or to surrounding words only via a naming scheme that looks peculiar to the human eye.

Capturing the semantics of treebank annotation, which is originally best suited for the linguistically informed human corpus annotator, in an annotation scheme that allows the rather limited and different point of view of a PCFG to take advantage of the original semantics, is at the heart of our undertaking. A more general automatic means thus seems to be desirable to transform an annotation scheme that is optimal for human corpus developers and users, into a scheme that grants context-free methods access to the knowledge encoded in the treebank. The description of a novel method with this goal will be the topic of chapters 7 and 8, and chapter 6 surveys existing approaches to this problem. Before we turn to these methods, however, we will describe how we turn the annotation of TüBa-D/Z into proper trees.

## 5.2 PCFG-Parsing of TüBa-D/Z

The Penn Treebank was the source of early PCFGs derived from a treebank, because its annotation was rather easy to map to proper trees. But even for the Penn Treebank there are cases where it is not obvious what the best context-free representation of a treebank's syntactic annotation is. The Penn Treebank, e.g., adopts the notion of null elements that mark the position of elements in the syntactic description that are either not realised, or where the realised position in a sentence does not coincide with the position where it is assumed to appear in the syntactic description. Empty elements that are not realised are introduced for, e.g., the subjects of infinitive clauses or objects in passive voice. The resulting empty ( $\epsilon$ -)productions can still be modeled by our context-free grammars. Keeping them, however, adds considerable complexity to parsing, because non-terminals that are not anchored in any visible word can appear in many places. Traces also appear in the Penn Treebank as empty elements on the level of words. They are realised at a different position than where they are assumed to belong in the syntactic description. Making the connection explicit via edges instead of via co-indexation can therefore force edges to cross, because traces effectively allow connecting arbitrary non-terminals and terminals. Figure 5.4 shows a sentence where a

clausal subject (S-1) is extraposed from sentence-initial position, leaving a

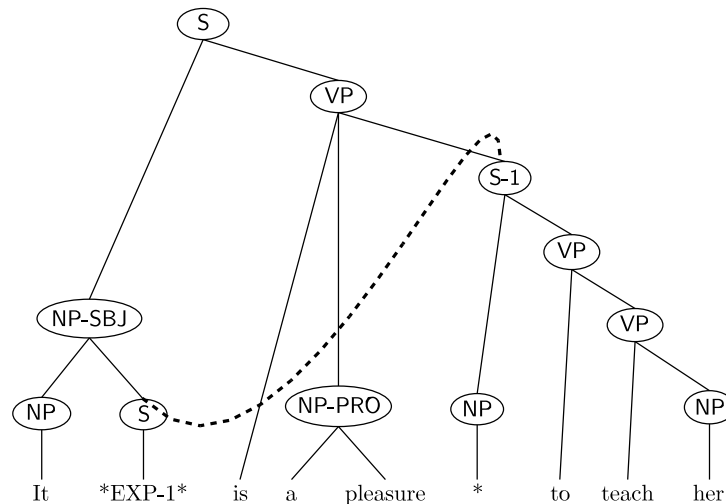


Figure 5.4: Labels encode edges

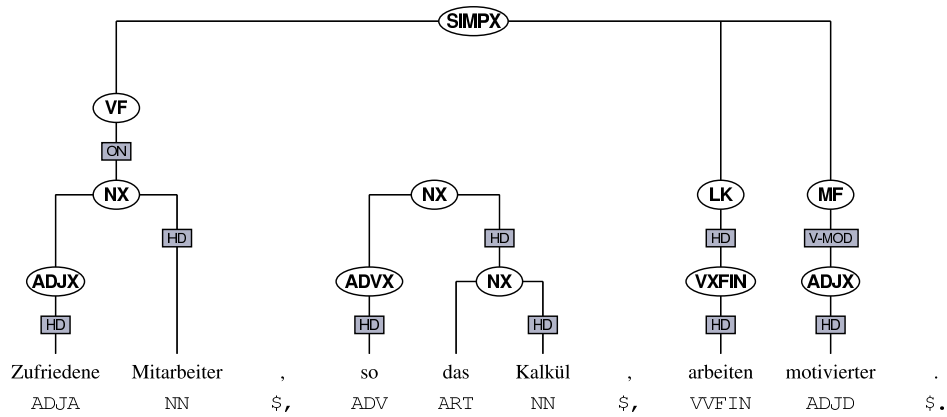
trace marked as *\*EXP-1\** (Bies & al., 1995, p. 271). The dotted line, which is not present in the original annotation, could be used as an alternative to the coindexed S-1 and *\*EXP-1\**. Using co-indexed traces, however, avoids crossing edges and is thus nearer context-free power. Plaehn (1999) shows that an extended algorithm that probabilistically models grammars similar to PCFGs, but allows crossing edges, is a much more complex task, resulting in exponential complexity.

The alternative empty elements like *\*EXP-1\** and *\** in Figure 5.4, however, also add complexity to parsing. Few treebanks are therefore used for obtaining PCFGs without prior transformations that give them the shape of proper trees without  $\epsilon$ -productions or crossing edges. Charniak (1996), e.g., reports that he simply ignores traces. Even though the Penn Treebank consists of simple labelled bracketings inserted into the text, which look remarkably similar to proper trees that can be directly converted into a PCFG, it thus turns out that “any grammar extracted from this bracketed corpus requires the adoption of a number of assumptions whose alteration would lead to different results” (Gaizauskas, 1995). This problem holds for other treebanks as well, of course. Each attempt to extract a PCFG from a treebank is faced with similar problems, which may not be too important, because the PCFG may only be needed to filter out candidate parses for more powerful methods (Frank & al., 2003). Other attempts use the derived PCFG treebank grammar directly for parsing, so that the effect of transformations is of direct importance (e.g. Dubey and Keller, 2003; Schiehlen, 2004).

In summary, transformations of syntactic annotation encoded in treebanks are necessary whenever the annotation follows a model more powerful than CFGs, but they are also often applied when the annotation is already quite similar to proper trees. We have seen before that differences in representation severely influence the decisions taken by the resulting PCFG models. TüBa-D/Z, of course, is no exception here. In this section we discuss representational issues that have to be solved to arrive at a context-free representation of TüBa-D/Z, and the solutions that we adopt. The major problems are those areas discussed in section 4.1 where proper trees are either suboptimal or genuinely insufficient for representing the assumed syntactic structure provided by the annotation. We also describe other relevant pre-processing, including POS-tagging, and our division of data into train and test sets.

### 5.2.1 Parentheses and Punctuation

The start symbol is not visible in the default representation of the *export* data model, because it gives this start symbol (VROOT) a special status. VROOT is the root symbol of each sentence’s annotation graph in TüBa-D/Z, and any terminal or non-terminal node in a sentence has it as a parent if this node is not explicitly attached to another node via a (visible) edge (Brants, 1997). We call all nodes *attached* that are attached explicitly to a node other than VROOT; all remaining nodes are *unattached*. Nodes with an unattached status do not exist in a CFG, so that a mapping has to be defined to capture this information in a CFG. In the simplest case, VROOT just becomes the explicit new mother of the formerly unattached non-terminal or terminal nodes as proposed above, marked by a distinct edge label. This solution is sufficient for the SIMPX node and the final full stop in figure 5.5. The other unattached nodes, i.e. the two commas and the NX nonterminal, cannot be mapped analogously, because they are surrounded by words that have a common mother node that is not linked to them via an explicit edge. Figure 5.6 shows the same sentence as figure 5.5, but this time the implicit edges from all unattached nodes to VROOT are shown as light grey lines, and it is easy to see that some of them cross explicit edges. The unattached constituent labelled NX forms what we will call a *parenthesis*, i.e. an unattached non-terminal node without an explicit mother surrounded by words that have a common mother, and normally separated from these words by commas. The lowest common mother of the preceding and following words of the parenthesis (SIMPX in figure 5.5/5.6) is the node to which it can be easily attached, resolving one of the crossing edges. This is the approach pursued for attaching parentheses in all our experiments. We always append the pipe symbol (|) to the existing label of parentheses instead of using an edge label, so



Zufriedene Mitarbeiter, so das Kalkül, arbeiten motivierter.  
 Happy staff, so the idea, work more\_motivated.  
 ‘Happy staff, that is the idea, work with more motivation.’

Figure 5.5: Unattached Elements: Parenthesis and Punctuation

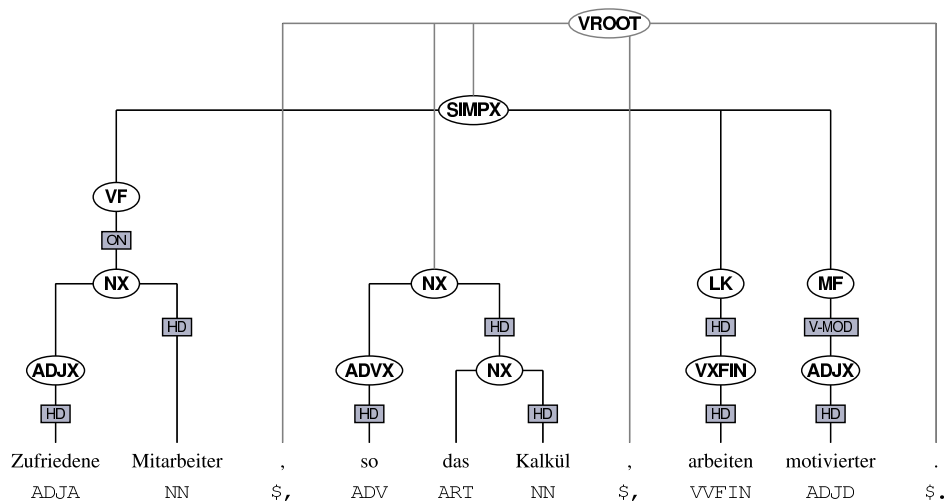


Figure 5.6: Unattached Elements with VROOT shown

that information about their status is also evaluated in experiments where edge labels are ignored. Parentheses are rather infrequent (3 in the first 100 sentences of TüBa-D/Z) and there is usually a node that dominates all and only the words that make up the parenthesis, which makes them rather easy to handle.

Punctuation is never attached in TüBa-D/Z and it usually occurs between constituents, which makes the decision where to attach the punctuation harder.<sup>1</sup> Punctuation marks are distinct from parentheses in that they are more directly linked to the rest of syntactic structure. They are part of a closed class of lexical items, and occur frequently. They rarely occur in sequences, and they usually separate constituents or clauses. They therefore provide interesting additional information for parsing, which is directly available in annotation schemes that explicitly map them to certain positions in clauses, or constituents (as in the BulTreeBank or the Prague Dependency Treebank, PDT, Böhmová & al., 2003; Osenova and Simov, 2004). While still attached near the constituents that they belong to, they are “at the very bottom of the pecking order” for other schemes (Bies & al., 1995, p. 54, about quotation marks in the Penn Treebank). Still other schemes leave them completely unattached to the syntactic annotation. This is true for the TüBa-D/Z, on which rests the main focus here, but also for the negra and Tiger treebanks (Brants and Hansen, 2002; Skut & al., 1998; Telljohann & al., 2003).

The potential usefulness of punctuation for parsing when appropriately anchored in syntax has been pointed out before (Doran, 2000; Jones, 1994). For us, punctuation seems to be especially valuable because on top of being connected to syntax, punctuation is far less ambiguous than other lexical items of input sentences, which is underlined by the few tagging errors observed for the POS tags \$., \$, (both no errors), and \$( (0.22 % errors).<sup>2</sup>

While punctuation is present in TüBa-D/Z as part of input data, but not explicitly linked to syntactic analyses, we will now try to answer the question at which position in syntactic annotation it seems to give maximum benefit for parsing. The difference between the equally unattached parentheses and punctuation is, that the former are usually enclosed by words of a single constituent, while the latter tend to occur at the end of, or in between constituents (which is partly what makes them valuable for annotation). Attachment, thus, can still be represented by an edge carrying a new label, but finding the place of attachment is harder for punctuation.

---

<sup>1</sup>We do not consider those punctuation marks here that replace e.g. prepositions, and which are tagged as the word they replace in TüBa-D/Z (Telljohann & al., 2003, p. 22).

<sup>2</sup>All errors are dashes that are wrongly tagged as punctuation despite replacing the preposition *gegen*. The error rates correspond to the upper right cell of table 5.6.



There are at least three obvious ways in which punctuation can be handled. First, it could be ignored, leading to a grammar that only considers linguistically motivated constituents along the lines of the original annotation. Second, punctuation marks and brackets may be attached at a position that is least decisive about their attachment site, which is arguably their highest possible attachment in a tree without crossing any edges. Last, they may be attached to a syntactic unit to which they belong most, i.e. lower in the tree. These three options will have an impact on the resulting treebank grammars. Table 5.1 summarises the expected impact on the grammar.

	#rules	generalises	information
ignore	few	better	losing
highest attach	many	worse	preserving
lower attach	medium	better	preserving & relocating

Table 5.1: Expected Impact of Punctuation on Grammar

The simplest strategy of removing all punctuation requires the insertion of removed punctuation after parsing, so that parses can be compared with the same test data as all other experiments. The second strategy of attaching unattached elements as high as possible in the syntactic tree simply determines the lowest common parent of the left and right sisters of the unattached punctuation marks, to which they are then attached. If there are several unattached elements next to each other, then all of them are attached to the lowest parent of the left and right sisters of the full sequence. If there is no sister to any one side (i.e. the unattached element forms the beginning or end of the sentence), then the element is attached to the start symbol of the sentence (i.e. to VROOT). While attachment sites thus can be determined unambiguously for highest attachment, and the strategy of removing all unattached material is also quite clear, the third option needs additional clarification. In the remainder of this section, we propose a heuristics that defines attachment sites for a subset of unattached punctuation. Experiments will show the impact on performance of all three modes of handling unattached elements, motivating our choice of the more informed low attachment strategy.

The third strategy determines certain relative attachment points depending on the type of punctuation. Depending on the type, pairs of elements are jointly reattached (quotation marks  $\wedge["'"]\$/^3$ ), the elements are reattached individually (e.g. comma or full stop  $\wedge[, .]\$/$ ), or kept in their default highest attachment position (e.g. dash or asterisk  $\wedge[-*]\$/$ ). The attachment is

<sup>3</sup>We adopt the Perl regular expression syntax (Wall & al., 2000) to match words.

left unchanged mainly for infrequent elements, where information about possible usages was felt to be too sparse to devise a general strategy (e.g. for the equation sign  $\hat{=}$ ), or where the original position normally seems to be appropriate (as for the ellipsis  $\hat{\backslash}\backslash\backslash$ ).

We will now informally describe the reattachment algorithm. A loop encloses the following choice of reattachments. Whenever something is reattached, the loop restarts. All unattached elements are expected to be attached in highest position before the loop starts, i.e. in their position according to the second strategy. The punctuation marks  $\hat{["'-'*\backslash[\backslash]=]|\backslash\backslash\backslash}$  and all others mentioned in the second step below are left as is and considered as being not existant in the following first step. The following loop is repeated per sentence until no more words are reattached:

$\hat{(\$}$  Attach to following lowest non-terminal node.

$\hat{)}\$$  Attach to preceding lowest non-terminal node.

A simple heuristics that handles the frequent case of orphaned non-terminals or appositions enclosed in brackets acceptably well (see figure 5.7; arrows mark reattachment, and dashed lines previous highest attachment of formerly crossing edges).

$\hat{[,;]\$$  Attach in original position, when preceding or following sister has KONJ edge.

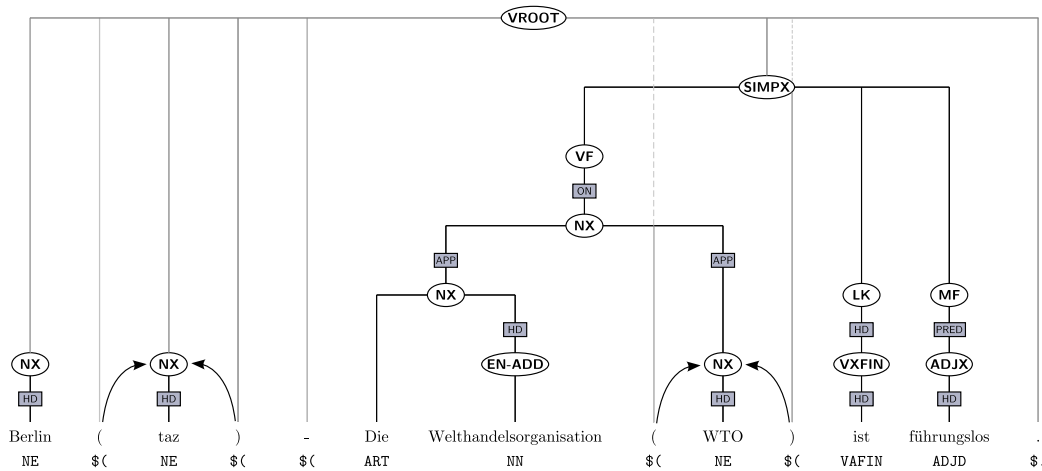
Else attach to the highest clausal descendant of the following, or, if there is none, of the preceding node (we start from highest attachment). If there is none, continue looking in the same way for a following or preceding node with an FKOORD label, one which is unattached, or has an APP edge, and attach it to the first found. We thus prefer to attach commas and semicolons to following clausal nodes, as in the prototypical case of the relative clause in the final field.

Else leave in original position.

$\hat{[:.!?/]\$$  Attach to preceding highest node. Normally this is the matrix clause.

When this loop has finished, punctuation marks that are assumed to occur in balanced pairs but whose counterparts need to be determined first are handled as follows in a second step:

$\hat{["']\$$  Wherever there appears an even number in one sentence, attach to the lowestmost common parent node of the



Berlin ( taz ) - Die Welthandelsorganisation ( WTO ) ist führungslos.  
 Berlin ( taz ) - The World\_Trade\_Organisation ( WTO ) is leader-less.  
 ‘Berlin ( taz ) - The World Trade Organisation ( WTO ) is without a leader.’

Figure 5.7: Reattach Brackets to Lowest Following/Preceding Nonterminal

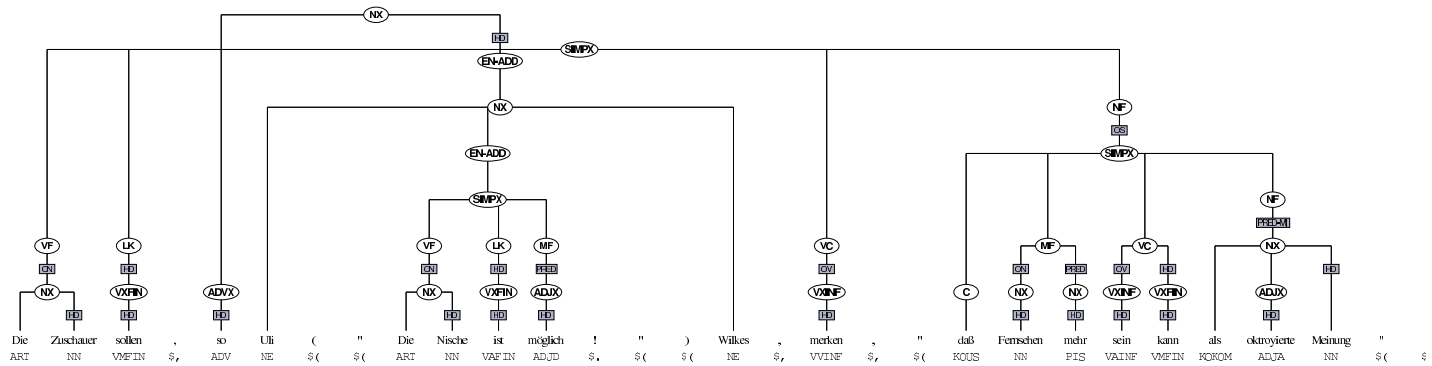
**following** word for a quotation mark with an even number, and of the **preceding** word for a quotation mark with an uneven number.<sup>4</sup>

There are several problems with these heuristics, and we do not claim that they work perfectly in all cases. We do hope, however, that they assign punctuation marks to more appropriate places most of the time. Figure 5.8 shows a sentence with a high number of punctuation marks (11 of the 30 words) as found in TüBa-D/Z (figure 5.8(a)), where punctuation is attached when following the heuristics just given (figure 5.8(c)), and when simply attached as high as possible (figure 5.8(b)). The annotation as shown in figure 5.8(a) cannot be used as is by a PCFG, because there are unattached elements and more than a single start symbol. The thick edges in figures 5.8(b) and 5.8(c) mark edges modified during reattachment.

Figure 5.8(c) shows some success, e.g. for reattaching all commas to appropriate positions, which enclose the parenthesis *so Uli ("Die Nische ist möglich!") Wilkes*, and introduce the subclause *"daß Fernsehen mehr sein kann als oktroyierte Meinung"*, which, moreover, also includes quotation marks at positions that embrace direct speech. The full stop at the end of the sentence has also been assigned an appropriate position after the final field (NF) as last part of the clause (SIMPX). The prescription for brackets,

<sup>4</sup>We start numbering with zero.

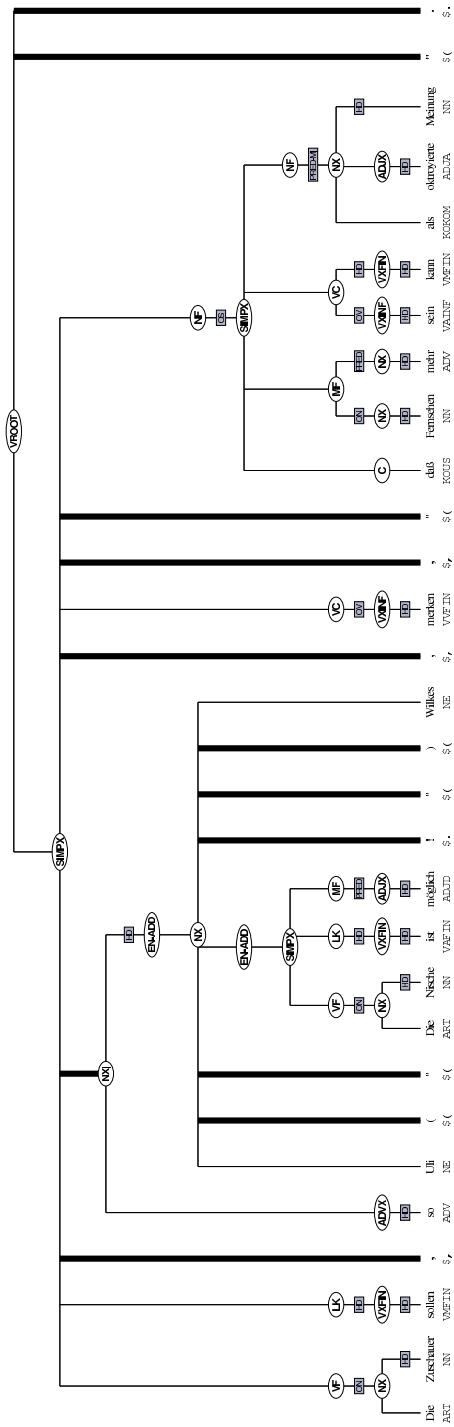
(a) Original Treebank Annotation



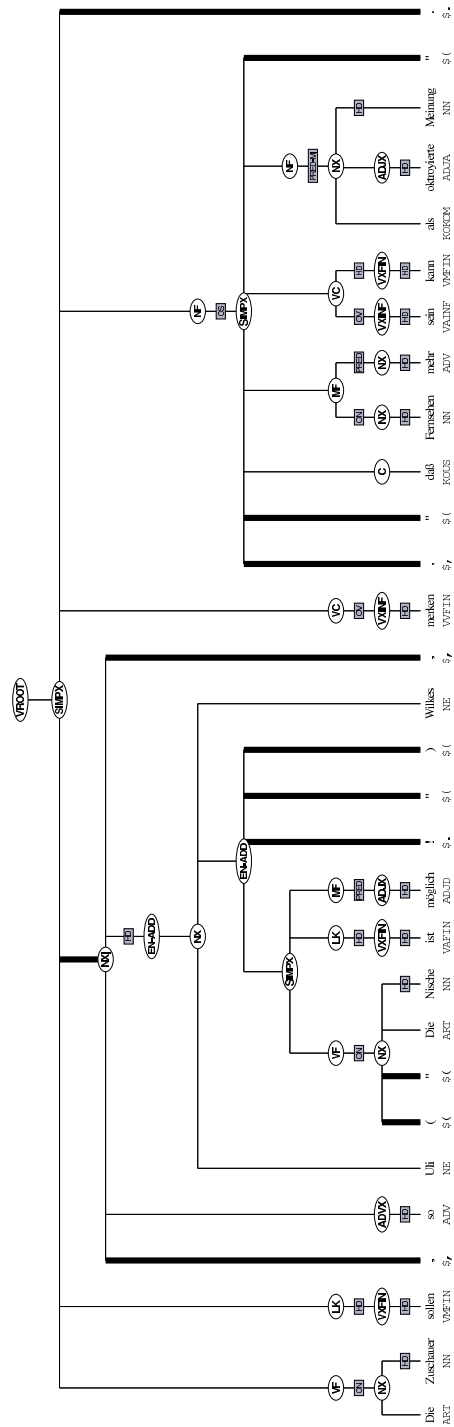
Die Zuschauer sollen, so Uli ( "Die Nische ist möglich ! " ) Wilkes, merken, " daß  
 The audience should, so Uli ("the niche is possible!") Wilkes, realise, "that  
 Fernsehen mehr sein kann als oktroyierte Meinung".  
 television more be can than octroyed opinion".

‘The audience should realise, according to Uli ("the niche is there!") Wilkes, "that television can be more than octroyed opinion".’

Figure 5.8: Punctuation and Parentheses, Unattached



(b) Highest Attachment



(c) Low Attachment Heuristics

Figure 5.8: Punctuation and Parentheses, Reattached

however, prevents a perfect attachment of quotation marks inside the parenthesis. Quotation marks are handled after all other punctuation. Brackets are attached first to the lowestmost preceding/following node (being sure shots), blocking a higher attachment of the double quotes at the start of the parenthesis. The heuristic is not intelligent enough to handle the sequence of balanced punctuation of different types enclosing the parenthesis: first, the left bracket is attached lowestmost (the quotation mark being ignored) to the following noun phrase (NX). The right bracket is assigned a new position next, but cannot be reattached, because the exclamation mark blocks its way: after the exclamation mark has been attached, the EN-ADD has become the lowestmost possible attachment position of the rightmost bracket. When the quotation marks are considered after no more punctuation marks have been attached in the first stage, their linear position in the sentence does not allow any other attachment than that shown in figure 5.8(c).

This is the first of a series of changes that augments treebank data, but preserves the information available in the original data. When the added information about attachment of punctuation is removed, the original data resurfaces. In the next chapter, we will introduce a method that explores similarly reversible changes in treebank data. Then, and now, we will use evaluation via the metrics and the PCFG model previously introduced, to assess the benefit of the changes for this kind of parsing. Evaluation is always performed on the original representation, i.e. it does not include a direct estimate on the quality of transformations, but only via the effect of transformations on improved parsing. Accordingly, table 5.2 shows the performance of three PCFG models derived from the treebanks without changes, and with punctuation attached high and low.

Punctuation	ignored	attached high	attached low
Parseval $F_{lab}$	80.03	83.15	83.48
Parseval $F_{unl}$	83.72	86.72	87.15
unparsed	0	1	0
grammar rules	2134	3132	3432
complete match	20.57 %	22.24 %	21.55 %
average sent $CB$	3.23	2.39	2.18
0 $CB$	49.69 %	54.56 %	55.43 %
$\leq 2$ $CB$	68.99 %	74.34 %	76.07 %
$perp$	14.51	12.45	13.49

Table 5.2: Impact of Punctuation on Plain PCFG Parsing Performance

Despite the partial failure of both reattachment heuristics, table 5.2 shows

that using and reattaching punctuation considerably boosts performance. Lower attachment increases the weighted average of precision and recall both labelled and unlabelled, does not lead to unparsed sentences, and has the lowest crossing brackets (*CB*) figures. As a downside, low attachment increases the number of rules, because the types of rules applied high in a tree becomes smaller. The lower reduction of cross-perplexity is probably also related to the higher number of rules, while cross-perplexity is still lower than when punctuation is ignored. The single rule  $\text{VROOT} \rightarrow \text{SIMPX } \$.$ , e.g., which describes the top of most main clauses with high attachment of punctuation, is turned into many rules with low attachment of punctuation, where the full stop is the final child of the main clause (*SIMPX*), and is not outside of it. Table 5.3 shows rules from low attached training data with *SIMPX* as their left-hand side which occur more than 100 times, plus corresponding rules from high attached training data. Having those rules once with, and once

attachment strategy	frequency of occurrence			
	low		highest	
	yes	no	yes	no
with trailing \$.				
$\text{VROOT} \rightarrow \text{SIMPX } (\$.)$	0	5801	4871	278
$\text{SIMPX} \rightarrow \text{VF LK MF } (\$.)$	1715	627	0	2020
$\text{SIMPX} \rightarrow \text{VF LK MF VC } (\$.)$	1397	364	0	1615
$\text{SIMPX} \rightarrow \text{VF LK MF NF } (\$.)$	694	68	0	30
$\text{SIMPX} \rightarrow \$, \text{ C MF VC } (\$.)$	0	762	0	0
$\text{SIMPX} \rightarrow \text{C MF VC } (\$.)$	24	82	0	1140
$\text{SIMPX} \rightarrow \text{VF LK MF } \$, \text{ NF } (\$.)$	33	4	0	695
overall types lhs = <i>SIMPX</i>	223	340	0	411
overall tokens lhs = <i>SIMPX</i>	6018	4345	0	10 363

Table 5.3: Main Clause Rule Frequencies vs. Attachment of Punctuation

without punctuation, of course yields more rule types overall than factoring out the trailing \$. into  $\text{VROOT} \rightarrow \text{SIMPX } \$.$  most of the time. The advantage of ending a sequence of fields with a full stop rather than disconnecting the full stop from that sequence is that you can assign distinct probabilities for matrix clause field sequences, as opposed to e.g. coordinated field sequences. In addition to the slightly better *F* performance, we think that this is a good reason for using low attachment of punctuation as the basis for all our remaining experiments despite the increase in the number of rules. The overall number of rule types does not increase as dramatically to twice the original number as suggested by table 5.3 (where it is 12 low vs. 7 high, from cells with zero frequencies). Instead, it grows to 563 from 411 types

for the highly affected rules with `SIMPX` as left-hand side, and to only 3432 from 3132 rule types overall (see table 5.2). This moderate increase is accompanied by better generalisation, exemplified by the 223 new low attachment rules covering more of the treebank than the 340 unchanged rules (second last row of columns two and three in table 5.3).

## 5.2.2 Secondary Edges and Co-Indexation

The information encoded in secondary edges is rather restricted and covers relations that cannot be easily expressed by proper trees, even when co-indexation is used to resolve the major part of crossing edges. Information encoded via secondary edges includes the structure of proper names when it deviates from constituent analyses, the internal dependency structure of large verbal complexes (which usually follows a default rule), and relations that do not hold between direct children of field nodes, i.e. either clause-external or phrase-internal relations (see section 4.3.2). Secondary edges are ignored in all our experiments. As a consequence, all information encoded in secondary edges is neither annotated nor part of the evaluation in our experiments.

Co-indexation is mostly used to cope with free constituent order in German. It can relate complements or adjuncts to their modifiers with a rather small set of labels (with edge labels ending in `-MOD`), or to their other parts in split-up coordinations (with edge labels ending in `K`; see table A.5 in the appendix). Co-indexation via edge labels is part of all experiments that include edge labels, and otherwise not considered.

## 5.2.3 Train and Test Regimes

All experiments that we perform follow the standard setup where the data set is split into data for training and data for testing. A training set is necessary because the language model used for parsing is derived almost exclusively from treebank data. The language model used for POS tagging takes advantage of additional data which will be described below, and pre-processing as described earlier in this chapter uses hand-crafted heuristics, which are the only other sources of information on top of TüBa-D/Z that determine parsing results.

The training and testing regime should guarantee a faithful assessment of parser performance, and will be more convincing with more test data. Setting aside more test data conflicts with the goal of maximising the amount of training data, however, which is desirable because treebank data is the most important source of information for building the language model. Cross-validation is a solution for maximising training data on the one hand, and



reducing the risk of biased test data through the choice of too small, and therefore unrepresentative samples. Cross-validation splits the data into  $n$  subsets of similar size and performs  $n$  experiments that use each of the  $n$  parts as test data once, and the remaining  $n - 1$  parts as training data. Results of all trials are averaged, where standard deviation shows if performance is uniform over all data. Unfortunately, such a setup is prohibitively expensive in our case, because repeating our search for more adequate representations of treebank data  $n$  times, and, equally importantly, parsing all sentences of the treebank with a PCFG parser, costs several days instead of less than a day.<sup>5</sup> We have instead settled on a single split of treebank data following the division of the newspaper data into days. We perform our experiments on the first release of TüBa-D/Z, which consists of the five days of newspaper data from May 3–7, 1999. Of these days, the last two have been annotated first and have been revised most intensively afterwards. We have therefore used them as test data since the beginning of our research. May 5 has been added last and only partly, because at the time of publication of TüBa-D/Z, only part of it had been fully revised. We have opted against using every  $n$ th sentence as training data, because we think that a realistic scenario of parsing unseen data will more likely be based on training data from the same source (here: newspaper) than on training data from the same text (here: newspaper article).

We determine certain parameters during the course of experiments to optimise performance, so that performance is geared towards the test data we use in these *tune* experiments, and we cannot guarantee that we would have arrived at the same settings for a different set of test data. In order to assess the generality of optimisations, we tune the model on a tune test set which is distinct from the final test set for the final evaluation of the model. Final evaluation is only performed once for parameter settings selected via experiments on tune data. The division of the whole of TüBa-D/Z into a tune and final data set, and of both sets into data used for training and testing is shown in table 5.4. Tune test is included in final train data, so that not the same model is evaluated on the final data set, which can be seen as a disadvantage, but the amount of training data is increased by more than 40 % for the final evaluation. The table shows that a rather high fraction of data is set aside for testing. The split of the treebank into tune/final train/test sections has been determined rather early during treebank annotation and was meant to guarantee that enough data is available for testing. Although the amount of data consequently available for constructing a model is rather limited as a downside, we think that the advantages make up for this. First,

---

<sup>5</sup>This situation has only changed with the advent of `BitPar` at the end of our research.

May,	3.	4.	5.	6.	7.
------	----	----	----	----	----

sentences with up to 40 words

words	48 607	50 289	23 765	50 327	65 868
sentences	3077	3090	1457	3067	4009
nodes	58 505	60 711	28 503	60 362	78 721

tune data set

	train	test
words	122 661	50 327
sentences	7624	3067
nodes	147 719	60 362

final data set

	train	test
words	172 988	65 868
sentences	10 691	4009
nodes	208 081	78 721

unrestricted length

words	53 217	56 492	26 384	56 422	73 926
sentences	3170	3215	1511	3190	4174
nodes	63 858	67 850	31 506	67 480	87 902

unrestricted final data set

	train	test
words	192 515	73 926
sentences	11 086	4174
nodes	230 694	87 902

Table 5.4: Division of TüBa-D/Z into tune and final data sets

larger amounts of test data yield more reliable evaluation, especially when cross-validation is not an option. Second, we hope to establish a division into train and test data that is easy to follow by others, so that results stay comparable (e.g. Müller, to appear; Veenstra & al., 2002, already follow the same setup). We especially hope that with a new release of TüBa-D/Z, the division can be kept.

If not specified otherwise, all experiments are performed on sentences with up to 40 words from the tune data set, and all frequency information is also drawn from this set by default.

### 5.2.4 POS-Tagging and Parsing

Using a language model that is derived from training data that does not cover all of the expected test data always raises the question of how to handle input symbols that are unknown to the model, i.e. how we handle unknown words with a treebank grammar obtained via maximum-likelihood estimation. A standard method for handling unknown words is to define a frequency cut-off, under which all words in the training data are handled like unknown words in the test data, assuming that words that are unknown in test data behave like low-frequency words in training data. Unknown words in test data are consequently replaced by the same new symbol representing the unknown words in the training data. This approach targets open word classes like proper names, adjectives or full verbs which can be expected to behave similar to other members of their class.

In most of our experiments we have observed that first applying a POS tagger completely detached from parsing, as a preprocessing step, performs better than letting the parser assign POS tags as preterminals, as part of the parsing process. This is contrary to other experiments where POS tagging is rather a by-product of parsing (Beil & al., 2002; Hinrichs and Trushkina, 2003), or where POS tags for words seen in training data are assigned during parsing, and a POS tagger is only used as back-off (Collins, 1997) The differences to our results seem to be related to peculiarities of the treebank on which we perform our experiments (see section 8.4). As a consequence, most of our experiments, including those that perform best, take POS tags as input, which completely hide the word forms, so that the unknown words problem becomes less significant, because POS tags describe classes of word forms and are less numerous, and, most of all, form a closed class. Nonetheless, there are POS tags that do not appear in test data. The partitioning of the data results in two tags not seen in the tune training set, describing substituting possessive pronouns (PPOSS; e.g. *meines* ‘mine’) and imperative auxiliary verbs (VAIMP; e.g. *Sei* ‘Be’). The next least frequent tag occurs

once in training data (the participle perfect of modal verbs VMPP; e.g. *Er hat gewollt/VMPP*. ‘He has wanted.’).<sup>6</sup> All other POS tags occur twenty times or more. It is unlikely that following the standard approach outlined above and replacing VMPP in the training data, and VMPP plus VAIMP and PPOSS in test data, will generalise well for parsing sentences containing these unknown input symbols. Instead, we map the tags not seen in TüBa-D/Z training data to other tags for all input sentences, taking advantage of the hierarchical structure of the STTS tag set used in TüBa-D/Z (Schiller & al., 1995, and appendix A.1). All words marked as imperative auxiliaries are retagged as the more frequent class of imperative full verbs (VVIMP), and those tagged as substituting possessive pronouns are assigned the proper noun tag (NE). The former decision rests on the intuition that all imperative verbs behave similarly. The latter decision may be less intuitive, but genitive proper names are expected to show a behaviour that is distributionally similar to substituting possessive pronouns (noun phrases that can express possession consisting of a single word without an article). Both assumptions cannot be completely correct (otherwise there would not have been any motivation for the replaced tags in the first place) but are considered an acceptable solution for backing up from unknown input symbols for the task of handling unrestricted text.

Prior POS tagging is performed with the `tnt` tagger, which uses a second-order Markov model, and a suffix trie for guessing unknown words (Brants, 2000).<sup>7</sup> Table 5.5 shows results from experiments that were carried out to determine the usefulness of large but less closely related additional manually tagged texts that were available for our experiments. The table shows two additional resources called DEREKO and Tiger. DEREKO is a corpus of about 360 000 manually POS-tagged words that partly come from the same newspaper as does TüBa-D/Z (183 040 words from the *taz* of September 2–17, 1986; TüBa-D/Z has been originally bootstrapped from another part of DEREKO data); otherwise texts mostly come from novels (see Ule and Müller, 2004). The other resource is the Tiger treebank consisting of about 700 000 words of newspaper data that also include POS tags (although from the *Frankfurter Rundschau* instead of the *taz*, Brants & al., 2002). Both use the STTS tagset, with minor deviations from Schiller & al. (1995). TüBa-D/Z departs from the standard and changes the POS tag PAV to PROP, with identical behaviour. Tiger also changes this tag, but deviates also in other respects from the STTS. First, the tag PIDAT is dropped, and PIAT is used instead. Second, prepositions that modify numerals are tagged ADV. Third, a new tag NNE is

<sup>6</sup>Including additional external POS training data as outlined below changes these numbers to 5/2/3 for PPOS/VAIMP/VMPP, so that the problem stays the same.

<sup>7</sup>We use `tnt` version 2.2c with default settings.

test on full TüBa-D/Z, train on:

Tiger	x			x
DEREKO <i>taz</i> 1986		x		
DEREKO w/o TZ			x	x
accuracy %	94.8	95.1	95.3	95.5
unknown words %	11	16	15	9

test on TüBa-D/Z via 10-fold CV, train on:

rest of TüBa-D/Z	x	x	x	x
Tiger	x		x	x
DEREKO <i>taz</i> 1986			x	
DEREKO w/o TZ				x
accuracy %	95.7	96.0	96.3	96.4
unknown words %	9	12	9	9

Table 5.5: POS Tagging Performance Depending on Training Material

introduced to partly replace NN for compounds of proper names and common nouns. Last, the STTS tag PAV is renamed to PROAV in a similar way as in TüBa-D/Z (Smith, 2003). The last two changes can easily be undone, which is what we do (mapping the Tiger version to the TüBa-D/Z version). The first two changes cannot be undone without human intervention, so that we choose to keep them, resulting unfortunately in slightly different behaviour of the tags across our training data. Another normalisation we perform on the additional Tiger training data is to replace two-character double quotes /`‘|’\$ / with single-character double quotes /"/, which is the preferred representation in TüBa-D/Z.

Two sets of experiments have been performed to evaluate the usefulness of the additional data. First, all of TüBa-D/Z has been set aside as test data, and the tagger was trained on combinations of Tiger and parts of DEREKO material, which was divided into a larger and a smaller set. The larger set (DEREKO w/o TZ) only excludes those days of newspaper data that constitute TüBa-D/Z. The smaller set (DEREKO *taz* 1986) is a subset of the larger set restricted to only newspaper data. The upper part of table 5.5 shows that similar data (DEREKO *taz* 1986) seems to be more important than large amounts of slightly different data (Tiger), while a combination of both performs best. Having seen the importance of similar data, we show in the lower half of the table that best results are indeed achieved when the training data most closely resembles test data: training the tagger on nine tenths of the treebank (240 000 words) outperforms Tiger as training data, despite Tiger’s size. Best performance of slightly over 96 % is similar to what

Brants (2000) reports on *negra* (the predecessor of *Tiger*).

We conclude from these experiments that as much data as possible should be used from TüBa-D/Z, in combination with *Tiger* and all DEREKO material. This is the setup that we will follow in all our experiments involving POS tagging as an independent preprocessing step. As we do not use ten-fold cross-validation for our parsing experiments but a fix split between (tune or final) train and test data, we can only include data from TüBa-D/Z for tagging the test set from the respective training set, where we always include sentences of unrestricted length. When we use treebank data to produce a syntactic model, we try to build it from training data that is as similar as possible to the test data on which its performance will be evaluated. As a consequence, we train a first `tnt` POS model on additional data and tag TüBa-D/Z training data with that first model. We train a second POS model on the additional data plus the TüBa-D/Z train data, and tag TüBa-D/Z test data with that second model. The syntactic model that is derived from the automatically POS-tagged TüBa-D/Z training data will then find similar regularities in the (possibly erroneous) POS tags in the training data as in the POS tags of the test data, so that it can try to reproduce the correct syntactic annotation on similar sequences of POS tags in test data as the model has captured in train data. Table 5.6 shows performance on the previously established data sets accordingly, where the train part is tagged with a POS model derived from *Tiger* and DEREKO without TüBa-D/Z, and the test part with a model additionally based on the train part. These POS tags are the basis of all our parsing experiments that are based on POS tags alone. We would like to note that the final test set seems to be harder to tag than all other data. It seems to be even more dissimilar from the remaining treebank data than the external data is from the train sets.

	train	test
tune data set	95.9 %	96.0 %
final data set	95.8 %	95.6 %
unrestricted final data set	95.8 %	95.5 %

Table 5.6: POS Tagging Accuracy for Parsing

We think that this setup is a good compromise for the demands on training data. First, according to the graph produced by Brants (1999) for *negra* (Skut & al., 1998), more training data seems to increase performance mainly due to the reduction of the more difficult guesses on unknown words. Second, it is very important to build a language model from data that is very similar to the data on which the language model should be applied (Gildea, 2001).

Some experiments are also carried out on words as input for determining candidates for closed-class lexicalisation (see section 8.3), raising again the unknown word problem. The results have been produced by replacing infrequent unknown words (encountered three times or less) with a special symbol as outlined above. For these experiments, smoothing has not been switched off explicitly, so that results are obtained via linear interpolation as defined in the implementation of `lpar`.

Performing POS tagging as a step that comes prior to parsing and that is completely separate does not prevent handling of unknown words from taking place, of course. It just moves it completely out of the scope of parsing, and into the scope of a specialised preprocessor, which does a very good job at determining the morphosyntactic class of unknown words. We think that this decision is justified for two reasons. First, because it allows for easier interpretation of the interaction between the input of the parser and its output, which will become rather important when we try to interpret consequences of treebank transformations. Second, because the level of POS tags seem to be a useful layer of abstraction in TüBa-D/Z, judged by its effect on parsing performance (see section 8.4.5).

Edges that connect words with non-terminal nodes may carry HD labels. In order to preserve this information and not to extend the set of POS tags, we add non-terminal nodes above all words that have this label, and label these nodes with the POS tags and a trailing +HD. This transformation is performed in all our experiments, even when edge labels are not subject to evaluation, in order to keep the overall number of nodes constant, which is necessary to keep some of the evaluation metrics comparable. The additional nodes are always transformed back into word edge labels prior to evaluation.

## 5.3 Conclusion

PCFGs offer efficient means to determine most likely parses, which makes them attractive for assigning syntactic structure to natural language despite their limited ability to take context into account. Even though these limitations may prevent them from reaching state-of-the-art performance, their simplicity makes them interesting starting points for more powerful methods that may use them simply to filter out very unlikely parses, or to add exactly only those features that exceed their power.

Treebanks provide syntactic analyses, which are normally encoded in tree-like structures. These structures often closely resemble proper trees assigned to strings by PCFGs. This resemblance gives an easy way to derive context-free grammars along with rule probabilities, arriving at PCFG tree-

bank grammars. The ease of this approach is questioned by the fact that the choice of graphs to represent the analyses has an impact on how PCFGs capture the probabilities of larger configurations of nodes in treebanks. Consequently, syntactic analyses that are meant to express the same linguistic observation but that are encoded differently do not necessarily find their way into PCFG models in the same way.

This chapter has shown how syntactic annotation in treebanks can be changed manually in order to improve parsing with language models of restricted power, by modifying edges or node labels and by evaluating the impact of these changes. Our goal is to keep the amount of manually performed transformations minimal, making as much information as possible that is available in the treebank also available for PCFGs. In this chapter, we have concentrated on information that is initially outside the scope of context-free language models, like unattached punctuation, and we have shown how we move it inside the scope of the models. The following chapters will focus on alternative ways to represent the information already within reach of these restricted models, replacing manual optimisations we have employed so far with automatic transformations.



# Chapter 6

## Related Research

Differences in the representations of syntactic analyses have an impact on the way they are perceived by human users and by machines. The same analyses can be represented in different ways, so that they become more useful for a given purpose. There is a line of research that focuses on the impact of representational issues on the performance of Probabilistic Context-Free Grammars for mainly two reasons. First, Context-Free Grammars provide a rather simple framework, so that it is possible to understand the effect of changes in the representation on the behaviour of the grammar. Second, the quality of annotation produced by parsers that rely on the simplifying assumptions of PCFGs does not seem to have reached its peak yet, so that optimisations of annotation quality are accompanied by efficient execution. In this chapter, we focus on related research that either modifies the structure of PCFGs as a goal in itself, or as a means to improve parser performance. We also mention most closely related approaches to syntactic parsing of German.

### 6.1 Algorithmic Transformations

Some researchers try to overcome the limitations of PCFGs by systematically changing the structure of the grammar model without human intervention. They introduce information about parents into the node labels or perform an exhaustive search over all possible distinctions of node labels in a grammar that is generated from a treebank, or they define an objective function that guides changes in grammar structure in unsupervised training.

**Johnson (1998)**

Johnson (1998) examines the representations of syntactic analyses in the Penn Treebank and notes that the likelihood estimates of the syntactic analyses according to a plain PCFG treebank grammar differ from the corresponding frequencies obtained from training data. He motivates different changes in a treebank that are likely to produce more faithful estimates, including a reduction in the number of nodes, so that a PCFG's insensitivity to non-local dependencies loses impact, and also including an increase in the number of node labels, which incorporate more contextual information. He gives linguistically inspired variants of the original analyses according to these two strategies and compares the performance of the corresponding PCFG treebank grammars by means of a simple corpus containing two different forms of PP-attachment in what he calls Chomsky adjunct representation, in a flattened tree representation, and in the original Penn Treebank representation. He finally examines *parent annotation*, where the label of the mother node is appended to the label of each node in the treebank.

It turns out that for all different representations, the PCFG estimate differs from the observed frequencies for almost all ratios of high vs. low PP-attachment in his corpus. The same transformations are finally applied to the Penn Treebank and the resulting PCFG treebank grammar is evaluated against held-out test data. In this more realistic scenario the parent annotation outperforms all other transformations in terms of precision and recall. A more detailed analysis of the benefit of parent annotation shows that splitting the high-frequent noun phrase and sentential node labels accounts for a major part of the improvement in performance. Johnson points out that “what makes a tree representation a ‘good choice’ for PCFG modelling seems to be quite different to what makes it a good choice for a representation of a linguistic theory” (p. 630). Instead of “considerations of parsimony” that influence the choice of representations in linguistic theories, a PCFG is faced with a trade-off between bias and variance, i.e. when more detailed node labels reduce the bias that is inherent in the model, the variance increases at the same time, because less training data is available per type of rule.

**Belz (2002b)**

Belz (2002b) calls the information that could be relevant to estimate PCFG production probabilities more reliably the *local structural context*. She determines whether information about this context should be included into the node label by searching the space of grammars that is defined by a certain *maximally split Grammar* (Max Grammar) and all grammars that can be

derived from that grammar by merging node labels. Splitting in this context means using two or more distinct node labels instead of a single original node label, and merging accordingly means that a single node label replaces two or more previously distinct node labels in a grammar.

The search for a better grammar is guided by an objective function that maximises  $F_{lab}$  for parsing the Penn Treebank with the grammar. Starting from the Max Grammar and proceeding by merging allows for an efficient *Partition Search* instead of a standard cycle involving generating and evaluating a PCFG treebank grammar for each attempt to merge nodes, which would become prohibitively costly. A beam search is used that maintains  $n$  best partitions of the node set of the Max Grammar and that replaces  $b$  of them by randomly selected partitions that include more merges and that perform better according to the objective function. Grammar size is minimised at the same time, because increasing numbers of merges correspond to fewer rules. Beam search is employed instead of exhaustive search because the size of the search space grows exponentially with the number of different node labels  $i$  in the Max Grammar. Complexity of the given algorithm is only  $O(nbi)$ .

The best performing Max Grammar in the experiments reported by Belz (2002b) starts with node labels that are suffixed by the node label of their parent node in the treebank, where evaluation tests the performance of the annotation of noun phrases. Results on full parsing of the same data set show similar improvements (Belz, 2002a). It should be noted that only modifications of the grammar that are given a priori via the definition of the Max Grammar can be considered. While parent annotation proves to be useful, the benefit of other information such as more distant node labels or lexical information is not tested.

### **Bockhorst and Craven (2001)**

Bockhorst and Craven (2001) optimise a grammar that parses sequences of nucleotides in ribonucleic acid (RNA). Their goal is to recognise a class of RNA sequences that terminate an important part of the process of building a gene from a protein. These so-called *terminators* are described by a PCFG that has been designed from features proposed in prior research. Typically, terminators assume the shape of a loop, so that the bases at the end of the sequence map to complementary bases earlier in the sequence in reverse order. Generally, this kind of dependency can be straightforwardly modelled by PCFGs. The parameters of a corresponding PCFG are determined via unsupervised learning with the inside-outside algorithm (i.e. an expectation maximisation algorithm). Learning is carried out on a corpus of RNA se-

quences that are known or proposed to be terminators.

Bockhorst and Craven address a problem of this approach, which is the fact that when the initial structure of the grammar is deficient, then parametrisation cannot arrive at the best performance that is possible for more appropriate PCFG models. They define *Grammar Refinement*, which spots node labels in the grammar that perform the function of two or more node labels in a hypothesised optimal grammar. Grammar Refinement consists of a *Refinement Operator* and a *Refinement Heuristics*. The operator applies to the grammar and replaces a symbol on the right-hand side of a given production with a new symbol. At the same time, it duplicates all productions that have the replaced symbol on their left-hand sides, where the duplicates receive the new symbol as left-hand side. The heuristics determines to which node the operator should be applied. The choice is based on the parameter estimates of the PCFG model according to the inside-outside algorithm. The heuristics compares the distribution of productions of a node label with the distribution of its productions when it is also part of the right-hand side of a specific rule. A  $\chi^2$  statistical test is used to compare these distributions as well as the Kullback-Leibler divergence.

Experiments are performed on a corpus of 142 positive examples and 125 negative examples that all have a length of 50 symbols (i.e. bases). Four different grammars are compared on the task of identifying terminators, including two baselines and Grammar Refinement with both methods to compare distributions. The first baseline is the original grammar, and the second baseline is a grammar where the Refinement Operator is applied randomly. Changes that are directed instead of random outperform both baselines, where using the  $\chi^2$  test to determine the kind of transformation performs better than when Kullback-Leibler divergence is used. No overfitting of the data is observed up to the maximum of 25 newly added node labels. Some of the changes introduced agree with known features of terminators that were not considered in the initial grammar.

## 6.2 Tuning a Treebank to a Parser

Few parsers operate directly on treebank data, so that transformations of the data are inevitable. Many attempts to syntactic parsing further specialise their conditioning on the specific properties of a treebank in order to improve performance (Charniak, 1999, p. 9), and often additional transformations are performed on top of those that are strictly necessary. Collins (1999), e.g., systematically relabels non-recursive noun phrases, because this modification eliminates many analyses preferred by PCFGs yet never seen in the training

data (p. 211). Two attempts to explore systematically such transformations show best the influence of the chosen representation on parser performance.

### **Klein and Manning (2003b)**

Klein and Manning (2003b) show that relying simply on a plain PCFG treebank grammar in combination with prior transformations of the treebank can improve performance further than previously assumed. Prior to them, it was common wisdom that poor performance of such an approach was due to limited access to lexical information, and that such access could only be granted by more complex conditioning of the probabilistic models. They argue that most useful information for parsing does not come from *bilexical* dependencies between two lexical items, but that resorting to *monolexical* dependencies between a lexical head and some surrounding structure works almost as well. Probabilities of monolexical dependencies require less data for reliable estimation, and when the set of lexical items to be considered for inclusion into the model is further reduced to high-frequent function words, then the transformations can even be controlled and interpreted by inspection.

Consequently, several linguistically motivated changes to the structure of the Penn Treebank are performed and evaluated in a “partially manual hill-climb”. Changes mainly attack well-known peculiarities of the treebank such as the rather flat structure, or overly general POS tags for punctuation and auxiliary verbs, but they also introduce new distinctions like splitting verb phrase labels to mark those with finite head verbs (raising performance by 2.7%  $F_{lab}$ ). Other transformations mark node labels dominating a unary production (further distinguishing unary determiners and unary adverbs), add parent annotation to POS tags, and further distinguish the general POS tags for six different types of prepositions, for the special auxiliaries *be* and *have*, the conjunctions *but* and *&*, and the percent sign. Functional annotation is added for temporal noun phrases, and clauses are marked in raising and control constructions. Possessive noun phrases receive distinct labels, and finally, attachment height is encoded by distinguishing non-recursive noun phrases, those that contain some verb (of a modifying relative clause), or that have some right-most noun phrase descendant.

All these changes lead to a performance that is between early and best performing lexicalised parsers. The benefit of these transformations is especially significant given that they “could presumably be used to benefit lexicalised parsers as well”. It would be desirable, though, to automate the process of finding the parts of a treebank that should be modified, because considerable effort is necessary to determine them by manual hill-climb.

**Schiehlen (2004)**

Schiehlen (2004) attacks a different parsing problem with a similar motivation as Klein and Manning (2003b). He shows that a German corpus (the *negra* treebank) can also be optimised by close inspection of the properties of the language, and by considering the encoding of the analyses in the treebank. He evaluates the performance of a PCFG treebank grammar derived from the transformed corpus by Parseval and dependency metrics, and uses the latter to direct optimisations. Transformations of the treebank often show opposite effects on performance according to the two metrics.

Some changes to the treebank are directly comparable to those applied by Klein and Manning (2003b) and include splits of auxiliary verb POS tags, adding functional information to non-terminal node labels, or marking verb phrases for finite or infinitival head verbs. Other modifications are not as successful as they proved to be for Klein and Manning, such as attempts to generalise rules either by markovisation (i.e. splitting right-hand sides of rules into Markov chains) or by merging certain labels of nodes (grouping coordinations, or distinguishing pre-terminals from non-terminals, or multi-word from single-word lexemes). Some other modifications target specific properties of German, such as percolating case information from the node that covers the full argument down to the level of POS tags, which gives the highest individual performance boost. Still other modifications are closely connected to the encoding of syntactic annotation in *negra*, such as marking sentential nodes as relative clauses, which standardly receive the same label, or by splitting collections of subsequent sentential trees that were erroneously concatenated under a single root by the annotators. Finally, external information is added to the treebank in the shape of subcategorisation information and a named entity recogniser.

The majority of transformations improves performance over the baseline individually, and shows greater improvements in combination. Performance on the test set is better than that of a lexicalised parser (Dubey and Keller, 2003) on a very similar test set even when only few of the above transformations are applied. The effect of transformations on a German data set is thus noticeable, while the effort of Klein and Manning (2003b) has to be repeated in order to find the most useful changes.

## 6.3 Parsing German

Several treebanks exist for German,<sup>1</sup> and roughly as many times these data sets have been used to produce and evaluate syntactic parsers. Unfortunately, even results for the same treebank are rarely comparable, because no standard division into development and test sets seems to have emerged for them. We therefore only briefly introduce those approaches also using context-free grammars for full parsing, and do not compare the absolute figures of Parseval or dependency evaluation between the approaches. We know of three attempts to perform or support full syntactic parsing of German with PCFGs including Schiehlen (2004) above. Further attempts exist to parse German with other models such as memory-based parsing or discontinuous phrase structure grammar (Kübler, 2002; Plaehn, 1999), or where PCFGs are employed to a different end, e.g. to determine subcategorisation frames or to assign POS tags that are morphologically more detailed (Beil & al., 1999; Trushkina and Hinrichs, 2004).

### Dubey and Keller (2003)

Dubey and Keller (2003) re-implement the parser presented by Collins (1999) and evaluate it on a data set that is very similar to the one used by Schiehlen (2004). They are the first to perform full syntactic analysis via probabilistic parsing for German. They point out that peculiarities of German such as free constituent order, and of the *negra* treebank require appropriate changes in the grammar model. Some of the choices specific to the treebank are also pointed out by Schiehlen (2004), such as the general label for main, subordinate, and relative clauses. Dubey and Keller (2003) try to find out whether successful models developed for English and the Penn Treebank are equally applicable to their data, and specifically, whether lexicalisation provides the same benefit for German as it does for English.

They perform two sets of experiments and evaluate success via Parseval  $F_{lab}$ . Parsing is always performed on previously POS-tagged text in order to factor out differences caused by smoothing probabilities of lexical items. They first measure performance of a plain PCFG treebank grammar and compare it with straightforward head-lexicalisation (Carroll and Rooth, 1998) and with a model similar to Collins (1999) that also markovises the right-hand sides of rules (Collins, 1997). They restrict the latter model to use only dependencies between dominating heads (called *head-head* dependencies), and not between neighbouring heads (*sister-head* dependencies). Much to their

---

<sup>1</sup>Tiger, *negra*, TüBa-D/S and TüBa-D/Z have been introduced in chapter 4.

surprise, both lexicalised models perform worse than the unlexicalised baseline. This observation still holds when additional information in the form of function labels is added to the node labels. Rather flat learning curves show that this behaviour is not caused by sparse data. Instead, a second set of experiments shows that it is caused by differences in the encoding of syntactic analyses between the Penn Treebank, for which the parsers were originally developed, and *negra*. In *negra*, the nominal part of a prepositional phrase is marked by edge labels, and not by an additional phrasal node. Incorporating conditioning on neighbouring heads into the probabilistic model boosts performance considerably.

Just as a treebank can be tuned to a PCFG parser, a parser can be tuned to a treebank. In fact, it seems to be necessary to tune a parser in order to achieve best performance. The space of possible extensions to conditionings of rule expansions is vast, so that intuition about the peculiarities of a language, and more specifically of a treebank, have to restrict this space in order to allow parameter estimation in the given framework.

### Frank *et al.* (2003)

Frank *et al.* (2003) filter the input of a constraint-based HPSG parser by a PCFG that annotates topological fields. Their goal is to improve throughput and robustness of the constraint-based parser that delivers detailed and precise analyses. In addition to enhancing efficiency, they also try to take advantage of the probabilistic verdicts on the quality of alternative parses licensed by the constraint-based parser to rank the otherwise equivalent analyses.

Obviously, the compatibility of the analyses of both parsers is a prerequisite to their combination. Filtering via a topological field parser seems to be especially suitable, because topological fields provide an underspecified and theory-neutral description of full syntactic analyses, and can be determined efficiently and reliably (Becker and Frank, 2002; Veenstra *et al.*, 2002). The constraint-based parser is directed via *priorities* assigned to edges considered for parses depending on their compatibility with given bracketings. Edges producing crossing brackets, e.g., are penalised, while rewards or penalties can be specified for any matching constraints. These rewards and penalties are changed from their default values when the topological field parser has high confidence in a parse. Confidence is determined from the tree entropy of the corresponding parse according to the PCFG, and from an expected accuracy determined from average precision per label type on separate train and test data. The combination of both parsers is evaluated in terms of coverage and speed relative to the constraint-based parser alone. The speed is



consistently doubled and reaches its peak when only topological field structure with low entropy or very high expected accuracy is considered. Filtering by topological fields outperforms similar filtering by chunk bracketings.

Frank & al. (2003) show the usefulness of PCFG models as pre-processors for more powerful but less efficient parsers. The success of PCFG models on annotating clause structure in terms of topological fields is an essential prerequisite to these results, as well as the compatibility of topological fields with other kinds of analyses.

## 6.4 Conclusion

Syntactic parsing with plain PCFGs is a current research topic and shows considerable success in supporting less efficient parsing methods, and in terms of absolute performance, where it was considered to have no chance of reaching performance levels of more sophisticated approaches to probabilistic parsing until recently. Most of this research focuses on the deficiencies of PCFGs in their ability to model relevant context and determines appropriate changes in the representation of syntactic analyses found in treebanks. Decorating node labels with additional information seems to be the method of choice, as opposed to introducing or removing nodes and edges, which is performed mainly when coverage is an issue and right-hand sides of rules are broken up into Markov chains. Changes guided by close inspection of treebank analyses are most successful but also most costly. The insight that representations of syntactic analyses found in treebanks are not necessarily optimal for PCFG parsing, and that equivalent representations can be found that are more appropriate for this task is a recurring idea in the research mentioned here.



Part III

Treebank Refinement



## Chapter 7

# Including Local Context into Nonterminals

Context-free grammars are in widespread use, because there are algorithms that allow calculation of all parses in at most cubic time relative to the length of the sentence, and much less on average for calculating only a best parse for probabilistic context-free grammars (Klein and Manning, 2003a). Despite their simplicity, they are capable of modelling recursiveness, which is vital for processing natural language, where embedding is theoretically unbounded. At the same time, it is clear that natural language has phenomena that exceed the descriptive power of context-free grammars, most prominently cross-serial dependencies.

Our focus is on syntactic parsing, performed by learning from a treebank. Reading a probabilistic context-free grammar directly off a treebank (Charniak, 1996) has more often than not resulted in poor parsing results (Corazza & al., 2004; Dubey and Keller, 2003). In fact, such treebank grammars often serve as a baseline, because they can be easily obtained and applied using new data and standard algorithms. Their weakness has usually been attributed to the inability of context-free grammars to model context adequately. A line of research adding sophisticated extensions to CFGs has led to the best performing parsers to date, mainly adding lexical and dependency information not captured well in the node labels of the underlying treebank (Collins, 1999).

Parsers based on context-free grammars, but extended to handle lexical dependencies, and thus departing from the rules given in treebank annotation, have proven to be very successful. It is unclear, though, to what extent these extended models exceed descriptive power of CFGs. The limitations of CFGs with respect to their ability to model context are usually especially severe in combination with certain representations of context in treebank an-

notation of syntactic phenomena. Choosing an annotation scheme has strong impact on their performance (see section 5.1), but the annotation scheme of a treebank is primarily chosen to let linguists encode and interpret syntactic analyses. Human labour performing the annotation is costly, so that also practical considerations will have an influence on choosing an annotation scheme for a treebank.

The annotation scheme of a manually annotated treebank will therefore be just complex enough to allow encoding of the desired analyses. Redundancy will be avoided, because human annotators should be faced with the minimal set of decisions, avoiding inconsistencies caused by slips of the pen, or lack of attention, and last but not least redundancy will be avoided in order to save labour. Trees will be chosen to look like the trees commonly used to describe linguistic analyses. The stylebook that encodes the interpretation of the annotation will be present in the users' minds, and it will usually be necessary to read it in conjunction with the annotation in order to understand the annotators' decisions. This means that a probabilistic context-free grammar, which only predicts a node's expansion from its label, has no access to this information. In TüBa-D/Z's representation of German, e.g., a noun phrase in the complementiser field is represented by the same node label as a noun phrase in the middle field. Expansion probabilities differ once you know about the larger context, however, which will not surprise a human user looking at the tree in total.

The weakness of a PCFG is simple to paraphrase. It is the weakness of node labels to distinguish expansion probabilities properly when used in different broader contexts. A number of questions arise when trying to put this paraphrase into use to improve the performance of a PCFG. These questions will be discussed in this chapter: In which way are context-free grammars limited when they describe natural language syntax (section 7.1)? Which information should be included into a node label (section 7.2)? How can the difference between expected and observed behaviour of a node label be assessed (section 7.3)? When does the increase in grammar detail have negative impact on performance (section 7.4)? Simply adding information about the parent node label to each node in the treebank has proven remarkably efficient and will serve as a comparison in conclusion (section 7.5).

## 7.1 More Context in Context-Free Grammars

Context-Free Grammars have their name from limitations in carrying information through the nodes of a structural analysis. They capture relations between immediately dominating nodes and between sister nodes in (context-

free) production rules. These rules establish links between the sister nodes that are given in their right-hand sides, and between these sisters and their common parent given in the left-hand side of each rule. Relations to grandchildren (of a left-hand side) or grand-parents (of a symbol in a right-hand side) cannot be stated directly in a single rule.

These relations can only be stated by adding new non-terminals for this purpose (see figure 7.1). If a direct connection needs to be drawn between  $A$  and its grand-children  $C, D, E$ , this can be accomplished by assigning  $B$  a new unique label when it is part of a right-hand side of a rule with  $A$  as left-hand side (we use the new node label  $B-A$ ). As a consequence, each tree fragment as shown in figure 7.1(a) is rewritten as shown in figure 7.1(b). When we express tree fragments as rules, then the context which determines the application of the context-free rule  $B-A \rightarrow C D E$  is now extended to include  $B$ 's parent (see the corresponding rules below the tree fragments in figure 7.1). The name Context-Free Grammars is thus misleading insofar as

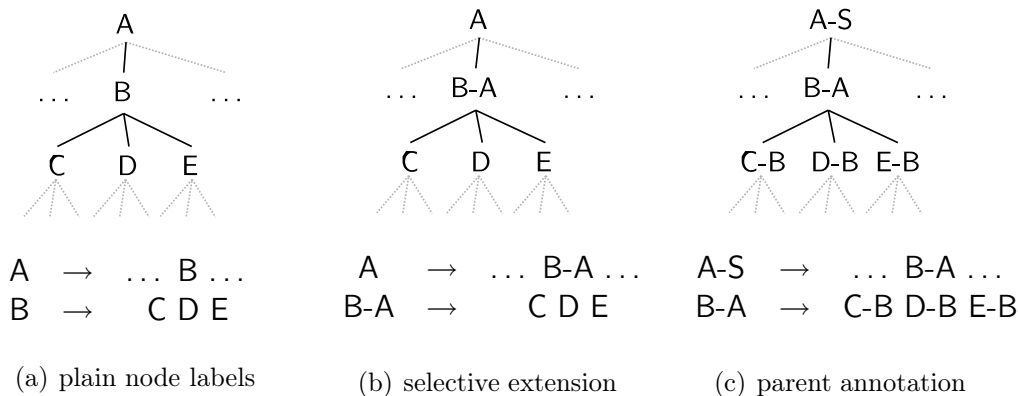


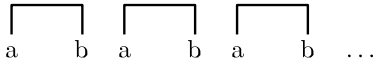
Figure 7.1: Adding Local Context via New Node Labels

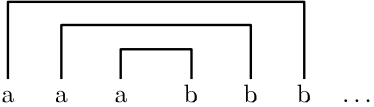
it does not mean that context cannot be modelled at all. It means instead that all context that is relevant for the expansion of a non-terminal symbol has to be expressed in the choice of the symbol. Parent annotation introduces this kind of information for all node labels (see figure 7.1(c), assuming  $A$  is dominated by  $S$ ).

As each non-terminal node is the only link between the inside below it and the outside above it, and the number of different non-terminal nodes that encode links is finite, there can only be a finite number of links passing through each node. The same is true for links between the elements in the smallest subtrees of two levels, which correspond to the rules of a CFG. There is only a finite set of these rules, too, so that in summary, there is only a finite number of types of dependencies that a CFG can encode between the

nodes in trees of any size. If the number of newly introduced nodes were not finite, then the grammar would achieve more than context-free power (Grune and Jacobs, 1990, p. 45). It is in this sense that CFGs have limited expressive power for structural descriptions. Note that the restricted number of types of dependencies does not necessarily limit the number of tuples that form a dependency. Certain types of dependencies can be repeated an unrestricted number of times, like that of direct adjacency of two terminal symbols, or between terminal symbols that are separated by a sequence of terminal symbols that can be derived from a fix sequence of non-terminal symbols (see figure 7.2).<sup>1</sup> In the remainder of this section we try to analyse in which way

$$\begin{array}{ll}
 G_{regular} = (\mathcal{N} = S, & G_{embed} = (\mathcal{N} = S, \\
 \quad S = S, & \quad S = S, \\
 \quad \mathcal{T} = \{a, b\}, & \quad \mathcal{T} = \{a, b\}, \\
 \mathcal{R} = \{(S \rightarrow abS), (S \rightarrow ab)\}) & \mathcal{R} = \{(S \rightarrow aSb), (S \rightarrow ab)\}) \\
 \\
 L(G_{regular}) = \{(ab)^n | n \in \mathbb{N}\} & L(G_{embed}) = \{a^n b^n | n \in \mathbb{N}\}
 \end{array}$$





(a) dependencies between adjacent terminal symbols (b) fix sequence of intervening non-terminal symbols derives string that separates dependent terminal symbols

Figure 7.2: Unlimited Number of Dependents in CFGs

renaming nodes in CFGs, i.e. enumerating certain dependencies, is more restricted than solutions available in a more powerful grammar formalism that can describe these dependencies generically.

### 7.1.1 More Context Required

Cross-serial dependencies are a structural configuration that is different from both types of dependencies shown in figure 7.2. A cross-serial dependency consists of tuples of elements that are connected, where all first elements of all tuples occur first, and then the second elements of all tuples follow in the same order, and then the third elements, and so on. A cross-serial

<sup>1</sup> $G_{regular}$  is right-linear and therefore regular.



dependency consisting of  $n$  tuples of words,  $a_i$  and  $b_i$ , where all  $a_i$  precede all  $b_i$ , and both are in the same order, thus is  $L_{CS} = \{a_1 a_2 a_3 \dots a_n b_1 b_2 b_3 \dots b_n\}$ . Cross-serial dependencies do not occur in English, but they are necessary to describe dependencies between words in sentences for some languages that have a less restricted word order.

Dutch is such a language. The Dutch sentence *Jan Piet Marie zag helpen zwemmen* ('Jan saw Piet help Marie swim'), e.g., has the form  $a_1 a_2 a_3 b_1 b_2 b_3$ , whereas its English translation groups the pairs as  $a_1 b_1 a_2 b_2 a_3 b_3$ . Figure 7.3 shows the syntactic dependencies between the words (cf. Joshi, 1985). On the

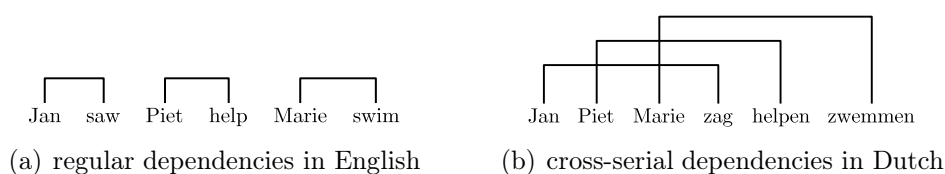


Figure 7.3: Dependencies between Pairs of Words

left-hand side it shows the right-branching structure of English, which can obviously be described by the rather simple context-free grammar  $G_{regular}$  given in figure 7.2(a), because each pair of dependents consists of adjacent terminal nodes. Figure 7.3(b) gives an analysis of the dependencies between the pairs of elements in the Dutch sentence. There is no obvious context-free rendering of this analysis, because the terminal symbols are neither adjacent, nor can the intervening non-terminals in such a kind of sentence be derived from a fix sequence of non-terminal symbols without losing any dependencies. The crossing edges in the graphical representation highlight that a model with more descriptive power is required.

We see that dependencies connecting pairs of words may be much easier to describe than in the case of cross-serial dependencies, when we look at our English translation, where they form the regular language  $L(G_{regular})$ . Dependencies may be much more difficult to express, too, e.g. when all words may occur in any order. An example for such a language would be

$$L_{free} = \{w | w \in \{a, b, c\}^* \wedge count(a) = count(b) = count(c)\}$$

As complete freedom of word order does not seem to be a prevalent feature of natural language, we are less interested in modelling it, as this may result in an unnecessarily complex model (example and reasoning follows Joshi, 1985).

In order to capture cross-serial dependencies, we need a grammar model that is slightly more powerful than Context-Free Grammars, like Tree Adjoining Grammars (TAGs), which are *mildly context-sensitive*. They allow modelling  $L_{CS}$  (and of course  $L_{regular}$ ), but not  $L_{free}$ . TAGs use a finite set of so-

called *elementary trees* to specify structural descriptions.<sup>2</sup> Elementary trees are either *initial* trees or *auxiliary* trees. Initial trees consist of connected non-terminals, and the grammar's start symbol is the single non-terminal at their top. At the frontier, they only have terminal symbols. Auxiliary trees are very similar to initial trees, but they may have a different non-terminal symbol at the top, and they have exactly the same non-terminal symbol at the frontier. Except for this non-terminal, all other symbols at the frontier are terminals. Auxiliary trees can be inserted into initial trees via a so-called *adjoin* operation. This operation replaces one non-terminal symbol  $S$  in an initial tree with the auxiliary tree that has a matching non-terminal symbol at the top and at the frontier. The auxiliary tree is inserted between the outside above  $S$  of the initial tree and the inside below  $S$ . The product of an adjoin operation may be the input for further applications of it. Adjoining allows TAGs to describe linguistic phenomena that go beyond context-free power, including cross-serial dependencies. Dependencies in TAGs hold between the nodes of an elementary tree. These dependencies are visualised by *links*. Adjoining may insert trees between the landing points of the links, so that the links between the nodes of the newly inserted trees may cross the links that have existed before.

Figures 7.4 and 7.5 show how TAGs can be used to analyse our example sentence that contains three cross-serial dependencies (all analyses taken from Joshi, 1985, with changes in the decoration of node labels). The left-hand side of figure 7.4 shows the initial tree  $\alpha$ , and the auxiliary tree  $\beta$ . We do not show the words that would lead to three different initial and auxiliary trees with the yields *Jan zag* ( $\alpha_1/\beta_1$ ), *Piet helpen* ( $\alpha_2/\beta_2$ ) and *Marie zwemmen* ( $\alpha_3/\beta_3$ ). The dashed lines show the links between the two parts of the initial and auxiliary trees that belong together, and that become separated by adjoining auxiliary trees. The rest of figure 7.4 shows sentential trees, i.e. trees that give a complete analysis of a string of terminal symbols after adjoining auxiliary trees. The first sentential tree  $\gamma_1$  is simply the initial tree  $\alpha_3$ . The subsequent sentential trees are generated by adjoining auxiliary trees at the nodes surrounded by circles:  $\gamma_2$  is produced by adjoining  $\beta_2$  to  $\gamma_1$ , replacing the encircled  $S$  in  $\gamma_1$ . When the words of the sentence are put into linear order, the dotted lines of the links cross, as shown in the last sentential tree  $\gamma_3$  given separately in figure 7.5. A context-free grammar does not have the device of adjoining auxiliary trees, which allows parts of the same tree to be separated by inserted material, so that dependencies cross.

When the elementary trees are not composed via the adjoin operation, and instead the non-terminals at the frontier of auxiliary trees are simply

---

<sup>2</sup>We follow the description of TAGs given in Joshi (1985).

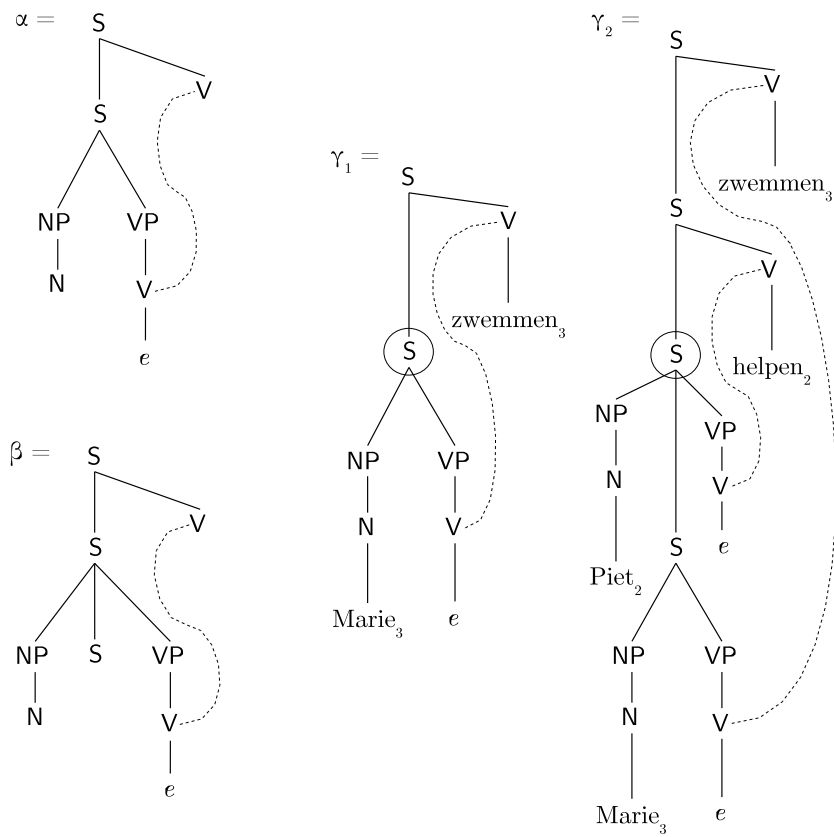
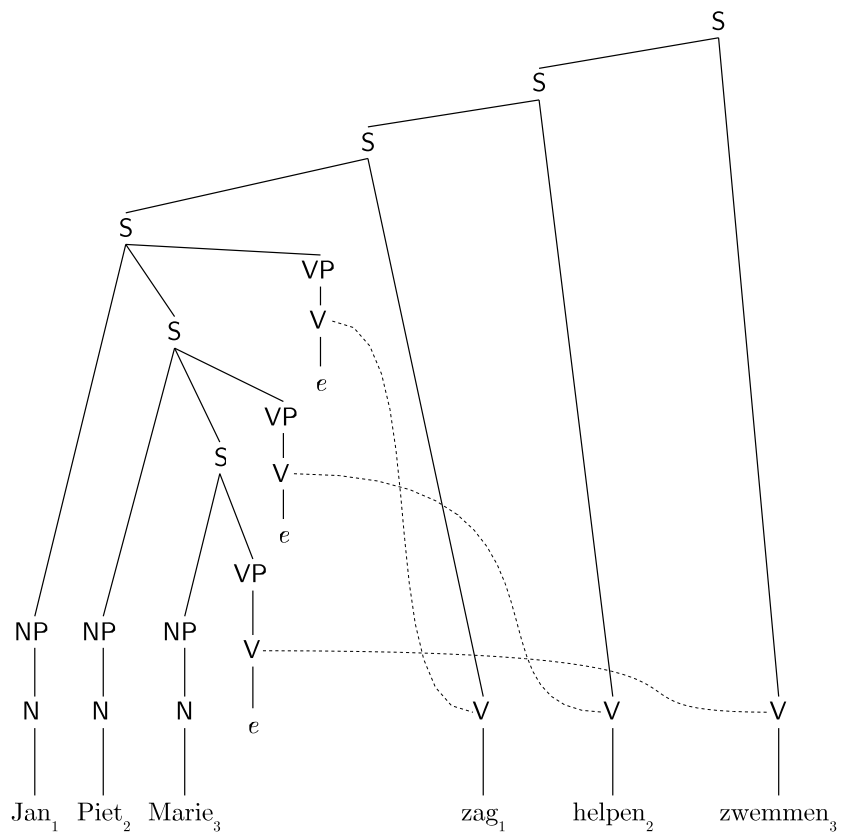


Figure 7.4: Elementary and Sentential TAG Trees

Figure 7.5: A TAG Analysis of Cross-serial Dependencies ( $\gamma_3$ )

replaced by other elementary trees, then we return to context-free power, and instead of TAGs we have context-free grammars which pose *local constraints* on non-terminal nodes. The inclusion of parent information as outlined before is such a constraint: node B has to be dominated by A, or

$$B \rightarrow C D E / \text{DOM}(A \_)$$

in the notation of Joshi and Levy (1982). Local constraints are also capable of expressing more complex concepts like *command*:  $\text{COM}(A B C)$ , where “B immediately dominates A and B dominates C, not necessarily immediately” (which contains an unbounded dependency; Joshi and Levy, 1982).

Local constraints may express, e.g., that two (or any given number of) cross-serially dependent words are dominating the node in the left-hand side of a context-free rule. The difference of major impact on descriptive adequacy between a CFG and a TAG with respect to cross-serial dependencies is that the latter are capable of describing cross-serial dependencies between an arbitrary number of dependents, which are not explicitly enumerated in the inventory of elementary trees. Enumerating them all would be against the idea behind TAG of describing only *minimal* facts on syntactic trees, i.e. facts that cannot be derived from other minimal trees via the adjoin operation. Enumerating cross-serial dependencies allows grammars with context-free power, however, to describe cross-serial dependencies up to a certain maximal level of embedding.

A context-free grammar with the nonterminals

$$\mathcal{N} = \{S, NP, N, VP, V\}$$

and the rules

$$\mathcal{R} = \{(S \rightarrow S V), (S \rightarrow NP S VP), (S \rightarrow NP VP), (VP \rightarrow V), (NP \rightarrow N)\}$$

could generate the same tree as TAG (omitting the obvious rules that expand N and V into words or the empty symbol). It would, however, overgenerate, because it does not encode the connections between the parts of the trees that are represented by the dashed lines. We have seen that the only way to encode a connection between two nodes in a CFG is via local constraints, i.e. via rules connecting nodes that directly dominate each other. Encoding a connection between the highest and lowest V in figure 7.5 thus would require changing all intervening node labels so that they incorporate the information that a number of cross-serial dependencies passes through them. This could be accomplished by changing the individual S to e.g.  $S_{33}, S_{32}, S_{31}, S_{30}, S_{20}, S_{10}$ , where  $S_{kl}$  means that  $k$  left cross-serial dependents and  $l$  right cross-serial



number of dependent pairs. Such a restriction is linguistically unconvincing and theoretically unappealing. The situation is different and the impact is less heavy when we restrict ourselves to modelling a certain finite language sample (e.g. our treebank), where the number of pairs of words or constituents that are engaged in a cross-serial dependency is always finite, and thus also has a finite maximum size. Recognising cross-serial dependencies up to this size is perfectly feasible with context-free power as we have just seen. Note that both context-free and tree-adjoining grammars do not produce analyses that are similar to those depicted in figure 7.3(b). Both grammars do, however, produce analyses that can be interpreted to express the same analysis as the simple characterisation presented there, so that both TAG and CFG analyses require a certain effort in interpretation.

The full CFG that produces the parse tree in figure 7.6:

$$\begin{aligned}
 G_{CF} = (\mathcal{N} = \{S_{33}, S_{32}, S_{31}, S_{30}, S_{20}, S_{10}, NP, N, VP, V, V_e\}, \\
 S = S_{33}, \\
 \mathcal{T} = \{Jan, Piet, Marie, zag, helpen, zwemmen\}, \\
 \mathcal{R} = \{\text{as given in figure 7.6, plus rules for all words}\})
 \end{aligned}$$

describes a finite amount of context, so that the tree fragment dominated by  $S_{33}$  is big enough to dominate all nodes that are involved in the cross-serial dependency. In fact, all nodes  $S_{kl}$  always occur in a given order when  $S_{33}$  occurs, and  $G_{CF}$  produces only the analysis shown in figure 7.6. In this sense, the tree fragment connecting  $S_{33}$  with all NP and V could be replaced by the single node  $S_{CS3}$  and the single production  $S_{CS3} \rightarrow NP NP NP V V V$  (see figure 7.7). Just as we have given a prescription for how  $k$  and  $l$  mark

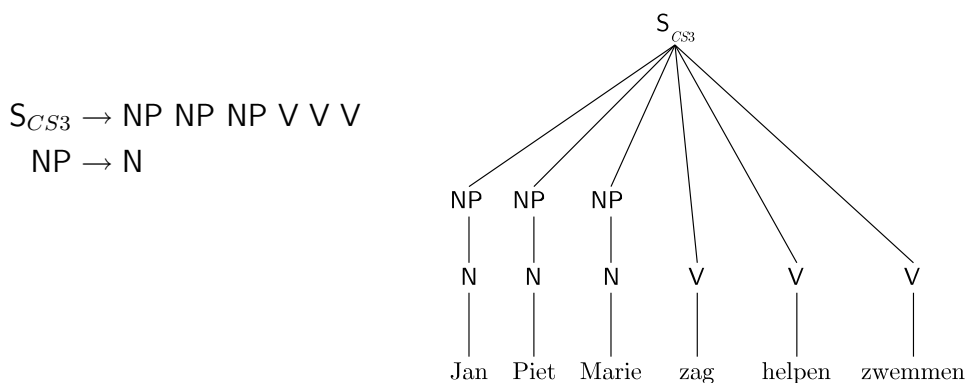


Figure 7.7: Alternative Context-Free Analysis of Cross-serial Dependencies

the parts of the tree that belong together, we can also say that the node  $S_{CS3}$  encodes a cross-serial dependency of three tuples, or the corresponding

tree fragment in  $G_{CF}$  if we prefer a more verbose representation. The full grammar is:

$$\begin{aligned}
 G_{CF\text{FLAT}} = (\mathcal{N} = \{S_{CS3}, \text{NP}, \text{N}, \text{V}\}, \\
 S = S_{CS3}, \\
 \mathcal{T} = \{Jan, Piet, Marie, zag, helpen, zwemmen\}, \\
 \mathcal{R} = \{(S_{CS3} \rightarrow \text{NP NP NP V V V}), (\text{NP} \rightarrow \text{N}), \\
 \text{plus rules for all words}\})
 \end{aligned}$$

$G_{CF}$  and  $G_{CF\text{FLAT}}$  define the same language, but not the same structural analyses, in terms of nodes and edges. Semantically, they are equivalent, because we define them to express the same phenomenon.

### 7.1.2 Formal Equivalence of Grammars

In a formal sense, equivalence between grammars is defined as describing the same language (*weak equivalence*), or describing the same language and at the same time assigning the same structural analyses to each string of the language (*strong equivalence*). From what we have seen so far it is clear that there are TAGs for which there are no strongly equivalent CFGs. It is also true that there are TAGs for which there are no weakly equivalent CFGs, e.g. the TAG that describes the language  $L = \{a^n b^n c^n | n \geq 1\}$  cannot be modelled by a CFG. It is interesting to apply these notions of formal equivalence to the grammars  $G_{TAG}$ ,  $G_{CF}$  and  $G_{CF\text{FLAT}}$ , which all describe cross-serial dependencies. It is clear that  $G_{TAG}$  and  $G_{CF}$  are neither strongly nor weakly equivalent.  $G_{CF}$  and  $G_{CF\text{FLAT}}$  are weakly equivalent but not strongly so, because they assign different structure to the same strings. This difference highlights that strong equivalence heavily depends on representational issues, because it does not consider the interpretation of a structural description, which is not part of the formal model. We have seen above that the single top symbol  $S_{CS3}$  in  $G_{CF\text{FLAT}}$  can be interpreted to be just a shorthand for all symbols  $S_{33}, \dots, S_{10}$  in  $G_{CF}$ , in the sense that  $S_{CS3}$  simply replaces a finite set of rules and nodes used exclusively here.  $G_{CF}$  and  $G_{CF\text{FLAT}}$  are not strongly equivalent only due to this representational variation.

We would like to draw a line between the structure assigned by a model and the interpretation of this structure, so that these two steps are considered independently. We are used to describe syntactic structure using the notational vocabulary of formal languages (trees with typed nodes), but we are not used to interpret the resulting trees from the perspective of a grammar model with restricted power. A node label in a context-free grammar



is atomic, and two node labels are either identical or different. As a consequence, the labels  $S_{33}$  and  $S_{32}$  in figure 7.6 are as different as the labels VP and NP, although the node labels suggest to a linguist by convention that the sentential nodes  $S_{kl}$  are more closely related. A linguist who knows German will hardly argue that there is no connection between the verb’s subcat frame and the case markers of the words that it takes as arguments, even though there is no evidence of case information in the nodes of the graph he uses to describe constituent structure that connects the verb and its arguments.

We argue that the limitations of context-free language models to take relevant context into account are less severe when we choose a formally adequate representation of syntactic analyses instead of a given linguistically adequate representation. As long as both representations convey the same information, they can be optimised independently. A certain type of grammar may require certain notational variants to model a certain syntactic configuration faithfully, and these formally adequate representations usually do not coincide with the variants preferred by treebank annotators and users.

### 7.1.3 Experimental Adequacy

Our goal is to judge whether a context-free grammar that has certain restrictions on its ability to model context is appropriate for our task of parsing natural language data. The notion of formal equivalence does not help us here, because in order to determine equivalence, we need two grammars, or the exact set of sentences in the languages that they license. We do not have the grammar that produces our natural language data. We only have a sample of natural language that we find in a treebank. From this sample together with the analyses provided by the treebank annotators we can gather that certain phenomena may exist in a language. We cannot judge, though, whether a grammar that we use is equivalent in a formal sense to the grammar that has produced the data. Even when our grammar is able to reproduce all strings and all analyses in our treebank faithfully, we do not know whether this corresponds to equivalence, as our grammar cannot be tested against strings that are not in our sample. For example, a simple grammar that just enumerates all analyses in our treebank will most likely not be equivalent to our natural language grammar, because it will fail miserably on any sentence not recorded in the treebank. Our experimental setup therefore tests performance of the grammar on data that has not been used during the construction of the grammar (see section 5.2.3).

Context-free language models can define languages that include languages which exceed context-free power.  $L = \{a^n b^n c^n | n \in \mathbb{N}\}$ , e.g., is not context-free, whereas  $L_{over} = \{a^n b^n c^m | n \in \mathbb{N}, m \in \mathbb{N}\}$  is, and  $L \subseteq L_{over}$ . On the

one hand, context-free languages tend to overgenerate, because they fail to model certain restrictions. On the other hand, they can take advantage of the semantics of node labels to undergenerate, i.e. to model a *subset* of the desired language. For a given upper bound  $l \in \mathbb{N}$ , a CFG can model  $L_{under} = \{a^m b^m c^m \mid m \leq l\}$ , and  $L_{under} \subset L \subseteq L_{over}$ .

The degree to which our grammar can cope with the complexities in our treebank shows how adequate it is for our problem of modelling syntactic structure. We cannot test a grammar for equivalence with a grammar producing natural language by using a treebank. We can, however, say that a grammar that reliably reproduces the analyses provided in the treebank is powerful enough for exactly this task. We are not very interested in notational variations of analyses that can be interpreted to express the same observation like the variants given by  $G_{CF}$  and  $G_{CF\text{FLAT}}$ . We are more interested in configurations that truly exceed the power of our type of grammar. Even for data that is truly problematic for CFGs, it may be sufficient to enumerate syntactic configurations when the context-free grammar is not capable to model them in a general way. We determine the degree to which our grammar succeeds to model these configurations experimentally. We judge the grammar's analyses by normalising notational variants. We vary the notation for the grammar, in order to find the best representation for our grammar among those equivalent to the analyses provided by the treebank annotators.

Standard evaluation methods give us a normalised interpretation of syntactic analyses. The idea of relational evaluation is to give us a reduced inventory of relations that allows us to test similarity of interpretations instead of strong equivalence. When we define a mapping of both our CFG analysis consisting of a single node as given in figure 7.7 and of our analysis according to TAG as given in figure 7.5 to a dependency analysis, then both the dependency analyses will look exactly like our most abstract representation of cross-serial dependencies in figure 7.3(b).

Apart from practical considerations that force us to model only a finite subset of natural language, there are also theoretical considerations that suggest that there are bounds on the number of types of dependencies, and on the number of dependents in natural language. A trivial but hard bound on the number of dependents in any sentence is the limited lifetime of a human being, which however, does not seem to be low enough, because it is unlikely that all series of dependencies that can be uttered in a lifetime are stored in a human brain. Other considerations are more precise, suggesting e.g. that central embedding, which can even be modelled by CFGs without an upper bound, is very infrequent at more than four levels. Sampson (2001) argues that multiple central embedding is possible after all (a view not held by all of his colleagues), and he indeed has recorded a number of embeddings of up to

four levels (p. 18). His records show that assuming a small maximum depth of embedding covers most data. This is in line with Karlsson (2004), who has found a single example that exceeds the depth of four on the more common type of final embedding, which is the rather special case of the nursery rhyme “This is the house that Jack built”. Enumeration might be more appealing for central, and maybe even for other forms of embedding than even an already rather restricted CFG model. The restrictions of CFG models may thus turn out to be not as harmful as suggested by their failure to generate unlimited numbers of cross-serial dependencies.

We try to determine the success of a CFG to model language experimentally. We generate a probabilistic context-free grammar from part of our treebank as outlined in chapter 5. Apart from the changes mentioned there that are necessary to model the treebank analyses with a CFG at all, we leave the nodes and the edges of the original analyses unchanged in all our experiments. We only modify node labels, which can turn a grammar that vastly overgenerates into a grammar that slightly undergenerates and correctly describes complex syntactic configurations up to a certain complexity although beyond the power of the grammar in the general case. The mapping that we need in our experiments reported below is simple, because we only specify subclasses of given node labels and leave the rest of the graph unchanged (merging classes of nodes as discussed in section 9.1 is an exception). After assigning the respective original class to each label, the type of linguistically adequate analysis is restored (e.g. stripping  $kl$  from  $S_{kl}$  restores  $S$  in figure 7.6). Our goal is thus to add contextual information in new node labels when certain types of context have not been distinguished by the treebank designers, and when introducing these distinctions leads to a language model that performs better. We have shown that even distinctions exceeding the power of CFGs can be made to a certain degree by enumerating instead of generating them, when a prescription to interpret their representation is given.

## 7.2 Context – Focus – Production

Context-freeness assumptions in PCFGs are harmful where they result in models that over- or underestimate the probabilities of derivations. Correct derivations, for us, are stored in a treebank, and in a context-free perspective, these derivations consists of a sequence of independent decisions. We thus compare automatically annotated and correct gold data by comparing these atomic decisions. They will occur in just the correct frequencies on average, because this is how the model has been obtained: by maximising the

likelihood of the data for the given model. What is more relevant here is how much the atomic decisions change when more context is taken into account. The answer to this question directly shows the negative impact of context-freeness assumptions. If the production probability of a certain non-terminal stays the same, no matter in which context it occurs, then context-freeness assumptions will not have negative impact. Recall that one of the independence assumptions inherent in PCFGs is that the probability of a subtree only depends on the non-terminal nodes inside of the subtree, and not on those outside of it. If, in contrast, the distribution of children of a certain type of node is different when looking only at the subset of its occurrences in a certain context, then this node label is likely to be a good candidate for inclusion of this context.

There are three parts that serve to compare a given context-free language model with a model containing more context. The first part is a certain node label which is examined in order to find out whether it is appropriate. We will henceforth call this node the *focus* node. The focus node connects the two other parts of a derivation tree that we need for a comparison: the inside and the outside of the tree relative to the focus node (see figure 7.8).

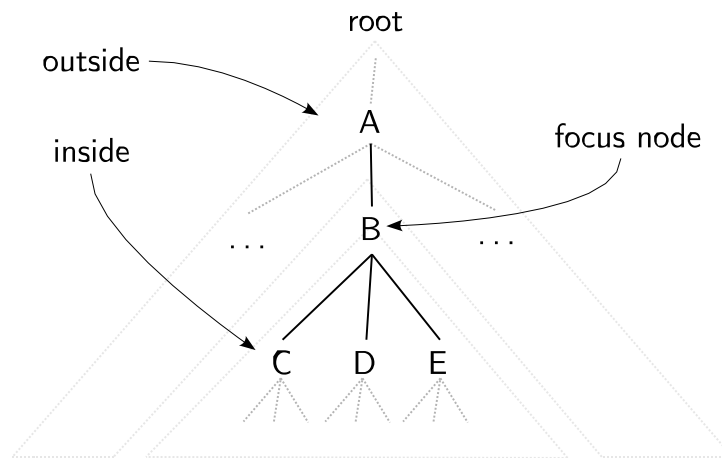


Figure 7.8: A Focus Node Connects Inside and Outside of a Local Tree

Looking at a top-down derivation of a sentence, we will normally consider the focus node to be on the left-hand side of a context-free production rule that has the focus node's children on the right-hand side. The context that should be tested for inclusion will then be in the outside of the tree according to figure 7.8. A very simple scenario where only the parent of the focus node is tested against inclusion is shown in figure 7.9. The parent is the minimal upward context, because any information from the outside part of the tree

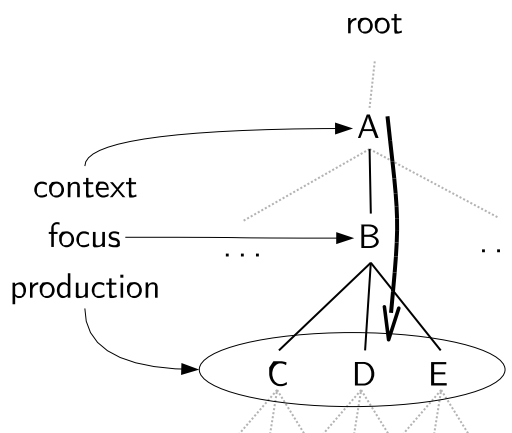


Figure 7.9: Top-down view on context – focus – production

that should be included into the focus node needs to pass through the parent node.

Transferring these general observations to labelled nodes in a treebank, we assume that a constituent label generalises perfectly when its children do not change systematically with the context in that it occurs. In order to verify this hypothesis, we take a look at noun phrases (NX) in TüBa-D/Z, and how the distribution of their productions changes under certain parents, namely the complementiser field (C), the middle field (MF), or embedded in a complex noun phrase as given in figure 7.10 below. Table 7.1 shows the occurrence frequencies of three different expansions of noun phrases that consist only of single words, represented by their POS: a substituting relative pronoun (PRELS), a common nouns (NN) and a personal pronoun (PPER). The second column shows the number of times each production is observed

Production	Context						
	all obs.	C exp.	obs.	MF exp.	obs.	NX exp.	obs.
PRELS	1805	33	1503	424	2	632	3
NN	13922	256	0	3274	1361	4878	7366
PPER	5641	104	0	1326	3797	1976	100
others	79947	1472	363	18799	18663	28011	28029
all	101315	1866		23823		35498	

Table 7.1: Selected Productions of Focus NX in Parent Context

in the whole treebank (*all*). The subsequent three pairs of columns show

the observed frequencies of the productions when looking only at nodes in certain contexts (*obs.*) and also the number of times each production would be expected assuming the same distribution in this context as over the whole corpus (*exp.*). The parent nodes that make up the contexts represent the Middle Field, the Complementiser Field and a complex noun phrase. The second last row gives frequencies of productions not listed separately in the table (*others*), and the last row shows how many times the focus node occurs in each context overall (*all*).<sup>3</sup>

The example shows that the distribution of productions of nodes labelled NX changes considerably with context. For all productions and all given contexts, the node label alone fails to predict the observed frequencies. The information present in the contexts is at least twofold. First, the Complementiser Field and the Middle Field represent distinct linear positions in German clauses. Second, the NX context represents part of a complex noun phrase (i.e. recursive embedding), as opposed to a focus NX that is dominated by nodes with field labels. A production consisting of POS in conjunction with a field context only allows for a simple focus noun phrase.

Figure 7.10 shows a tree fragment with an example for each case that is underrated according to Table 7.1: relative pronouns are frequently found in the Complementiser Field, personal pronouns in base noun phrases in the Middle Field, and common nouns in the embedded part of complex noun phrases.<sup>4</sup>

According to the division of the data into context, focus and production, there is a link missing between the context and the focus for all three types of NX in context. Dividing all NX into subclasses, with one new subclass for each context, alleviates the problem of incorrect context-free estimations of productions. The new estimation still predicts production probabilities only from the focus node label, but the new labels incorporate contextual information. The new name can be chosen arbitrarily, as long as it only occurs in one type of context. For the new node labels that occur exclusively in one context, the expected occurrence frequencies match exactly the observed frequencies, i.e. in table 7.1, the *exp.* and *obs.* columns become identical for the C, MF and NX contexts. Node label transformations of this kind are what we will call *Trebank Refinement*.

---

<sup>3</sup>The two occurrences of PRELS in MF are errors. Recall that we base our experiments on automatically POS-tagged data.

<sup>4</sup>Figure 7.10 is taken from TüBa-D/Z sentence 1266: *Die Soloalben , die er im Laufe der achtziger Jahre veröffentlichte , hatten nichts mehr vom Disco-Glamour früherer Japan-Tage .* ('The solo albums that he released during the eighties had no connection to the disco glamour of early Japanese days.') Focus and Production are shown in bold, and the context with a double outline.

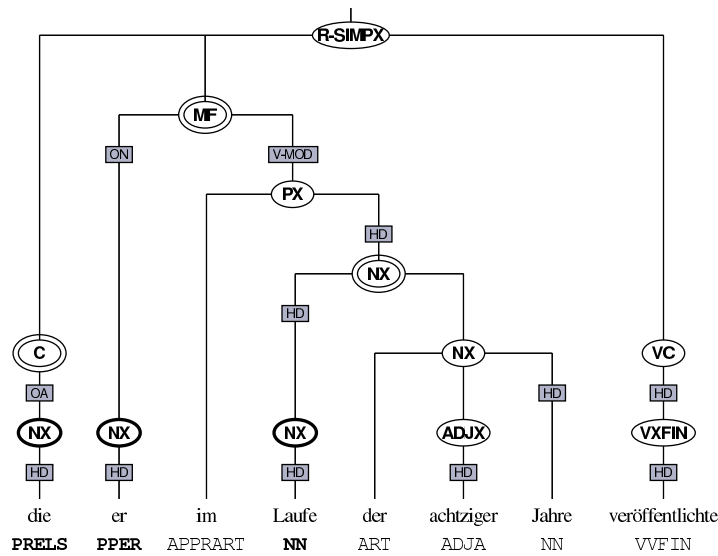


Figure 7.10: Unexpected Productions of a Focus Node in Context

From a more general point of view, the focus node connects some parts of the inside and the outside of a tree. This does not necessarily have to happen in a top-down fashion, which would imply that some part of the outside has impact on all children of the focus node. It could also be the case that only part of the inside (i.e. only some children) are connected to the outside. This can be accommodated into the notion of context, focus and production by reversing the direction from top-down to bottom-up (see figure 7.11). This extension to the notion of context will be discussed, along

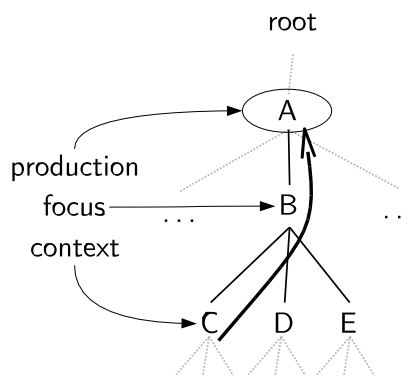


Figure 7.11: Bottom-up View on Context – Focus – Production

with others, in chapter 8. In the remainder of this chapter, we will restrict ourselves to parent node context in a top-down view.

### 7.3 Determining Deviant Distributions

A PCFG language model obtained from a treebank can be harmed by exaggerated independence assumptions. In the last section we have argued that the appropriateness of node labels, which express these assumptions, can be tested by looking for node labels that differ in predicted distributions of productions between smaller and wider contexts.

The distributions are given by the frequencies of productions of focus nodes with and without context. The reference distribution is the distribution of productions of a focus node alone. This equals the frequencies that are used to estimate a PCFG from the treebank by maximum-likelihood estimation. Given a certain focus node type, the set of different productions and their frequencies is easily obtained from treebank data. For a given focus node type, this results in a set of productions with accompanying frequencies. Adding a context results in choosing a subset of productions from this reference set, with corresponding (reduced) frequencies.

To determine the most deviant distribution in a treebank, we look at all types of focus nodes in turn, and for each focus node type we look at all types of contexts in turn. For each combination of focus node type and context type, we have a number of different production types as sketched in figure 7.12, which shows a single focus node type  $f$  that occurs in context types  $c_1$  and  $c_2$ . In these contexts we find three different production types

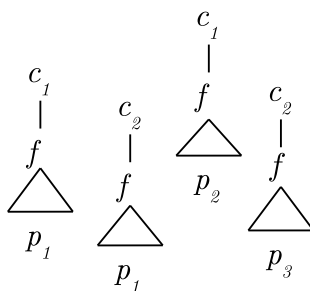


Figure 7.12: Some Occurrences of a Single Focus Node Type in a Treebank

$p_1, p_2, p_3$ , where  $p_1$  occurs twice and the other two types just once. We write this as

$$\begin{aligned} \text{freq}(c_1 > f \rightarrow p_1) &= \text{freq}(c_1 > f \rightarrow p_2) = 1 \\ \text{freq}(c_2 > f \rightarrow p_1) &= \text{freq}(c_2 > f \rightarrow p_3) = 1 \end{aligned} \quad (7.1)$$

The focus node  $f$  occurs two times with production  $p_1$ , and once with each



$p_2$  and  $p_3$ , which we write as

$$\begin{aligned} \text{freq}(f \rightarrow p_1) &= 2 \\ \text{freq}(f \rightarrow p_2) &= \text{freq}(f \rightarrow p_3) = 1 \end{aligned} \tag{7.2}$$

The reference distribution is given by the frequencies in equation 7.2. This distribution is compared with each subdistribution given by a distinct context type in equation 7.1. We consequently determine  $\text{freq}(c > f \rightarrow p) = 0$  for any production  $p$  that has never been observed for focus node  $f$  in context  $c$ .

These frequencies are all we consider for the deviance metric

$$D(c > f \parallel f)$$

which compares the behaviour of the focus node  $f$  in the reference case with its behaviour when it occurs in context  $c$ . When we are looking for the most deviant distribution of a focus node  $f$ , then we are looking for some context  $c$  that maximises  $D$ . When we are looking for the most deviant distributions in a treebank, we seek to find the largest  $D$  for any combination of focus node and context. We expect such a deviant distribution to be a good candidate for renaming, because it represents most harmful independence assumptions according to our method.

We seek to compare distributions of events that follow a certain frequency distribution. One of them represents the underlying distribution. The other distribution is assumed to originate from the same distribution, and we would like to test whether this is actually the case. Inferential statistics offers goodness-of-fit tests, which try to answer exactly the question whether an observed distribution is the same as an assumed underlying distribution, i.e. in our case whether the contextual distribution is drawn from the reference distribution.

### 7.3.1 $\chi^2$ Goodness-of-Fit

We need a test which is non-parametric, because we do not know any details about the distributions. A standard non-parametric test is Pearson's  $\chi^2$  test (Rinne, 1997, p. 552), which will serve as the first instantiation of our deviance metric,  $D^{CS}$ . We use this test as a goodness-of-fit test, which checks whether a distribution fits an assumed underlying distribution. The underlying distribution is our reference distribution, represented by production frequencies of the focus node in all its contexts. In our notation, the

value of the test can be determined as:

$$D^{CS} = \sum_{p \in \mathcal{P}_f} \frac{(\text{freq}(c > f \rightarrow p) - \text{exp}(c > f \rightarrow p))^2}{\text{exp}(c > f \rightarrow p)} \quad (7.3)$$

where  $\mathcal{P}_f$  is the set of productions of node  $f$ , and  $\text{exp}(c > f \rightarrow p)$  is the expected frequency of a focus  $f$  having production  $p$  in context  $c$ , judged by the overall co-occurrence frequencies of  $c > f$  and  $f \rightarrow p$  in the corpus:

$$\text{exp}(c > f \rightarrow p) = \frac{\text{freq}(c > f)}{\text{freq}(f)} \text{freq}(f \rightarrow p)$$

We next apply  $D^{CS}$  straightforwardly to TüBa-D/Z. The focus node in context with the highest  $\chi^2$  value is what we call the *most deviant distribution*. It is NX below EN-ADD, which occurs 1944 times.<sup>5</sup> The  $\chi^2$  value is 36 275, which means that the null hypothesis that the distribution of EN-ADD > NX corresponds to the distribution of NX over the whole corpus can be rejected at the 100 % level. Such a 100 % level does not make sense, and is caused by low frequencies that fall below minimal requirements of the  $\chi^2$  test: no  $\text{exp}(c > f \rightarrow p)$  should be lower than one, and at most 20 % should be lower than 5; also required are  $\text{freq}(c > f \rightarrow p) \geq 10$ , and  $|\mathcal{P}_f| \geq 2$ .<sup>6</sup> In our case, there are 122 productions, and 90 of them are expected less than once, and 108 are expected less than five times, corresponding to 90 % of all types. There are 60 more  $c > f$  candidates that can be interpreted to have such an overly high significance. Still, it seems plausible to subclassify NX whenever it occurs below EN-ADD from a linguistic point of view judged by the largest contributor to the overall  $\chi^2$  test result, which is EN-ADD > NX  $\rightarrow$  NE NE. A constituent consisting of two words with proper name tags is much more frequently a complex named entity than a plain noun phrase. Their  $\text{exp}(c > f \rightarrow p) = 49.95$  and  $\text{freq}(c > f \rightarrow p) = 1201$ , so that this configuration alone makes up 73 % of the overall  $\chi^2$  value.

Other candidates also reach the same overly high confidence level, all in all 88. EN-ADD > NX has been chosen because its  $\chi^2$  value is higher than that of all other candidates. As a third selection criterion after confidence level and absolute test value, those  $c > f$  that occur with fewer productions are preferred, confidence level and  $\chi^2$  being equal. We motivate this criterion by our search for specialised (i.e. rather infrequent) consistent usage of a general

<sup>5</sup>Results reported in this section have been obtained on the tune training set described in chapter 5. Only  $\text{freq}(c > f \rightarrow p) \geq 10$  are considered, and edge labels are ignored.

<sup>6</sup>Minimal requirements seem to be given in the literature mostly as rules of thumb. Ours come from Rinne (1997), p. 552.

node label. Another  $c > f$  with similarly high values is  $D^{CS}(\text{VROOT} > \text{VXINF}||\text{VXINF}) = 5396$ , i.e. an infinite verb which is not attached to any constituent in a sentence, vs. an infinite verb in general. It derives the high  $\chi^2$  from eight productions occurring rarely ( $5 \times 1$ ,  $1 \times 2$ , and  $1 \times 3$  times). All these occurrences represent errors in language productions (e.g. *Er hofft, "daß dieses denkwürdige Ereignis irgendwann völlige Normalität sein sein wird"*. ‘He hopes "that this memorable event will be be normal someday"’; our emphasis), or verbs that cannot be attached easily to the rest of the sentence (Gehört: *Fidel-Bastro-Festival. ‘Overheard: Fidel-Bastro-Festival.’*; our emphasis). These usages are rare, but they occur still much less often than expected. This behaviour rather points to unusual, or erroneous language use, which can be exploited for detecting errors (chapter 10), but it does not seem to be useful for improving parsing performance, which should rather focus on more frequent events.

### 7.3.2 $\chi^2$ Merging Infrequent Classes

The  $\chi^2$  test can be applied to a modified data set to overcome the problem of low marginal frequencies. Standard test instructions allow classes to be regrouped by instructions determined independently of the observed events. We do not follow this requirement closely and simply merge the smallest classes iteratively, until all production frequencies comply with the requirements given in the previous section. If the requirements are not met when only two classes remain, then merging of classes fails. Classes are merged so that

1. if some  $\text{freq}(c > f \rightarrow p) < 10$  exist, then the two least frequent classes with least contribution to  $\chi^2$  are merged,
2. else, all  $\text{freq}(c > f \rightarrow p) = 0$  are merged with the least frequent remaining production that least contributes to  $\chi^2$ ,
3. else if any  $\text{exp}(c > f \rightarrow p) < 1$ , or too many  $\text{exp}(c > f \rightarrow p) < 5$ , then the two productions occurring most often with least contribution to  $\chi^2$  are merged,

as long as more than two classes remain. This can be written as equation 7.4, where the  $\mathcal{P}_f^R$  is the set of productions after the minimal number of production regroupings that satisfy the  $\chi^2$  test requirements on minimal frequencies:

$$D^{CSR} = \sum_{p \in \mathcal{P}_f^R} \frac{(\text{freq}(c > f \rightarrow p) - \text{exp}(c > f \rightarrow p))^2}{\text{exp}(c > f \rightarrow p)} \quad (7.4)$$

This production regrouping heuristics tries to keep high contributors to overall  $\chi^2$ , but it is neither certain that the resulting regrouping is optimal in some sense, nor does it guarantee that any regrouping is found at all. No regrouping is found e.g. for the best candidate of  $D^{CS}$ , EN-ADD > NX, where all productions but NE NE NN, FM FM and \$( EN-ADD NX \$( are grouped into a single class. These three productions are all expected less than once, so that only part (3) of the heuristic applies, which will not result in at least two classes satisfying the test requirements. The other candidate of  $D^{CS}$ , VROOT > VXINF, can never comply with the requirements, because there are less than 20 occurrences, and at least two classes with ten events each are required. Last, but not least, merging productions as outlined above is not statistically sound.

A technical drawback of  $D^{CSR}$  is that for each combination of  $c > f$ , a potentially high number of low-frequency events has to be merged, which is inefficient. An even more important drawback is that a certain context that frequently co-occurs with only one type of production, but with no other production, will never be chosen.  $D^{CSR}$  chooses MF > NX as most deviant candidate for renaming from our TüBa-D/Z data set, i.e. noun phrases in the middle field, with  $D^{CSR}(\text{MF} > \text{NX} \parallel \text{NX}) = 6790$ , covering 11 188 focus nodes. Only five alternative  $c > f$  have been found to comply with our choice of  $\alpha \geq 0.9$  for statistical significance. This means that the null hypothesis, that the distribution in context is the same as the reference distribution, can be rejected at the 90 % level. Noun phrases in the Middle Field consist much more often of personal or reflexive pronouns (PPER, PRF) than expected, so this choice does not run counter to intuition.

### 7.3.3 Kolmogorov-Smirnov Goodness-of-Fit

$D^{CSR}$  is neither statistically sound, nor overly efficient, so that searching for other metrics seems to be in order. Another non-parametric test, which moreover has less demands on minimal frequencies, is the Kolmogorov-Smirnov goodness-of-fit test, which will serve as the third instantiation of our deviance metric,  $D^{KS}$ . This test compares cumulative probability distributions, and judges goodness-of-fit by their largest difference:

$$D^{KS} = \max_{i=1, \dots, |\mathcal{P}_f|} |F_{fc}(p_{<i>}) - F_f(p_{<i>})|, \quad p_{<i>} \in \mathcal{P}_f \quad (7.5)$$

where

$$F_{fc}(p_{<i>}) = \frac{\sum_{j=1}^i \text{freq}(c > f \rightarrow p_{<j>})}{\text{freq}(c > f)}$$

$$F_f(p_{<i>}) = \frac{\sum_{j=1}^i \text{freq}(f \rightarrow p_{<j>})}{\text{freq}(f)}$$

It is not obvious how the productions  $p_{<i>}$  should be sorted, because there is no intrinsic ordering of node labels or sequences of them (i.e. of productions). While mnemonics of node labels partly support lexical ordering, a lexical sort order is less convincing for capturing productions consisting of two or more symbols. The ordering is important because the outcome of the test depends on the order of productions,  $p_{<i>}$ . While there is no inherent order in production types, the lexical order of the node labels that make up the production defines a consistent ordering for a given corpus so that we use it to order  $p_{<i>}$ . Still, it is somewhat unsatisfactory that node labels, which are by definition arbitrary, have an impact on the result of the deviance metric.

The deviance metric is normalised to  $0 \leq D^{KS} \leq 1$ , so that two corpora with exactly the same distribution of  $c > f$ , but of different sizes, yield the same  $D^{KS}$ , as opposed to e.g.  $D^{CS}$ , which increases with sample size. Statistical significance of  $D^{KS}$  values depends on the number of production types  $\mathcal{P}_f$  and  $\mathcal{P}_{fc} = \{p \in \mathcal{P}_f | \text{freq}(c > f \rightarrow p) > 0\}$  according to  $n$ :

$$n = \left\lceil \frac{n_1 n_2}{n_1 + n_2} \right\rceil_{\text{floor}} \quad \text{with} \quad n_1 = |\mathcal{P}_f|, n_2 = |\mathcal{P}_{fc}|$$

The relation is not linear, but lower  $n$  require a strictly higher  $D^{KS}$  to achieve the same statistical significance. Requiring that the smallest distribution with  $|\mathcal{P}_f| = |\mathcal{P}_{fc}| = 1$  is different at the 0.9 level of significance corresponds to  $D^{KS} \geq 0.9$ . The treebank does not contain any matching  $c > f$ .<sup>7</sup>

Values for  $D^{KS}$  that conservatively approximate significance values at a 0.9 level can be computed by  $1.073/\sqrt{n}$ . We define a new deviance metric  $D^{KSS} = 1.073/\sqrt{n} D^{KS}$ . There are 44  $c > f$  tuples in the corpus that have  $D^{KSS} \geq 0.9$ , i.e. for which the assumption that they are drawn from the focus node's distribution can be rejected at the 90% level. The most deviant focus node type is the simplex sentence in the final field (NF > SIMPX), of which 1560 appear in TüBa-D/Z. The third best candidate according to  $D^{KSS}$  is EN-ADD > NX, which has also been proposed by  $D^{CS}$ .

<sup>7</sup>Threshold values taken from Rinne (1997), p. 595.

The Kolmogorov-Smirnov goodness-of-fit test can be computed efficiently and is less susceptible to low-frequency productions in the corpus than plain  $\chi^2$ . There is an overlap with productions that are judged by  $D^{CS}$  to deviate from what is expected from a PCFG prediction (see table 7.2 below). Yet, its outcome is determined by otherwise arbitrary decisions in corpus compilation, because sort order is based on lexical ordering, which is arbitrary given that node labels are nominally, and not ordinally different. While consistent for a single data set, sort order may change when nodes are renamed, which is an undesirable side-effect when using node labels to carry information about context as we do.

### 7.3.4 Kullback-Leibler Divergence

The Kullback-Leibler Divergence is not a statistical test, but an information-based metric that describes the amount of information necessary on top of a given distribution to model a second distribution. It is also known as relative entropy (see section 3.3). Given the distribution of all productions of a focus node over the whole treebank  $prob(f \rightarrow p)$  and the distribution of productions in a certain context  $prob(c > f \rightarrow p)$ , the Kullback-Leibler Divergence is computed as

$$D^{KL} = \sum_{p \in \mathcal{P}_f} prob(c > f \rightarrow p) \log_2 \frac{prob(c > f \rightarrow p)}{prob(f \rightarrow p)} \quad (7.6)$$

We estimate probabilities via maximum-likelihood, so that we have

$$prob(c > f \rightarrow p) = \frac{freq(c > f \rightarrow p)}{freq(c > f)}$$

$$prob(f \rightarrow p) = \frac{freq(f \rightarrow p)}{freq(f)}$$

$D^{KL}(c > f || f)$  measures the average number of bits that we need to encode the probability distribution of focus node  $f$  in context  $c$  starting from the overall distribution of productions of  $f$ . It finds that EN-ADD > SIMPX is most different from NX alone. It turns out that most simplex clauses that make up a named entity are special in that they are enclosed by quotation marks (25 of the overall 32 occurrences found). The four most deviant  $c > f$  are all rather infrequent (see table 7.2 below). Only C > NX occurs somewhat more frequently (769 times).

The  $D^{KL}$  metric normalises occurrence frequencies, yielding log-likelihood ratios as approximations of probability distributions. When there is a focus

node  $f_2$  in context  $c_2$  with  $\text{freq}(c_2 > f_2 \rightarrow p) = n \times \text{freq}(c_1 > f_1 \rightarrow p)$  for all  $p \in \mathcal{P}_f$ , and  $\mathcal{P}_f = \mathcal{P}_{f_1} = \mathcal{P}_{f_2}$ , then  $D^{KL}(c_1 > f_1 \| f_1) = D^{KL}(c_2 > f_2 \| f_2)$ . In other words, nodes occurring more frequently have the same deviance according to  $D^{KL}$  as long as the number of different production types is the same, and each production occurs the same number of times more frequently.

In a parsing experiment, you will expect node types more frequently in novel input data that also occur more frequently in training data. If a misleading node label occurs more frequently, it will also have more impact on parsing performance, so that all else being equal, more frequent nodes are likely to be better candidates for Treebank Refinement. A straightforward way to introduce frequencies into  $D^{KL}$  is to multiply it by its frequency in the treebank:

$$D^{KLF}(c > f \| f) = \text{freq}(c > f) \times D^{KL}(c > f \| f) \quad (7.7)$$

which proposes again EN-ADD > NX as the most deviant  $c > f$ , occurring a much more frequent 1944 times (see table 7.2 below).

Stressing the impact of  $D^{KL}$ , and penalising events that are less frequent than a certain minimal threshold  $f_{min}$ , we finally specify

$$D_{f_{min}}^{KLP}(c > f \| f) = (\text{freq}(c > f) - f_{min}) \times 2^{D^{KL}(c > f \| f)} \quad (7.8)$$

so that the perplexity, and not the entropy, is considered equally important as occurrence frequency.

### 7.3.5 Skew Divergence

The metrics 7.6 to 7.8 are not defined if the domain of the first distribution is not a subset of the domain of the second distribution. This will never happen when a set of productions is compared to its superset as in the case of comparing a node's behaviour in certain contexts with its overall behaviour. A robust metric, however, that allows judging the distance of two different subsets that overlap only partially, can be useful for merging nodes instead of splitting them. Distributions of nodes with similar behaviour may overlap only partially, so that each distribution contains productions not found in the other one (see chapter 9). The Kullback-Leibler Divergence can still be calculated in such circumstances when you define  $0 \log \frac{0}{q} = 0$  and  $p \log \frac{p}{0} = \infty$  (Manning and Schütze, 1999, p. 72). However, when comparing the distance between two distributions, it is not clear whether all distributions with a larger domain than the first distribution should be handled similarly by having  $D(p \| q) = \infty$ . It is more likely appropriate to penalise an incomplete domain heavily, but still differentiate between how much of the domain is

missing, and how similar the two distributions are on where their domains overlap. The Skew Divergence accomplishes this by smoothing the second distribution with the first (Lee, 2001):<sup>8</sup>

$$D_{\alpha}^{SD}(p||q) = \sum_{x \in X} p(x) \log_2 \frac{p(x)}{\alpha q(x) + (1 - \alpha)p(x)}$$

The smoothing factor  $\alpha$  adds a parameter to the metric. We rely on observations that indicate best performance for highest  $\alpha$  (Lee, 2001). An  $\alpha$  near 1 also has the benefit of coming close to the behaviour of the Kullback-Leibler divergence, which is also theoretically appealing. We therefore choose to set  $\alpha = 0.99$  in all our experiments, so that:

$$D^{SD} = \sum_{p \in \mathcal{P}_f} \text{prob}(c > f \rightarrow p) \log_2 \frac{\text{prob}(c > f \rightarrow p)}{0.99 \text{prob}(f \rightarrow p) + 0.01 \text{prob}(c > f \rightarrow p)} \quad (7.9)$$

Note that this metric shows greatest additional benefit over the Kullback-Leibler divergence only for merging nodes. We still prefer it where both could be used, because experiments below show that the metrics behave similarly when we split node labels, and because we think that it is useful to have a constant deviance metric to make experiments easier to compare.

We also define Skew Divergence metrics that are weighted by occurrence frequencies similar to  $D^{KLF}$  and  $D_{f_{min}}^{KLP}$  above:

$$D^{SDF}(c > f||f) = \text{freq}(c > f) \times D^{SD}(c > f||f) \quad (7.10)$$

$$D_{f_{min}}^{SDP}(c > f||f) = (\text{freq}(c > f) - f_{min}) \times 2^{D^{SD}(c > f||f)} \quad (7.11)$$

Table 7.2 shows the five most deviant nodes in context for all metrics, from what would be expected when looking at the focus node labels alone, compared to the distribution of the nodes in a context consisting of their parent's node label. The first column lists the metrics, and the following columns give the first five highest-rated context and focus nodes in decreasing order, and their frequencies. The  $c, f$  node labels are given without  $>$  in each cell for space reasons. For the metrics that take a minimal frequency as a parameter, we set  $f_{min} = 10$ , which then is the minimal frequency for  $c > f$  to be considered at all.

---

<sup>8</sup>The original definition turns the parameters around because it trusts the model more than it trusts the estimated frequencies:  $s_{\alpha}(q, r) = D(r || \alpha q + (1 - \alpha)r)$ . We trust the model less and exchange arguments; the other order led to a slight reduction in parsing performance with the resulting treebank grammar according to Parseval  $F_{lab}$  after changing the most deviant node in context.



$D^{CS}$	EN-ADD NX 1944	C NX 769	VROOT NX 1370	NF NX 159	VF SIMPX 717
$D^{CSR}$	MF NX 11 188	R-SIMPX C 794	VC VXFIN 2307	FKONJ MF 1079	MF ADVX 4354
$D^{KSS}$	NF SIMPX 1560	NX NX 16 086	EN-ADD NX 1944	NX ADJX 7918	VF SIMPX 717
$D^{KL}$	EN-ADD SIMPX 32	R-SIMPX R-SIMPX 18	LV NX 12	C ADVX 24	C NX 769
$D^{SD}$	C ADJX 14	EN-ADD SIMPX 32	NX SIMPX 110	LV NX 12	LV SIMPX 45
$D^{KLF}$	EN-ADD NX 1944	NX NX 16 086	MF NX 11 188	C NX 769	NF SIMPX 1560
$D^{SDF}$	EN-ADD NX 1944	NX NX 16 086	MF NX 11 188	NF SIMPX 1560	C NX 769
$D_{10}^{KLP}$	C NX 769	EN-ADD NX 1944	VF SIMPX 717	VROOT NX 1370	NF SIMPX 1560
$D_{10}^{SDP}$	C NX 769	EN-ADD NX 1944	VF SIMPX 717	NF SIMPX 1560	VROOT NX 1370

Table 7.2: Most Deviant  $c > f$  and their Frequencies for Several Metrics

Except for the unweighted information-based metrics  $D^{KL}$  and  $D^{SD}$ , none of the chosen focus nodes in context is very rare, and many, if not all of them seem to be candidates that could also be predicted to behave differently from a linguistic point of view than judged by the more general node label alone. Discussing only the most relevant examples that are given in the second column, there are two types of named entity (EN-ADD>NX and EN-ADD>SIMPX), which contain more often NE POS tags in the former, or more often simpler, and quoted, clauses in the latter case, as pointed out before. Simplex clauses in the final field will be introduced by a conjunction, or a comma, more often than a SIMPX in general (NF>SIMPX). The complementiser field only allows a very restricted set of constituents, which are moreover quite special in their internal structure (C>NX, C>ADJX). Finally, and maybe least obviously, noun phrases in the middle field contain pronouns much more frequently than noun phrases on average (MF>NX).

The information-based metrics show a high degree of overlap as soon as they are frequency-weighted ( $D^{KLF}$ ,  $D_{f_{min}}^{KLP}$ ,  $D^{SDF}$ ,  $D_{f_{min}}^{SDP}$ ). We take this to indicate that smoothing of the Kullback-Leibler Divergence by Skew Divergence works well, and that the type of weighting by frequency does not overly matter. We also observe that C > NX wins over EN-ADD > NX when the divergence is raised to the exponent, because

$$\begin{aligned} D^{KL}(C > NX || NX) &= 5.31 \\ D^{KL}(EN-ADD > NX || NX) &= 3.62 \end{aligned}$$

so that the former is judged more deviant than the latter by the factor

$$\frac{2^{5.31-3.62}}{1934/759} = 1.27$$

and similarly for  $D^{SD}$ .

There are numerous other ways to measure distances between distributions (Lee, 1999). Only our secondary goal is to find a node label having the most unexpected productions (via a statistical test), or the productions that need most additional information for modelling based on PCFG conditioning (via an information-based metric). Our primary goal is to find node labels that are not optimal for PCFG parsing. We only assume that our secondary objective functions (unexpectedness/divergence) are connected to our primary objective function (parsing performance). We try to assess the benefit of the metrics with respect to parsing performance by observing the impact of only splitting the most deviant node in context and by computing Parseval  $F_{lab}$  (Pev  $F_{lab}$ ) on the tune data set without edge labels, and also the cross-entropy between the PCFG language model and the treebank (*perp*).

Results are given in table 7.3, which shows the annotation performance when obtaining a treebank grammar from a treebank where the top scoring candidate for each metric is assigned a new label. The table is ordered by the  $F$  score, with a plain treebank grammar read off the original treebank serving as the baseline in the last row. Most splits improve performance (rows have

	Pev $F_{lab}$	prec.	recall	$perp$
$D_{f_{min}}^{KLP}, D_{f_{min}}^{SDP}$	84.10 %	85.60 %	82.65 %	13.23
$D^{KSS}$	84.08 %	85.51 %	82.69 %	13.14
$D^{CS}, D^{KLF}, D^{SDF}$	84.07 %	84.97 %	83.19 %	13.33
$D^{CSR}$	83.90 %	85.22 %	82.63 %	13.19
$D^{KL}$	83.47 %	84.96 %	82.04 %	13.49
$D^{SD}$	83.47 %	84.96 %	82.04 %	13.49
unrefined	83.48 %	84.99 %	82.02 %	13.49

Table 7.3: Performance after Renaming Single Most Deviant  $c > f$

been collated where several metrics propose the same split). Some splits yield an absolute increase of 0.5 % Parseval  $F_{lab}$ . On the other hand, the first splits of  $D^{KL}$  and  $D^{SD}$  even harm performance, highlighting the importance of frequent events. Cross-perplexity is accordingly reduced for all but the last transforms.

## 7.4 Iteratively Adding Local Context

Renaming a single focus node type all over the treebank when it occurs in a certain context noticeably increases PCFG performance. Many of the highest-rated candidates presented in the previous section represent occurrences in the corpus where renaming does not alter any of the contexts of the other candidates. For  $D_{10}^{SDP}$ , e.g., three disjunct subsets of NX are proposed (below C, EN-ADD and VROOT), and two disjunct subsets of SIMPX (below VF and NF). It is quite likely that these independent subsets also have independent contribution to performance gains. Renaming all of them would take advantage of this observation. Nonetheless, after the first renaming according to  $D_{10}^{SDP}$ , statistics change, because there is a new node label (formerly NX below C), and the number of occurrences of NX is accordingly reduced. So the only way to properly find the next candidate is to reapply the metric to all

$c > f$  in the modified treebank, and to determine and then change the next top candidate. Maximum deviance is likely to drop, but there is no reason to assume monotonous decline, because the focus node that is changed could well be the context of the subsequent iteration, maybe transporting relevant information from a more distant context into reach of just parent context.

A corresponding example from a treebank would be the percolation of linguistic information up or down a tree. We take a look at the influence that the not-so-immediate context may have on the content of noun phrases. Looking back at figure 7.10 on page 139, we have seen a noun phrase (NX) expanding to a personal pronoun (PPER). This NX belongs to the middle field (MF) of a relative clause (R-SIMPX). Table 7.4 shows that the probability of

Context	any		MF		MF#1	
	abs	rel	abs	rel	abs	rel
PPER	2491	5.3 %	1719	15.4 %	77	11.5 %
other	44 247	94.7 %	9469	84.6 %	594	88.5 %

Table 7.4: Percolation of Features Changes Relative Frequencies

the NX expanding into a PPER does not only depend on the immediate parent MF, which considerably raises the expectation of encountering a pronoun (from 5.3 % to 15.4 %). It also depends on the clause type, which can lower the expected ratio of personal pronouns to 11.5 % when the clause type is encoded in MF#1 as a result from a transformation that changes all MF into MF#1 that satisfy  $R-SIMPX > MF$ . This new node label percolates down the information about the grand-parent of NX to the parent of NX, and lowers the expectation to come across a personal pronoun even when looking at the more narrow parent node context.

Table 7.4 also shows that it cannot be predicted exactly when the process of finding deviant node labels and renaming them will finish, because new relevant context can result from splits. However, splitting will eventually come to an end when we specify a minimal threshold of the deviance metric, because the same split is never performed twice, and there is only a limited number of nodes in the corpus, yielding a finite set of  $(f, c)$  tuples. A single focus node can be split more than once, though, if its context changes after it has been first split. This is the reason why  $D(f > c || f)$  is not necessarily monotonically decreasing. Contexts may change, and may reveal new deviations from the expected distribution of productions.

A synopsis of Treebank Refinement is given in algorithm 1. It consists of three loops. The two inner loops iterate through all  $(c, f)$  type tuples found in the treebank and store the largest value of the deviance metric that has been

**Algorithm 1** Iteratively Relabelling Nodes in Context

---

```

1: repeat
2:    $\hat{d} \leftarrow 0$ 
3:   for all  $f \in \mathcal{F}$  do ▷ Focus node types  $\mathcal{F}$ 
4:     for all  $c \in \mathcal{C}_f$  do ▷ Context types  $\mathcal{C}_f$  of  $f$ 
5:        $d \leftarrow D(c > f \| f)$  ▷ Deviance of focus in context
6:       if  $d > \hat{d}$  then
7:          $(\hat{d}, \hat{f}, \hat{c}) \leftarrow (d, f, c)$ 
8:       end if
9:     end for
10:  end for
11:  if  $\hat{d} \geq$  minimum deviance threshold then
12:    RENAMEFOCUSINCONTEXT( $\hat{f}, \hat{c}$ )
13:  end if
14: until  $\hat{d} <$  minimum deviance threshold

```

---

observed, along with the corresponding  $(c, f)$ . If it is above a given threshold, the corresponding focus nodes are renamed in the treebank and the inner loops are entered once more. The subroutine RENAMEFOCUSINCONTEXT( $f, c$ ) changes all occurrences of  $f$  that occur in context  $c$  to one single new unique node label per iteration of the outer loop.<sup>9</sup> In case of ties, and if  $D(c > f \| f)$  is a weighted value, then the unweighted value is also compared, and the larger unweighted value is preferred. Examples are  $D^{CS}$  and  $D^{CSR}$ , where the weighted value is the significance level of the statistical test, and the underlying value is the  $\chi^2$  value. For all information-based metrics that are weighted by frequency, the underlying value is the divergence ( $D^{SD}$  or  $D^{KL}$ ), and the final value is the weighted divergence. Finally, for  $D^{KSS}$  the underlying value is the corresponding value given by  $D^{KS}$ , and the final value is  $1.073/\sqrt{n} \times D^{KS}$  to arrive at a value resembling a statistical confidence. The complexity of the algorithm depends on the number of focus nodes  $|\mathcal{F}|$ , the number of contexts per focus node  $|\{(c, f) | c \in \mathcal{C}_f, f \in \mathcal{F}\}|$ , on the complexity of determining  $D(c > f \| f)$  (usually  $|\mathcal{P}_f|$ ), and on the minimum weight threshold. Except for the threshold, all factors may increase after each iteration.

Applying algorithm 1 to the tune data set, and using  $D_{10}^{SDP}$  yields changes in performance per iteration as shown in figure 7.13. The shape of the curve is similar to a curve produced by evaluating only topological field labels (figure 7.14, as presented in Ule and Veenstra, 2004), hinting at a general

---

<sup>9</sup>We will always append a hash (#) and the number of the current iteration to the existing node label to create a unique new node label.

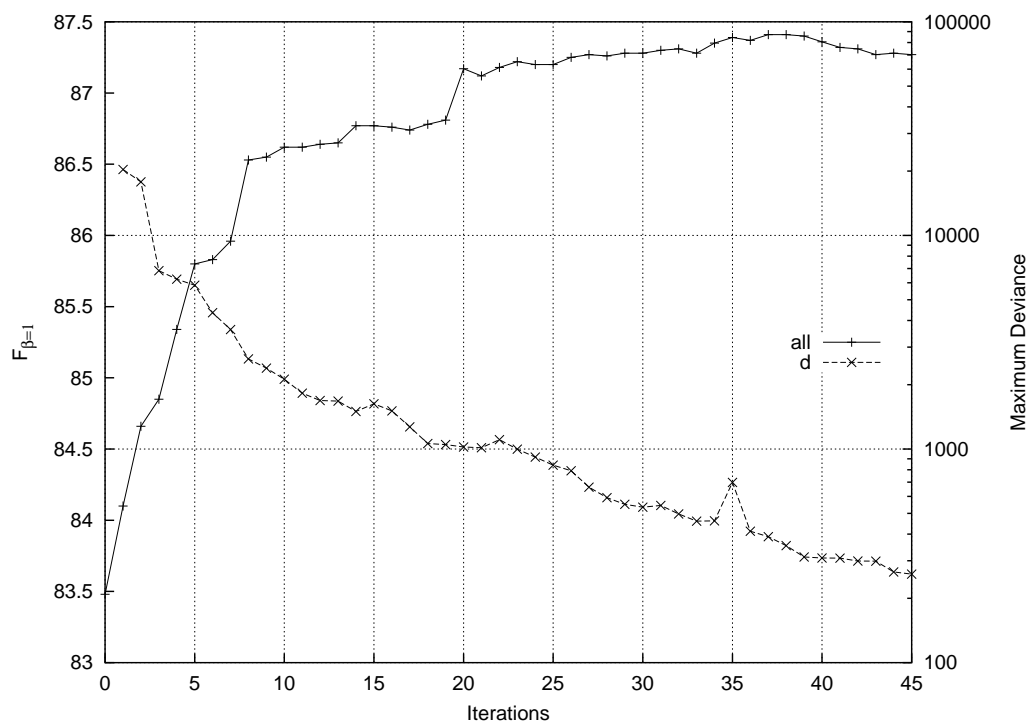


Figure 7.13: All Node Labels:  $F$  and  $\hat{d}$  by Iteration for  $D_{10}^{SDP}$

characteristic of the iterative behaviour of Treebank Refinement.

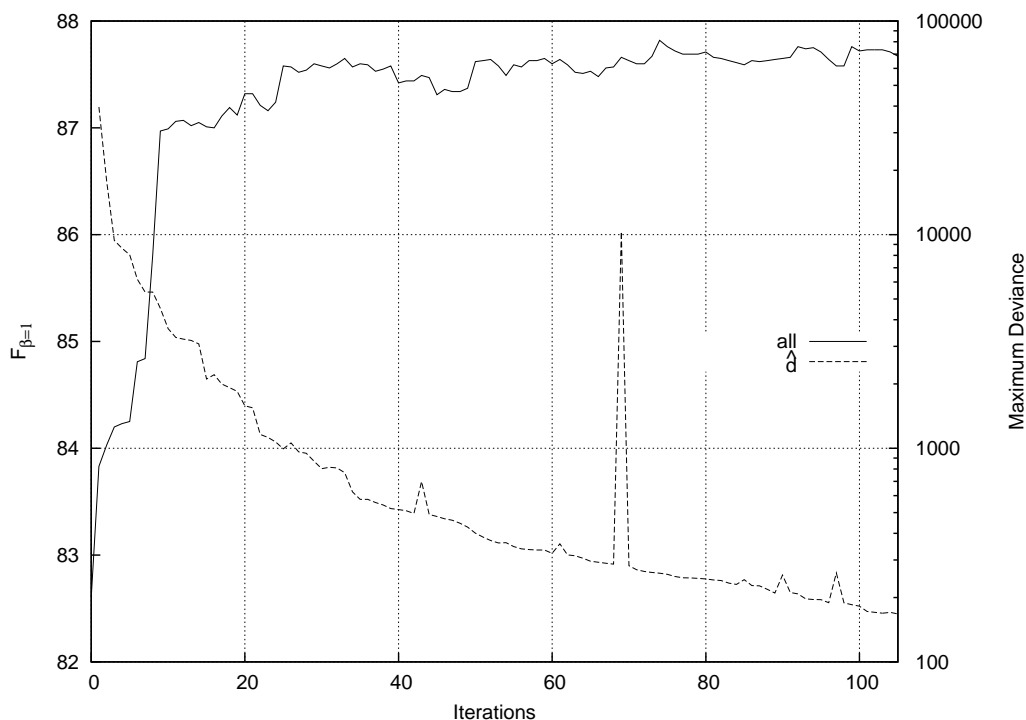


Figure 7.14: Topological Fields:  $F$  and  $\hat{d}$  by Iteration for  $D_{10}^{SDP}$

We will also use the iterative behaviour to reflect on the appropriateness of the deviance metrics, because choosing the best metric is not easy according to table 7.3 alone. The metric  $D^{CS}$  does not seem to be reliable given its failure to handle low-frequency events. The remaining three metrics perform similarly. The  $D^{CSR}$  metric performs best, yet it is very inefficient and theoretically not satisfactory.  $D^{KSS}$  can be computed more efficiently, but depends on the otherwise arbitrary lexical order of node labels.  $D^{SD}$  and likewise  $D^{KL}$  do not increase performance after the first split, and seem to concentrate too heavily on low-frequency events. Given that  $D^{SDF}$  and  $D_{f_{min}}^{SDP}$  seem to approximate their Kullback-Leibler counterparts quite well leaves them as most interesting candidates. The choice between weighting frequency and cross-entropy alike, or emphasising cross-entropy, is what remains. On the one hand,  $D^{SDF}$  performs worse than its counterpart  $D_{f_{min}}^{SDP}$  after the first iteration. On the other hand, the best candidate of  $D_{f_{min}}^{SDP}$  is the second-best of  $D_{f_{min}}^{SDP}$ , making it likely that it will be selected in  $D^{SDF}$ 's next iteration. Table 7.3 does not reveal the performance of repeated applications of the metric so that we repeat it for the more promising metrics

	$\hat{d} \geq$	iterations	Pev $F_{lab}$	prec.	recall	$perp$
$D^{KSS}$	1.073	50	87.32 %	87.70 %	86.94 %	10.94
$D^{SDF}$	7.97	50	87.29 %	87.62 %	86.96 %	10.82
$D_{10}^{SDP}$	250	45	87.27 %	87.70 %	86.86 %	10.93
$D^{CSR}$	0.9	24	84.67 %	85.92 %	83.45 %	13.05
base	n/a	0	83.48 %	84.99 %	82.02 %	13.49

Table 7.5: Performance after a Maximum 50 Iterations

$D_{fmin}^{SDP}$ ,  $D^{SDF}$ ,  $D^{KSS}$  and  $D^{CSR}$  in table 7.5 for several iterations. The number of iterations is limited to 50, because performance does not seem to change drastically any more at least for  $D_{10}^{SDP}$  during the last iterations in figure 7.13.

The curves shown in figure 7.13 and 7.14 suggest that high performance is reached rather early at least with  $D_{10}^{SDP}$ . In order to test this behaviour for the other metrics, we determine their performance for a single reduced number of 38 iterations, corresponding to best performance in figure 7.13. We test performance only of  $D^{KSS}$  and  $D^{SDF}$ , given the overall poor results of  $D^{CSR}$  in table 7.5. Results are given in table 7.6. Stopping earlier achieves

	Pev $F_{lab}$	prec.	recall	$perp$
$D^{KSS}$	87.30	87.70	86.90	11.06
$D^{SDF}$	87.31	87.75	86.87	10.87
$D_{10}^{SDP}$	87.41	87.86	86.97	11.00

Table 7.6: Performance after 38 Iterations

a better performance for  $D_{10}^{SDP}$  and  $D^{SDF}$ , but not for  $D^{KSS}$ , which seems to indicate an ability to perform relevant splits early and thus minimising the changes in the treebank and consequently the size of the derived grammar. These experiments are admittedly quite crude, and stopping after 38 iterations is a biased towards  $D_{10}^{SDP}$ , being the only metric for which dependence on the number of iterations has been sufficiently evaluated. We nonetheless think that the combination of a good  $F$  score and low cross-entropy after rather few changes to the treebank justifies the choice of  $D_{10}^{SDP}$  as our metric for all remaining experiments. We also choose the corresponding stopping condition of  $\hat{d} \geq 350$  for all our remaining experiments.



## 7.5 Unconditionally Spreading Parent Information

A very successful yet simple approach to transforming a treebank for alleviating independence assumptions has been proposed by Johnson (1998). Without looking at the distribution of nodes, he proposes to give each label in a treebank the information about the parent category for all nodes at once. He shows that this transformation can considerably improve parsing accuracy of a derived plain PCFG treebank grammar. Adding context to all nodes by adding parent labels to node labels everywhere and at once would resemble an unconditional split of all nodes in a single iteration in Treebank Refinement. This kind of splitting would only be possible to obtain with algorithm 1 if all children were found before their parents. This is highly unlikely, and only possible if for any  $c > f$ ,  $c$  never equals  $f$ . The simple parent encoding (JP) performs surprisingly well in comparison to Treebank Refinement (TR) as can be seen in table 7.7. The main advantage of Tree-

Trial	base	TR	JP
Iteration	0	38	n/a
Pev $F_{lab}$	83.48	87.41	87.30
<i>perp</i>	13.49	11.00	10.94
<i>failed</i>	0	0	1
# rules	3432	4652	5731

Table 7.7: Unconditional Parenting and Selective Parenting

bank Refinement over Johnson’s parent encoding seems to be that similar performance can be achieved with fewer changes in the grammar and that the changes are individually motivated. We will show in the next chapter that the difference in performance widens, and fewer rules and higher Parseval  $F_{lab}$  are also accompanied by lower perplexity when we extend the notion of context beyond parent nodes to, e.g., parents and grandparents, or when using it with contexts below focus nodes.

## 7.6 Conclusion

The famous context-freeness of context-free grammars is a limitation that does not exclude all kinds of contexts from being modelled correctly. There is only an upper bound on the number of different types of contexts that a CFG can capture, so that it seems to be feasible to model a substantial part

of the limited number of different types of contexts observed in a treebank when the set of non-terminals is extended accordingly.

Probabilities that are given via the frequencies with which a certain kind of context is observed in a treebank can be very useful to decide which context should be included first into the context-free grammar. We have introduced an objective function that compares the behaviour of a node type judged by its label alone and its behaviour when more context is available for conditioning together with the node label. This objective function compares the PCFG way to assign probabilities with a model that assumes less strong independence between a context of a node and its expansion probability.

At the heart of this function is a deviance metric that can be instantiated in several ways. We have explored statistical tests and information-based metrics and experimentally determined the most promising metric. It is based on the Skew Divergence, which is a smoothed version of the relative entropy between a probabilistic model and a given distribution. It is an information-based metric which shows best improvements for PCFG parsing when additionally weighted by the frequency of the node type we examine. We finally show that multiple applications of the objective function iteratively increase parsing performance and lift it on par with results achieved by a simpler well-known transformation while the size of the grammar is increased considerably less, and the individual changes can serve as a characterisation of problematic node labels.

# Chapter 8

## Extended Local Context

Extending the scope of contextual information that is considered for conditioning expansion probabilities is paramount to improve PCFG parsing performance. This chapter tries to outline which context might be relevant for inclusion (section 8.1), and extends the notion of context introduced in the previous chapter accordingly (section 8.2). Context that includes lexical items poses specific problems (section 8.3) and can help disambiguating especially recursive and complex syntactic structure (sections 8.4 and 8.5).

### 8.1 Relevant Context

The goal of parsing natural language is to assign structure to utterances. Structure groups part of an utterance and relates it to other parts of it, so that it ultimately relates words to other words. Constituent-based analyses, which generate hierarchical, or tree structures, assume that certain groups of words, or constituents, have a special status, in that they act as a single entity. In dependency analyses of syntax, the relations are always established directly between words. Both can be related. It has been shown that for TüBa-D/Z a dependency analysis can be derived from constituent structure, because a head is specified for each constituent, marking the word that prominently represents it (see section 3.2). Constituent analyses, on the other hand, can also be approximated by dependency analyses, with respect to the relations that constituents specify between words, so that both ways to describe the syntactic structure of utterances bear close resemblance (Charniak and Carroll, 1992).

When assigning syntactic structure with a PCFG, the words are connected via a number of nodes and edges that are given by the assigned parse tree with the corresponding constituent structure. Heads of constituents are

rarely separated by just one node, including those words that are assumed to represent the major elements of sentences, such as verb, the objects, and the subject. Figure 8.1 shows the original constituent analysis and the derived dependency analysis of an example sentence from TüBa-D/Z. The depen-

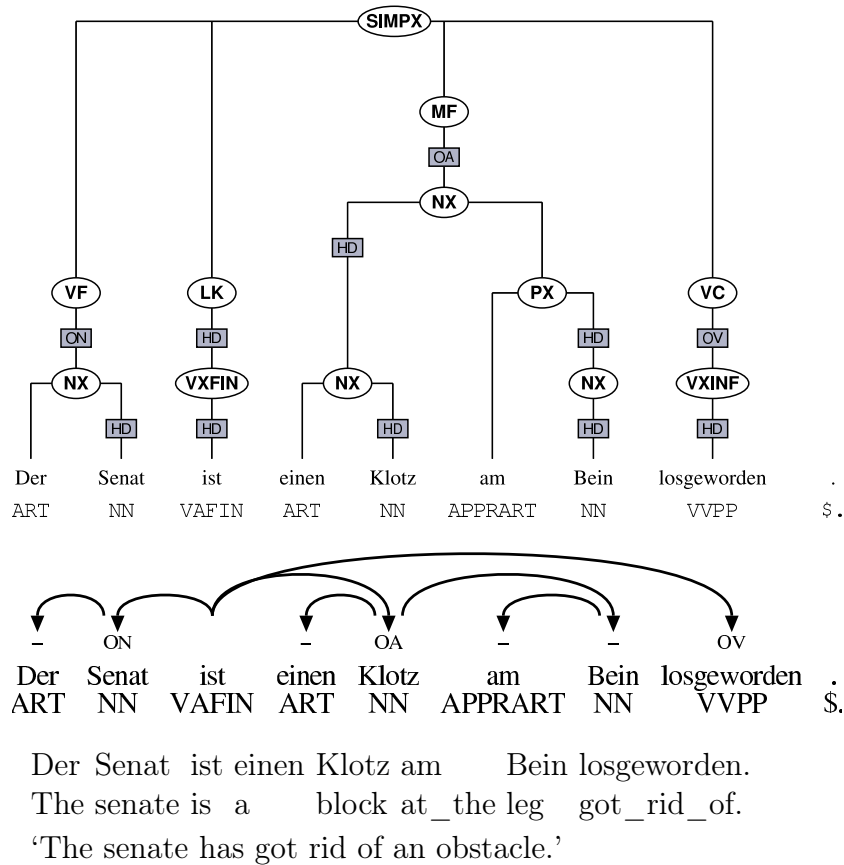


Figure 8.1: Constituent and Dependency Analyses

dependency arcs between the verbal parts and the parts of the accusative object connect the lexical items that are commonly assumed to disambiguate the attachment of the prepositional phrase. In the example sentence, these are the items that include the lexical head of the verbal phrase *loswerden*, the head of the accusative object, *Klotz*, and the head of the item to attach, *Bein*. These three lexical items are connected via three arcs in the dependency analysis, and via seven nodes (*losgeworden* – *Bein*), or four nodes (*Klotz* – *Bein*) in the constituent analysis.<sup>1</sup>

<sup>1</sup>This dependency representation assumes that the inflected verb is the head of the sentence, and that the prepositional phrase has a nominal head (see also section 3.2).

In the example, the attachment is decided below the MF node in the constituent model. Perfect context for the node MF to decide whether to expand it to PX NX, or just to NX, would include the lexical head of the sentence, and thus all the nodes and edges necessary to reach it, following the dependency arcs: SIMPX–VC–VXINF–*losgeworden*. Perfect context, that is, for a probabilistic model that assumes  $P(X \rightarrow Y|context)$  instead of  $P(X \rightarrow Y|X)$ , where *context* includes all information relevant for the expansion of X. For English, experiments have shown that lexical items can help deciding PP-attachment problems (Hindle and Rooth, 1993), and indeed, highest performing parsers adopt the notion of head-lexicalisation (e.g. Collins, 1999). These parsers also achieve high performance on the German negra corpus when slightly modified (Dubey and Keller, 2003).

The algorithm for inclusion of context into node labels presented in the previous chapter is highly unlikely, if not unable, to detect such a context consisting of SIMPX–VC–VXINF–*losgeworden* in order to condition the expansion of MF on it, by renaming node labels so that the probability is determined as  $P(MF \rightarrow NX|SIMPX–VC–VXINF–losgeworden)$ . First, the percolation of a feature using parent context requires that structural information above the focus node changes systematically with structural information below it. If both items are linked via their lowest common ancestor, like MF and *losgeworden* via SIMPX in the example sentence, then the connection cannot be discovered, because there is no node label higher in the tree that changes systematically with both items. Second, no lexical information has been used yet, rendering it impossible in a language like German, where constituent order is rather free, to tell apart structural differences that are linked to case or to open-class lexical items like *losgeworden* in the example. Third, all children of a focus node are related as a whole to their parents, because they always make up an atomic *production*. We will extend the notion of context in this chapter to loosen these restrictions: We will allow information to percolate up, as well as down a tree; we will consider lexical information; finally, we will allow information to cross several nodes at once.

It seems that the ideal probabilistic context should be selected from the set of all paths, or even from all branching paths (i.e. subtrees), of the syntactic analysis of a sentence. There is evidence, however, that parsing performance can be improved considerably by restricting the search to more local contexts. The results of Klein and Manning (2003b) relate directly to the performance of early lexicalised parsers mentioned above (Hindle and Rooth, 1993), and outperform them. They use lexical elements, but rather than looking for open-class words, they concentrate on closed-class words. Performing just closed-class lexicalisation circumvents the problem of estimating probabilities for rare or unseen events, which should usually not be

considered to be zero. Handling these events is a major problem of parsers, and is normally handled by a combination of smoothing and backing-off, i.e. by allocating and assigning part of the overall probability mass to unseen events, or by resorting to simpler estimates (e.g. of word classes instead of word forms) when exact estimates are not available (Manning and Schütze, 1999, p. 196ff. and 217ff.). Collins (1999) backs off bilexical frequencies by using co-occurrence frequencies of one lexical item and a node label instead. Comparing the original distribution of bilexical co-occurrence patterns with the back-off distribution containing only a single lexical item amounts to comparing the effect that two lexical items have on syntactic structure with the impact of a single item. It turns out that the two distributions are remarkably similar (Bikel, 2004).

It thus seems that individual lexical items on their own can have considerable impact on syntactic structure. They seem to trigger certain analyses, and in absence of complete information, these preference patterns can improve parsing. It is in this sense that we seek to find relevant context: there are relevant local connections between words and structure (as pointed out by Bikel, 2004), and between local subgraphs that only cover small parts of whole syntactic analyses (as pointed out by Klein and Manning, 2003b). Rather than trying to incorporate dependencies between distant elements of a sentence that would likely overload focus nodes in the sense of the connecting nodes presented in the previous chapter, we try to model relevant local phenomena, resulting in a model that selects the best parse more often. The more distant connections may ultimately be handled separately by more powerful methods than with plain PCFGs, once the coarser preferences are incorporated into a treebank grammar transformed by the methods discussed here.

## 8.2 Extending Context

We stick to the same question as presented in the previous chapter: Do *productions* of a *focus* node change consistently with *context*? The *focus* remains a single node in the syntactic description in what follows. What we will extend now is the *context*. Instead of being restricted to the parent node, it also may include an arbitrary number of nodes on a path through the tree in a single direction, beginning from the node adjacent to the focus node. What can also change is the direction, so that in the same direction as before, instead of the parent context we have an *ancestor* context. In the opposite direction, this amounts to a *descendant* context. The *production* changes accordingly: it is still the sequence of direct children for ancestor context, but

it is the parent node for descendant context. A final extension pertains to the subclassification of categories that is often available for syntactic categories, for which there is not always an obvious level of detail: the *attribute* context allows selecting some optional information that is available for nodes. All these extensions to the original notion of parent context are discussed in turn below.

### 8.2.1 Ancestor Context

A path that connects the focus node and one or more nodes into the direction of the root of the tree is an extended context that we call *ancestor* context. It tests dependence on more distant nodes than parent context. While parent context is generally able to percolate information up or down a tree, it is dependent on directly dominating material. If that is not found, e.g. because the parent of the focus node is always the same, but the grand-parent changes consistently with the productions of the focus node, then ancestor context can, and parent context cannot detect this connection. Ancestor context still looks strictly for dependencies in a top-down direction. Similarly to parent context, it is thus not capable of determining the relevance of another node at the same level or below the focus node directly, so that e.g. dependence on terminal nodes cannot be tested directly.

The deviance metrics can be used as before with the only difference that there are more contexts available for each focus node. In addition to the parent context, which remains part of ancestor context, a single new atomic context is added for each node in the treebank that dominates the parent (i.e. for each grandparent), or dominates the grandparent, and so on.

In figure 8.2 there are two new contexts 2-1 and 3-2-1 on top of the parent context 1 for the focus node, which has a bold outline at the centre of the figure. The number of contexts per node thus increases with the average depth of trees. In TüBa-D/Z, there are 944 227 ancestor contexts in the tune train set, as opposed to 217 221 parent contexts.

Once a relevant context is found that consists of more than a single node, also the corresponding context nodes are assigned new unique labels, with the exception of the most distant contextual node, because a single connection can already be modelled by PCFGs. If the focus node X is split in e.g. context 3-2-1 in figure 8.2, then the context nodes 1 and 2 are changed in addition to the focus node. By convention we rename nodes by affixing a hash and the number of the current iteration. Assuming the context was the most deviant in the first iteration, then the unique suffix #1 marking this split is added to the focus node and to all but the most distant context nodes. We choose a different suffix for renaming nodes of the context in order to avoid incidental

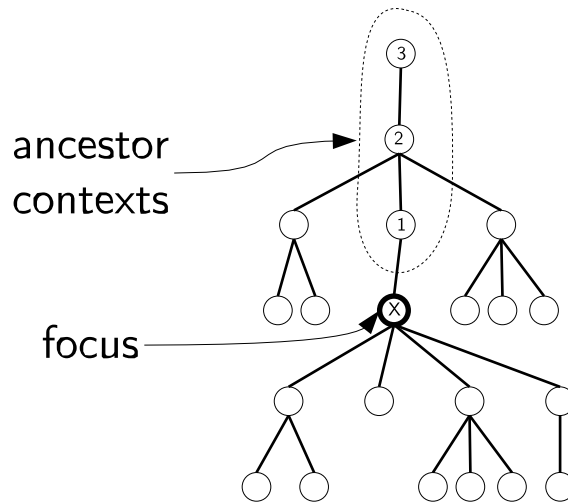


Figure 8.2: Ancestor Contexts

identity between the name of contextual and focus nodes. We consequently choose the suffix  $\tilde{\#}1$  for the context nodes, i.e. the suffix of the focus node with an initial tilde  $\tilde{\phantom{x}}$ , turning the context into  $3-2\tilde{\#}1-1\tilde{\#}1$  (see figure 8.3).

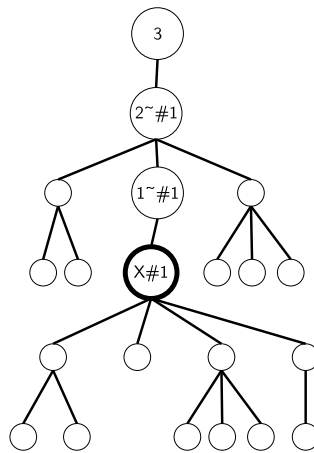


Figure 8.3: Renamed Focus and Ancestor Context Nodes

It is obvious that many more ancestor than parent contexts are considered for each focus node. Larger contexts can carry information over larger distances, but they also introduce more changes into the treebank, resulting in more different rule types and less data for estimation of each rule's probability. Larger contexts are therefore only used if they yield higher deviances than smaller contexts. If the same largest deviance is found for some smaller



context, then the  $(c, f)$  with the smaller context is strictly preferred. Algorithm 1 is changed accordingly to algorithm 2, introducing a comparison of context size. The subroutine `SMALLERTHAN` $(x, y)$  compares contexts  $x$  and

---

**Algorithm 2** Iteratively Relabelling Nodes in Extended Contexts
 

---

```

1: repeat
2:    $\hat{d} \leftarrow 0$ 
3:   for all  $f \in \mathcal{F}$  do                                     ▷ Focus node types  $\mathcal{F}$ 
4:     for all  $c \in \mathcal{C}_f$  do                                   ▷ Context types  $\mathcal{C}_f$  of  $f$ 
5:        $d \leftarrow D(c > f || f)$                              ▷ Deviance of focus in context
6:       if  $d > \hat{d} \vee (d = \hat{d} \wedge \text{SMALLERTHAN}(c, \hat{c}))$  then
7:          $(\hat{d}, \hat{f}, \hat{c}) \leftarrow (d, f, c)$ 
8:       end if
9:     end for
10:  end for
11:  if  $\hat{d} \geq$  minimum deviance threshold then
12:    RENAMEFOCUSINCONTEXT $(\hat{f}, \hat{c})$ 
13:  end if
14: until  $\hat{d} <$  minimum deviance threshold

```

---

$y$ , returning `True` if the first context  $x$  is smaller than the second context  $y$ , where the number of ancestor nodes in the context is compared.

Changing the treebank iteratively using ancestor context of any depth changes results from parent context as shown in table 8.1. The number of contexts in parent context equals the number of nodes in the training set, because each node has a non-terminal node as a parent, or if unattached, the special parent `VROOT`. The row *Contexts Tokens* gives the fixed number of individual contexts, and not the number of different types of contexts, which changes after each iteration.

Ancestor context consisting of more than just the parent of a node is used only four times. Still, Parseval  $F_{lab}$  slightly increases and cross-perplexity slightly drops. Among the first ten iterations, there is a single case where more than the parent node is selected as context: `C-PX` in the third iteration, covering 148 occurrences of `NX`. Figure 8.4 shows schematically the third split with larger context (focus node `NX#3`) together with the first split (focus `NX#1`), that both describe the special behaviour of noun phrases in the `C` field. The context is above the focus nodes for both splits, depicted by the continuous arrows pointing from the context into the direction of the focus node. `NX` is first split in parent context `C`, and receives the new unique name `NX#1`. `NX#3` behaves unexpectedly in context `C-PX`, so that the context is partly renamed as well in order to bring this distributional

Context	parent	ancestor
Parseval $F$	87.41	87.62
<i>perp</i>	11.00	10.80
Iterations	38	42
parent contexts	38	38
other ancestor contexts	0	4
Focus tokens renamed	46 379	50 597
Context tokens renamed	0	6298
Grammar Rules	4652	4748
Failed	0	0
Context Tokens	217 221	944 227

Table 8.1: Ancestor Context vs. Parent Context

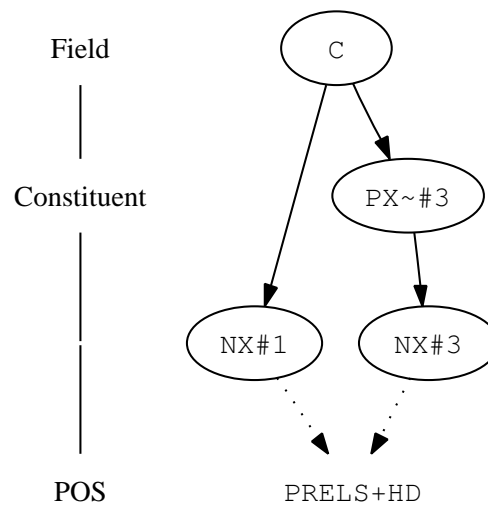


Figure 8.4: Ancestor Context connecting C Field and NX

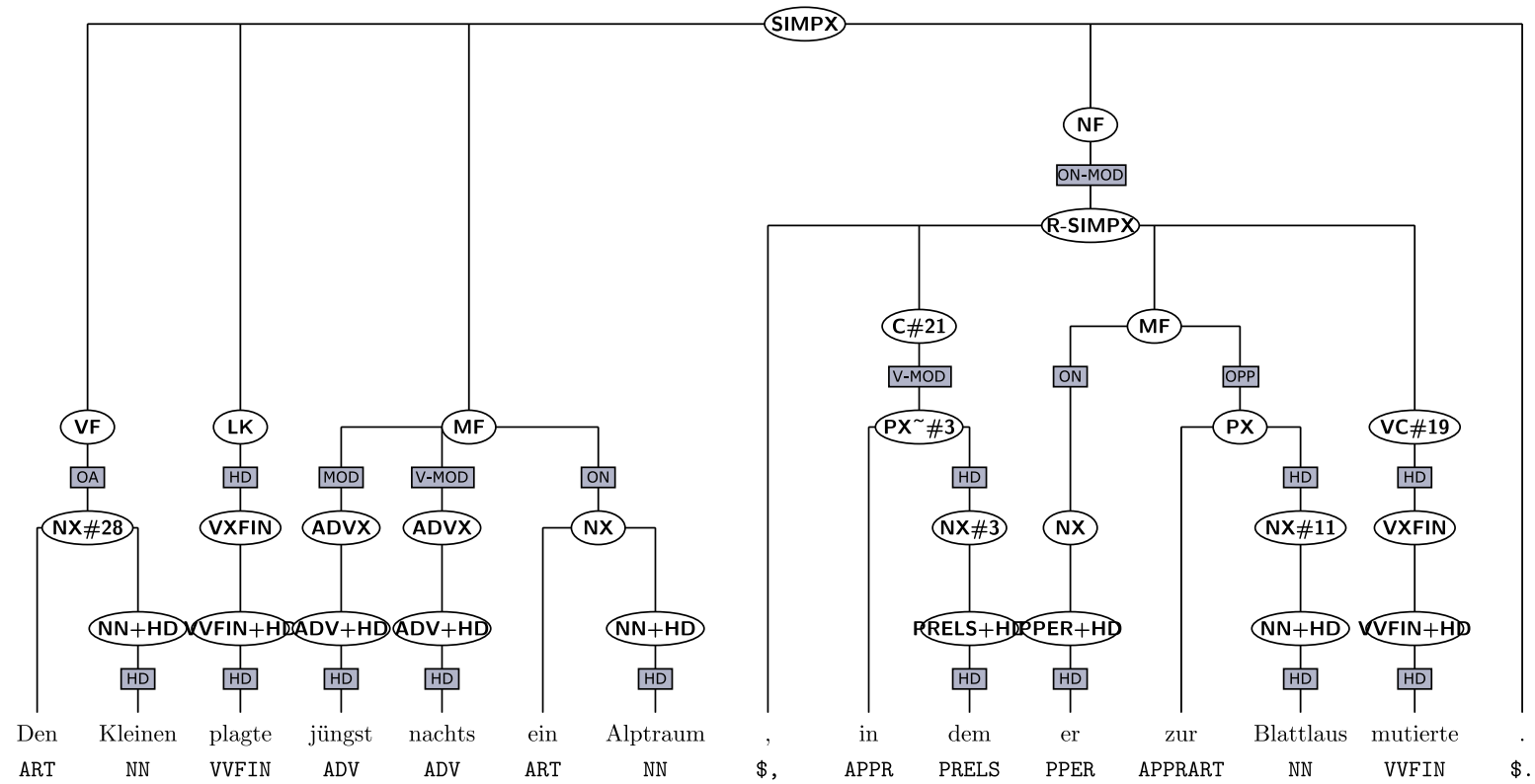
preferences into reach of a probabilistic context-free grammar (represented by the new unique name  $PX\sim\#3$ ). The dotted arrows give the most unexpected productions of the focus nodes in the wider context (both times the head-marked substituting relative pronoun PRELS).<sup>2</sup> Figure 8.5 shows a sentence with all ancestor context splits applied, and shows an example for  $NX\#3$ .<sup>3</sup> It also shows that the noun phrase in the C field and the finite verb phrase in the verbal complex (VXFIN) are connected via splits to the relative clause label (R-SIMPX), resulting in the new node labels  $C\#21$  and  $VC\#19$ . These changes are given step by step in figure 8.6, where (a) corresponds to the original annotation, (b) to the annotation after iteration 3, (c) after iteration 19, and finally (d) after iteration 21.  $C\#21$  represents the complementiser field and  $VC\#19$  the verbal complex of relative clauses (as opposed to simplex clauses). Both make up a chain of uniquely named nodes connecting the parts of the relative clause that have characteristic behaviour while originally labelled like nodes also found in other sentence types. More specifically,  $VC\#19$  has mostly finite verbal productions, and  $C\#21$  holds mostly noun phrases or prepositional phrases with relative pronouns. The latter are found in  $NX$  that have been turned into  $NX\#3$ . All three together only occur in relative clauses, because  $VC\#19$  and  $C\#21$  have been renamed every time they appear below R-SIMPX. Together, they describe the distinct behaviour of the verbal bracket in relative clauses.

Parent context breaks up the split of prepositional phrases in the complementiser field into two not quite equivalent splits, in contrast to the single split that creates new focus and context nodes  $PX\sim\#3$  and  $NX\#3$  for all  $C-PX\sim\#3 > NX\#3$  in ancestor context. In parent context, noun phrases inside prepositional phrases are first split into a separate class, and only much later, prepositional phrases in the complementiser field are split into a class of their own (covering 43 occurrences). Noun phrases inside prepositional phrases in the complementiser field consists almost exclusively of relative pronouns in combinations such as *an dem* ('at which'), *auf der* ('on which'), *über den* ('over which'), making them highly distinct from other  $PX$ . In ancestor context, 148  $PX$  in the C field are split as part of the context of  $NX\#3$ . While parent context also finds some regularities for those  $PX$ , ancestor context seems to be able to detect them more directly.

Another example that shows how ancestor context with more than parent context leaves a trace through the tree, so that larger context can become available to context-free productions, is given in figures 8.7 and 8.8. The

<sup>2</sup>Please note that we interpret PRELS+HD as POS tags here, although they are modelled as additional nonterminals by the PCFG (see section 5.2.4).

<sup>3</sup>TüBa-D/Z sentence 8818. The initial parenthesis *Schlimmer noch*: ('Even worse:') has been omitted from the original sentence.



Den Kleinen plagte jüngst nachts ein Alptraum, in dem er zur Blattlaus mutierte.  
 The little\_boy troubled recently at\_night a nightmare, in which he into\_the plant\_ouse turned.  
 ‘The little boy was troubled by a nightmare recently, in which he turned into a plant louse.’

Figure 8.5: Example Sentence split with Ancestor Context (cf. figure 8.4)

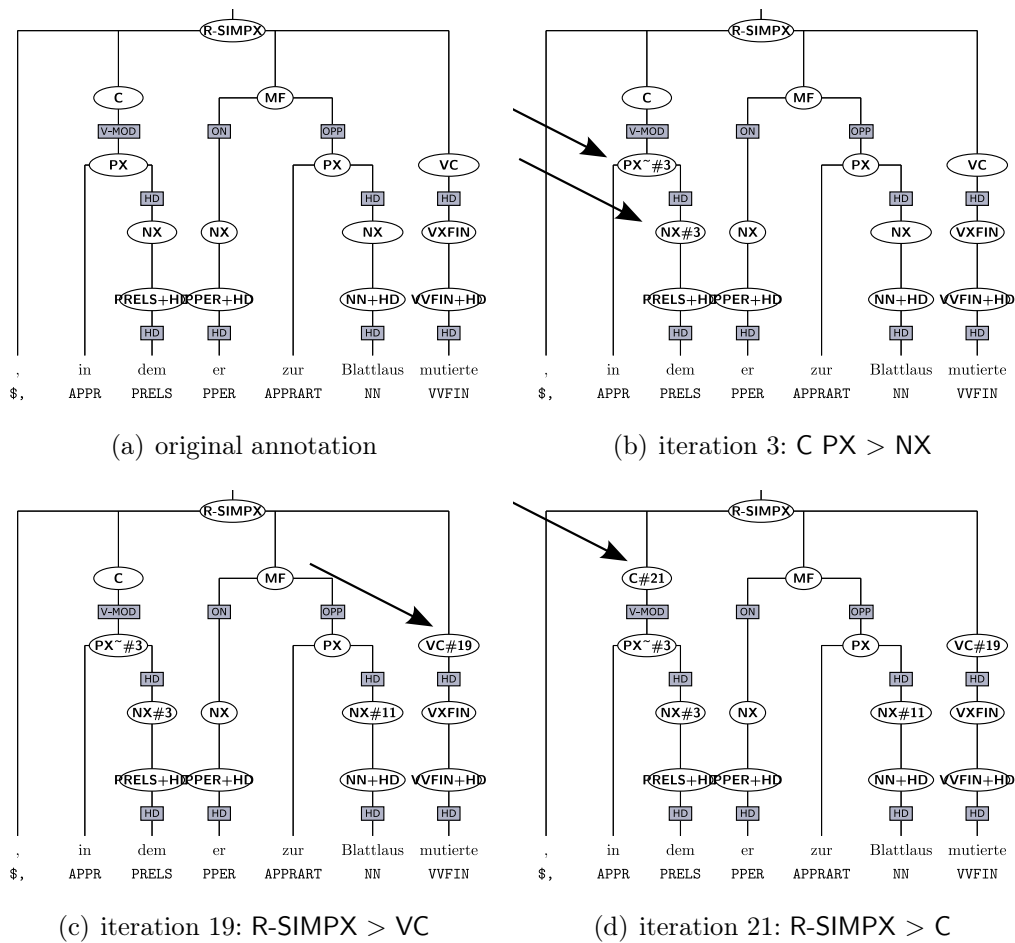


Figure 8.6: Individual Changes in C/VC of figure 8.5

figures list all splits that involve simplex clauses (SIMPX) and the verbal complex (VC), culminating in a description of the verbal complex in simplex clauses embedded in the final field (NF). We would like to stress that figures 8.4, 8.7 and 8.8 do not necessarily show tree fragments that occur as a whole in the treebank. They rather show focus nodes that do not occur without their contexts in the transformed treebank, e.g. VXFIN#42 in figure 8.7 does not occur without VC~#40, but VC~#40 may well occur without VXFIN#42, because due to the iterative behaviour of Treebank Refinement, only a later split can depend on a former split, and not vice versa. At the

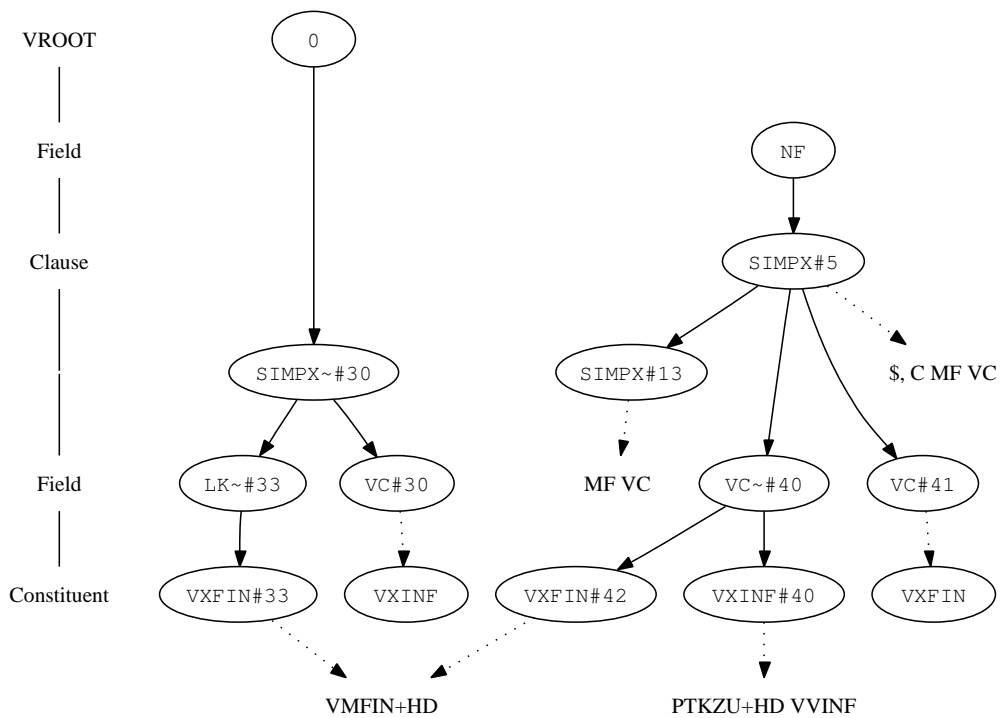


Figure 8.7: Ancestor Context Splits of Simplex Clause/Verbal Bracket Labels

same time, these splits characterise the behaviour of the verbal bracket in simplex clauses. In the order of iterations, SIMPX#5 in figure 8.7 captures distinct behaviour of simplex clauses in the final field (mainly the verb-last order of verbal brackets, and a leading comma as expressed in \$, C MF VC). Later, split #40 marks the non-finite part of these subclauses, which often contains combinations of the verb particle *zu* and a non-finite full verb (PTKZU+HD VVINF). Given that the right sentence bracket contains such a non-finite verbal part, the finite verbal part of the verbal complex (if any) consists mostly of a modal or auxiliary verb (split #42). Figure 8.9 shows

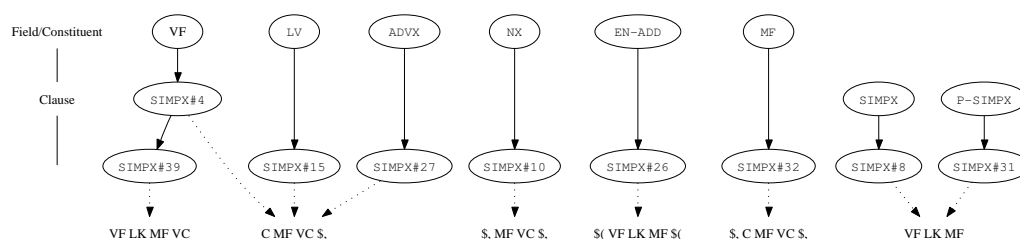


Figure 8.8: Ancestor Context Splits of Simplex Clause Labels

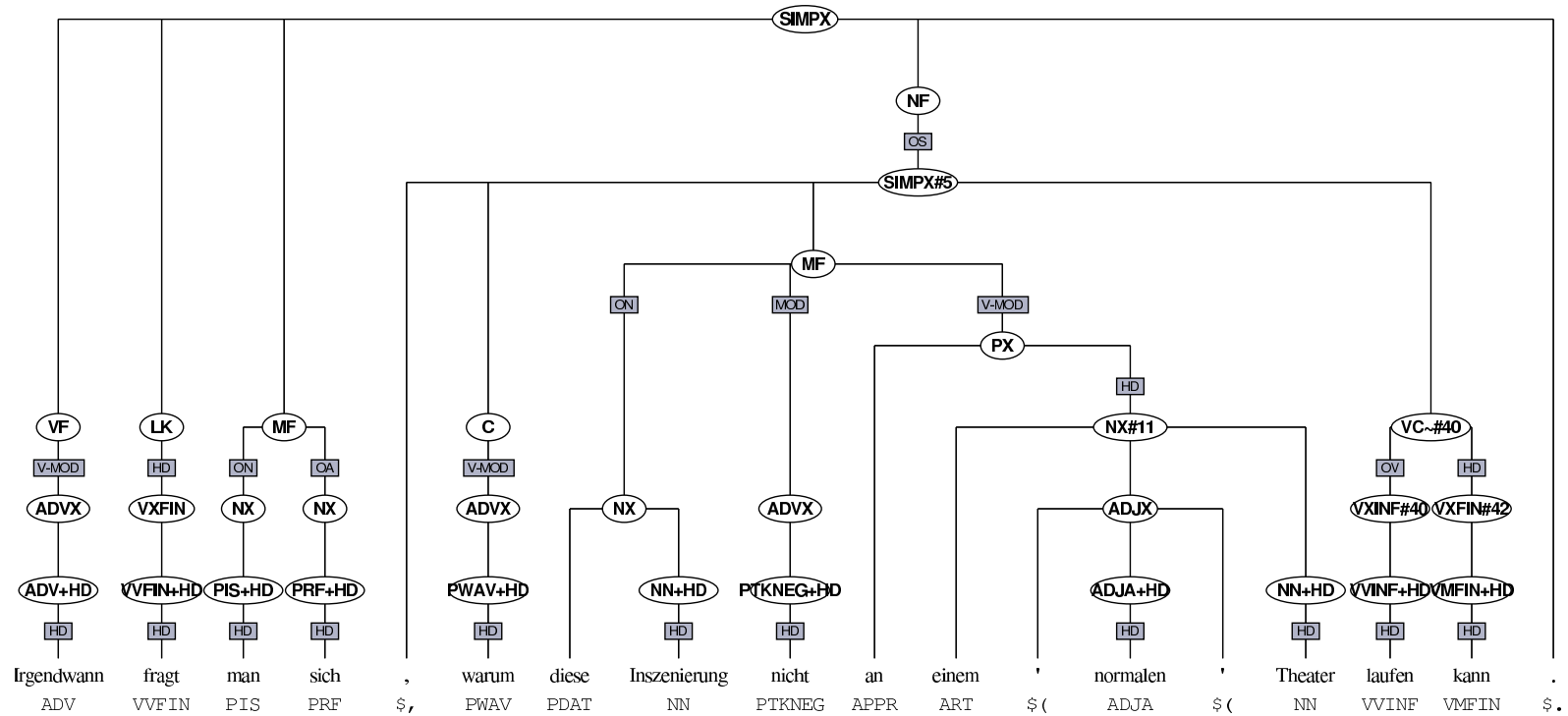
an example sentence where splits #40 and #42 connect sisters in the verbal complex of a subclause embedded in the final field. Last, if the verbal complex does not contain a non-finite part (all of which have been renamed to VXINF#40 before), then it usually consists of a single finite verb (VC#41).

A similar effect can be observed between the left and right sentence brackets of non-embedded clauses, shown in the left half of figure 8.7. Split #30 mainly adds the information to the VC label that the verbal complex is part of a main clause, but because the context is also changed to 0-SIMPX~#30, this information becomes available to a subsequent split of the left verbal bracket. The left bracket in this case will contain mostly modal or auxiliary verbs (split #33). In the end, the nodes shown in figure 8.7 form a unit that models local context spanning a set of connected nodes in a syntactic tree. The nodes are connected to form the local context via distributional information that has percolated up and down the tree via splitting of node labels in the way just outlined.

Preferences for certain field sequences in certain contexts can be observed for the class of clausal nodes in figure 8.8. The most probable sequences of split nodes that are given at the bottom of the figure are quite similar for all simplex clauses in resumptive constructions (LV), directly in the initial field (VF), and in adverbial phrases (ADVX; splits #4, #15 and #27). Coordinated simplex clauses show different behaviour in the initial field than elsewhere, in that the clause is ended more often by a verbal complex below VF (SIMPX#39 and SIMPX#8).

### 8.2.2 Descendant Context

Descendant context changes the direction of looking at focus nodes in context. So far, the production has always consisted of the nodes *below* the focus node, making anything above the focus node a candidate for context. Descendant context refers to a context below the focus node, so that the *production* is the single node above the focus node, i.e. the focus node's parent node. There is



Irgendwann fragt man sich, warum diese Inszenierung nicht an einem 'normalen' Theater laufen kann.  
 Sometimes asks one oneself, why this staging not in a 'normal' theatre run can.  
 'At some stage you ask yourself why this kind of production cannot be staged in a 'normal' theatre.'

Figure 8.9: Example Sentence Split with Ancestor Context (cf. figure 8.7)



not a single node that is definitely part of a descendant context, as was the parent node for ancestor context. Instead, any child can be the initial part of descendant context, and any descendant of this child can extend the context to form an alternative larger context.<sup>4</sup>

A node in a syntactic tree links information between its exterior (the outside part) and its interior (the inside part; see figure 7.8 on page 136). Parent and ancestor context link some information in the exterior of a tree to the root of the interior. It may be useful to determine the most relevant inside information, too, that needs to be linked to the outside. Descendant context tries to do this and assesses the information below a node that would be relevant to predict its upward production (the parent). Figure 8.10 shows

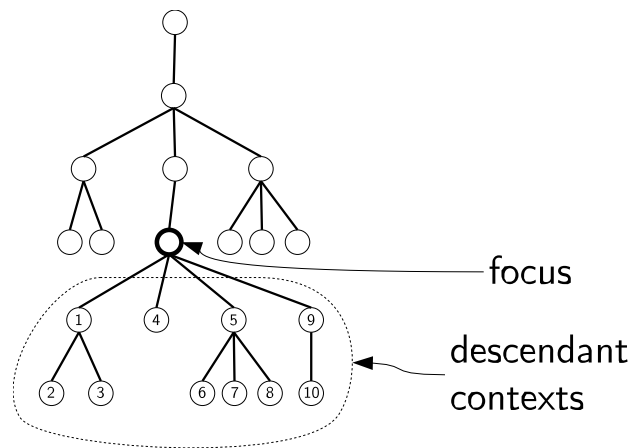


Figure 8.10: Descendant Context

a tree with a focus node in the centre. The descendants of the focus node are all numbered. Each number represents one context, because each numbered node is connected via a unique sequence of nodes to the focus node. The resulting set of descendant contexts thus is  $\{ 1, 2-1, 3-1, 4, 5, 6-5, 7-5, 8-5, 9, 10-9 \}$ . We order the nodes in the name of contexts always left to right from most distant to closest to focus node.

The number of contexts per focus node is larger for descendant context than for ancestor context. There are as many descendant contexts for a focus node as there are descendants to the node, and this number grows with the depth and the width of a tree. Table 8.2 shows that descendant context adds roughly six times the number of parent contexts for TüBa-D/Z. Parseval  $F_{lab}$  increases over parent context, and also over ancestor context.

<sup>4</sup>The parent *node* always refers to the node dominating the focus node. Parent *context* is an ancestor context of length one. In descendant context, the parent node corresponds to the *production*.

Context	parent	descendant
Pev $F$	87.41	87.74
<i>perp</i>	11.00	10.67
Iterations	38	40
parent contexts	38	0
child contexts	0	36
other descendant contexts	0	4
Focus tokens renamed	46 379	56 760
Context tokens renamed	0	1033
Grammar Rules	4652	5438
Failed	0	1
Contexts	217 221	1 390 136

Table 8.2: Descendant Context vs. Parent Context

Linking a parent to a characteristic child as with ancestor context mainly captures dominance relations: whether a noun phrase (NX) that has an article as one of its children is dominated by a field node or by another NX in TüBa-D/Z amounts to saying that the noun phrase is either complex or simple. When there is an NX parent, nothing is said about whether the original NX with the article is the head of the complex noun phrase or not – this information is encoded separately in edge labels. A field label parent provides additional information about the linear position of the noun phrase in the sentence. Most of the time, adjective phrases containing a predicative adjective (descendant context POS tag ADJD ) appear in the middle field (production MF), much more often than in the initial field (VF). They are also part of complex constituents (below ADJX, PX, ADVX), but the most unexpected position of an ADJX that has an ADJD child is in the middle field when contrasted with the behaviour of all ADJX. Many other splits for descendant context reveal similar discrimination of linear position, e.g. personal pronouns (PPER) have the same preference for the middle field, while interrogative or demonstrative pronouns (PWS, PDS) prefer the initial field. While this information may be rather obvious for the reader, it is hidden for PCFGs in the node label NX that is the same for all these pronouns.

Descendant context seems to be more useful for annotation schemes that use rather flat trees to represent syntactic structure. In flat trees, there are more types of productions for ancestor context, and consequently productions tend to be less frequent, which is a challenge for metrics comparing distributions. Descendant context, on the other hand, singles out one of the sister nodes of such a production and makes it the context for the descendant

context production. Figure 8.11 schematically shows a subtree containing six sister nodes, and compares ancestor and descendant context of depth one. The focus node is given as a hollow circle/rectangle. Descendant context

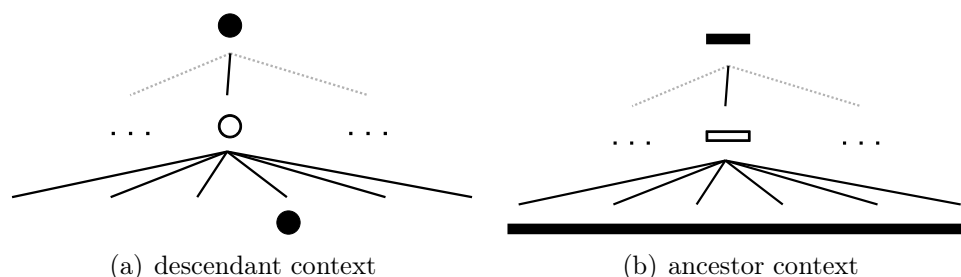


Figure 8.11: Ancestor and Descendant Context in a Flat Tree

(depicted as circles) is able to connect a single child node (the context) with a single parent node (the production; figure 8.11(a)). Ancestor context, on the other hand, always connects the parent (as part of the context) with all of the focus node’s children (the production; depicted as rectangles in figure 8.11(b)). The more children a node tends to have, or the flatter the annotation scheme tends to be, the more this difference will matter. Table 8.2 shows that the number of different context types with size greater one stays the same between ancestor and descendant context (cf. table 8.1), while the number of occurrences is lower for descendant context (row “Context tokens renamed”). This indicates that descendant context is more selective. On the other hand, ancestor context can take the relations between sister nodes into account, which are then all linked in total to their grandparent. Performance for descendant context slightly increases over ancestor context.

Descendant and ancestor context consider mutually exclusive sets of contexts for inclusion into node labels, so that joining both sets of  $(c, f)$  candidates and selecting the most deviant focus in context is an obvious consequence. Table 8.3 shows that applying both kinds of context improves performance slightly indeed. The context size never exceeds two nodes (the rows “other ancestor/descendant context” always refer to exactly two nodes of context). Interestingly, the number of newly introduced rules is lower than using descendant context alone, which may partially explain the better performance.

### 8.2.3 Attribute Context

Attribute context is an extension to ancestor and descendant context that also considers optional information. So far, we have always used node labels as

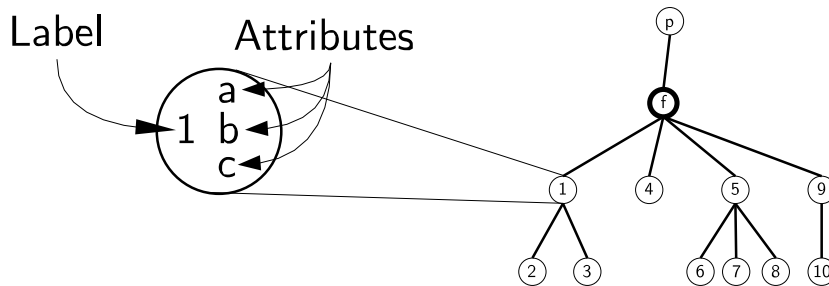
Context	parent	ancestor & descendant
Pev $F_{lab}$	87.41	87.91
<i>perp</i>	11.00	10.64
Iterations	38	42
parent contexts	38	30
other ancestor contexts	0	2
child contexts	0	9
other descendant contexts	0	1
Focus tokens renamed	46 379	50 930
Context tokens renamed	0	1033
Grammar Rules	4652	5142
Failed	0	0
Contexts	217 221	2 384 363

Table 8.3: Combined Descendant and Ancestor Context vs. Parent Context

input that represent immutably given information. Evaluation of the output has always been performed on the set of non-terminal and terminal node labels as given in TüBa-D/Z. The edge labels that are present in TüBa-D/Z but that have not been considered so far represent additional information that may help assigning structure, even when edge labels are not subject to evaluation themselves. Attribute context tries to make this additional information available to the  $(c, f, p)$  view of probabilistic local context. For each node in the syntactic tree, one or more *attributes* can be specified, and the values of these attributes are added to the sequence of nodes that make up ancestor or descendant context. For each sequence of ancestor or descendant nodes, a new context is added for each permutation of attribute values for the sequence of nodes starting with the most distant context node and continuing to the focus node  $(c_n, \dots, c_2, c_1, f)$ , resulting in new contexts  $(a(c_n), c_n, a(c_{n-1}), \dots, a(c_2), c_2, a(c_1), c_1, a(f))$ , where  $a(\cdot)$  represents one of the attributes of a node, including the empty attribute, and  $c_1$  represents a child of the focus node in descendant context, or the parent of the focus node in ancestor context (see figure 8.12).<sup>5</sup>  $c_2, \dots, c_n$  consequently refer to the more distant context nodes, so that the set of contexts for a node  $f$  is given as  $\mathcal{C}_f = \{(a(c_n), c_n, a(c_{n-1}), \dots, a(c_2), c_2, a(c_1), c_1, a(f))\}$ .

As an example, the middle field of figure 8.1 is shown again in figure 8.13(a). Considering only all descendant contexts for the top NX node,

<sup>5</sup>The idea of using optional context evolved from personal communication with Helmut Schmid.



additional descendant attribute contexts for focus  $f$ :

$a-1, b-1, c-1, 2-a-1, 2-b-1, 2-c-1, 3-a-1, 3-b-1, 3-c-1$

Figure 8.12: Attribute and Descendant Context

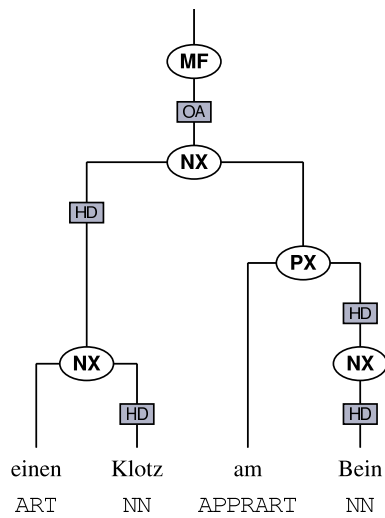
and stopping at POS tags, all descendant contexts without attribute context are given in figure 8.13(b). The additional contexts found with attribute context are given in figure 8.13(c). It is immediately obvious that much more contextual information is considered for inclusion into node labels with attribute context.

The increased number of contexts should pay off through new insights into deviant distributions of productions. Looking at examples from treebank data, you find e.g. that  $NX$  focus nodes prefer production  $MF$  in descendant attribute context  $OA$ , meaning accusative objects are more likely to occur in the middle field, and even more so in context  $PX-OA$ , representing accusative objects modified by a prepositional phrase. All  $NX$  disregarding their function modified by a  $PX$  show yet another behaviour. Table 8.4 shows the relative frequencies of the fields in which noun phrases appear that are acting as accusative objects ( $OA$ ), that are additionally modified by a  $PX$  ( $PX-OA$ ), or that are modified by a  $PX$  disregarding their function ( $PX$ ). The behaviour

Production	Context			
	$OA$	$PX-OA$	$PX$	any
$VF$	6 %	6 %	12 %	9 %
$MF$	91 %	93 %	29 %	24 %
$C$	3 %	1 %	0 %	2 %
others	0 %	0 %	59 %	65 %
absolute	4695	544	3374	46 738

Table 8.4: Distribution of Productions for  $NX$  Descendant Contexts

of all four seems to differ sufficiently to make all the contexts interesting candidates for inclusion into the focus node label.



(a) top NX is focus node

{NX,  
 ART-NX,  
 NN-NX,  
 PX,  
 APPRART-PX,  
 NX-PX,  
 NN-NX-PX}

(b) descendant contexts

{ OA,  
 HD-NX-OA,  
 NX-OA,  
 HD-NX,  
 ART-HD-NX-OA,  
 ART-NX-OA,  
 ART-HD-NX,  
 HD-NN-HD-NX-OA,  
 NN-HD-NX-OA,  
 HD-NN-NX-OA,  
 HD-NN-HD-NX,  
 HD-NN-NX,  
 NN-HD-NX,  
 NN-NX-OA,  
 PX-OA,  
 APPRART-PX-OA,  
 HD-NX-PX-OA,  
 HD-NX-PX,  
 NX-PX-OA,  
 HD-NN-HD-NX-PX-OA,  
 NN-HD-NX-PX-OA,  
 HD-NN-NX-PX-OA,  
 HD-NN-HD-NX-PX,  
 NN-NX-PX-OA,  
 NN-HD-NX-PX,  
 HD-NN-NX-PX }

(c) additional attribute contexts

Figure 8.13: Edge Labels as Attribute Contexts for a Fragment of Figure 8.1

While contexts so far have consisted of information that has been defined to be part of the transformed treebank a priori, attribute context considers information for optional inclusion. Primary candidates for these attributes are information already present in the treebank, when their status with respect to the parsing goal is not fully clear. In TüBa-D/Z there are a number of edge labels that add information to node labels, but it is not obvious whether including this information into node labels will improve parsing performance or not. These edge labels include grammatical functions for constituents embedded in topological fields, or apposition, head, and conjunct labels for edges that connect nodes in complex constituents, or that connect topological fields in field coordinations. Attribute context may test all these edge labels for their influence on the behaviour of the nodes. If they do not influence their behaviour, then they will likely only create a new type of node that behaves just like before, i.e. they will reduce the amount of training data without improving parsing performance. If they influence the behaviour of the nodes, however, they may provide a good motivation to split nodes.

Attributes may also be created with information derived from the treebank, e.g. features of lexical items such as case information, suffixes, or base-forms. As the additional information is only present in the contexts, but the productions still contain only information relevant for evaluation, attribute context can be seen as selecting additional information to predict a given goal, wherever the additional information consistently co-occurs with the goal.

Algorithm 2 (p. 165) does not need to be changed to work with attribute context, because we only need to extend the set of contexts  $\mathcal{C}$  to include attribute contexts. As in ancestor and descendant context before, smaller contexts are strictly preferred to larger contexts. A context is smaller if it contains fewer nodes, or, given it has the same number of nodes, if it has fewer attributes. The function `SMALLERTHAN()` is correspondingly changed to compare the number of descendant contexts. See algorithm 3 for the final definition of this function covering ancestor, descendant and attribute contexts.

The number of contexts in a treebank using attribute context is the number of ancestor or descendant contexts times the size of the set of permutations for all attributes of the nodes in these contexts. Table 8.5 shows the number of contexts that are considered when all edge labels in TüBa-D/Z are added as attributes (row *Contexts*). The number increases only moderately over plain ancestor and descendant context, because not all nodes carry edge information, and never more than one edge label.

The major problem of adding more information to node labels is data sparseness. Adding information to node labels means adding new produc-

**Algorithm 3** Comparing Context Sizes

---

```

1: function SMALLERTHAN( $c_1, c_2$ )
2:    $n_1 \leftarrow$  COUNTNODES( $c_1$ )           ▷ The number of nodes in  $c_1$ 
3:    $n_2 \leftarrow$  COUNTNODES( $c_2$ )           ▷ The number of nodes in  $c_2$ 
4:    $a_1 \leftarrow$  COUNTATTRIBUTES( $c_1$ )     ▷ The number of attributes in  $c_1$ 
5:    $a_2 \leftarrow$  COUNTATTRIBUTES( $c_2$ )     ▷ The number of attributes in  $c_2$ 
6:   if  $n_1 < n_2$  then
7:     return True
8:   else if  $n_1 = n_2 \wedge a_1 < a_2$  then
9:     return True
10:  else
11:    return False
12:  end if
13: end function

```

---

Context	ancest. & desc.	ancest./desc. & attribute
Pev $F_{lab}$	87.91	87.38
<i>perp</i>	10.64	11.14
Iterations	42	76
parent node contexts	30	23
parent attribute contexts	0	3
other ancestor contexts	2	4
child node contexts	9	8
child attribute contexts	0	35
other descendant contexts	1	40
Focus tokens renamed	50 930	85 527
Context tokens renamed	1033	8810
Grammar Rules	5142	8102
Failed	0	1
Contexts	2 384 363	6 287 564

Table 8.5: Ancestor/Descendant Context and Attribute Context



tions, or reducing the amount of data available per class. The added information is useful as long as it helps predicting annotation. Attribute context starts with few information (the non-attribute context) and adds information if it supports predicting annotation, following the paradigm of iteratively fitting node labels to the parsing task and the treebank.

Noun phrases with different grammatical functions show that attributes carry information that is relevant for disambiguating syntactic annotation, which may, however, be too fine-grained if used unconditionally for all nodes. Nominative noun phrases tend to occur more frequently in the initial field, while accusative and dative noun phrases tend to occur in the middle field (see table 8.6). Grammatical functions, however, will be most useful to de-

Field	Grammatical Function			
	ON	OA	OD	OG
VF	39 %	6 %	8 %	0 %
MF	55 %	91 %	91 %	100 %
C	6 %	3 %	0 %	0 %
absolute	9508	4695	695	11

Table 8.6: Preferred Fields of NX with Grammatical Functions

termine structure when they can be anchored in lexical items, which so far is not the case. Attribute context may be restricted to ancestor context, to test whether constituents with certain grammatical functions have distinct types of children. On the other hand, the grammatical function attribute in conjunction with descendant context may help deciding whether a constituent is part of a complex constituent (thus having no grammatical function attribute), or is attached to some field. The result of splitting in individual contexts that all correspond to a single function will be more fine-grained than the broad, and probably more useful, division between a part and the maximal projection of a complex constituent. Instead of deciding whether a constituent is attached to a field or to another constituent, a new subclass will be added for each constituent type in every function, because this is what the attribute contexts capture.

In our tune data set, the splits in figure 8.14 are all triggered by different grammatical functions, but the behaviour of all prepositional phrases is the same – they prefer to appear in the middle field.<sup>6</sup> A single uniform split of all PX with any of these functions at once will probably help improve parsing performance more than the individual splits when assigning grammatical functions is not part of the parsing goal. Merging these nodes could

<sup>6</sup>We show attribute context in rectangular boxes.

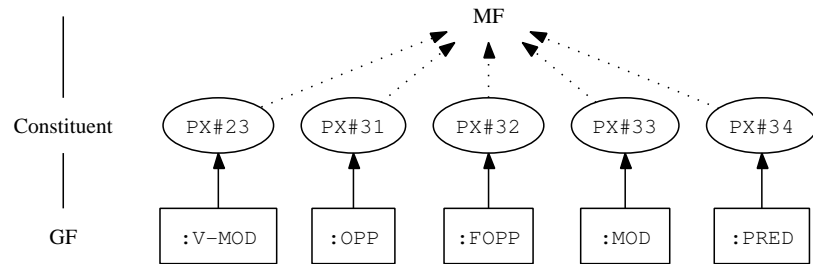


Figure 8.14: Attribute Splits Leading to Identical Productions

be promising, which will be discussed in chapter 9.

Not all splits triggered by attributes carrying edge labels show this degree of uniformity in productions and should therefore be merged. Figure 8.15 shows many splits that are ultimately triggered by a proper name tag NE in a noun phrase NX. Most different edge labels subsequently lead to splits

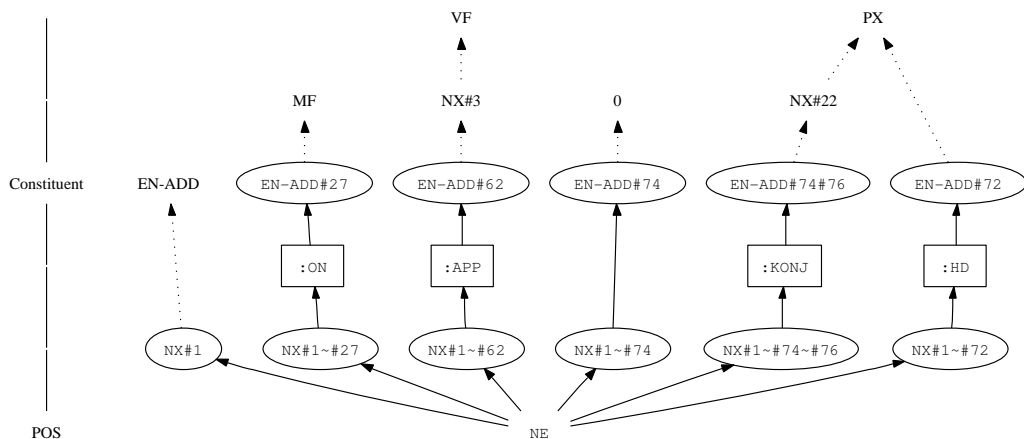


Figure 8.15: Attribute Splits Leading to Different Productions

diversifying EN-ADD, which appears as a subject (ON) mostly in the middle field, and as part of an apposition (APP) mostly in the initial field. Many of the remaining proper names containing an NX#1 are not part of a sentence (EN-ADD#74), or are part of a complex constituent, always depending on the edge label included via attribute context (the two other splits shown).

These very detailed splits of a single node label could hint at a performance of the iterative refinement that has gone past its peak. A curve plotting performance per iteration for including still more contextual information does not show a step decrease, however (figure 8.27 on page 210).

Splits rather seem to add relevant information, but not the ideal type for simply assigning structure without edge labels, as we evaluate it.

## 8.3 Lexical Context

Lexicalisation grants a grammar access to lexical information. Heads of constituents take this information to higher levels in syntactic structure, where the heads considered more important percolate further up a tree, dominating the other heads. Lexicalisation is considered the key to high-performance parsing, because syntactic structure is normally arranged to make the lexical heads of constituents meet in the proper order. Experiments have demonstrated that lexical information can help resolving structural ambiguities (Hindle and Rooth, 1993), and in fact, the best performing parsers to date lexicalise treebanks for parsing (Charniak, 2001; Collins, 1997; Collins & al., 1999; Ratnaparkhi, 1999; Taskar & al., 2004).

Lexicalisation can be added to the context – focus – production view on treebanks straightforwardly. Experiments in the previous chapters have all been performed on POS tags as terminal symbols, but it is easy to extend the fringe of syntactic trees with word forms, promoting POS tags to non-terminals.<sup>7</sup> Lexicalisation via lexical context is thus simply an increase in the level of detail of the POS tag set. The STTS tagset has been widely applied, but whether it provides the best level of abstraction for parsing is not obvious. It is rather likely that at least some information about case may be useful where constituent order is as free as in German. We do not try to show the impact of explicit case information directly here, because German word forms are often ambiguous with respect to case and therefore not trivial to disambiguate morphologically. Instead, we concentrate on frequent word forms, which may also be correlated to certain morphological readings. POS tags that are promoted to nonterminal symbols can serve as focus nodes in addition to being only productions and context in ancestor and descendant context, respectively, so that POS tags, too, can be refined when they appear to be too general. Figure 8.16 shows how POS tags are lifted into a position where they can act as focus nodes, so that the lexical items form the new fringe of the tree. The intended result of applying refinement to the resulting treebank are changes in POS node labels that allow percolation of lexical information up a tree wherever the former POS tag in the position of a nonterminal node label does not represent well the connection between local contextual changes above it and changes in the distribution of the lexical items that it dominates. Note that via a combination of ances-

---

<sup>7</sup>Pre-terminals are not considered separately here.

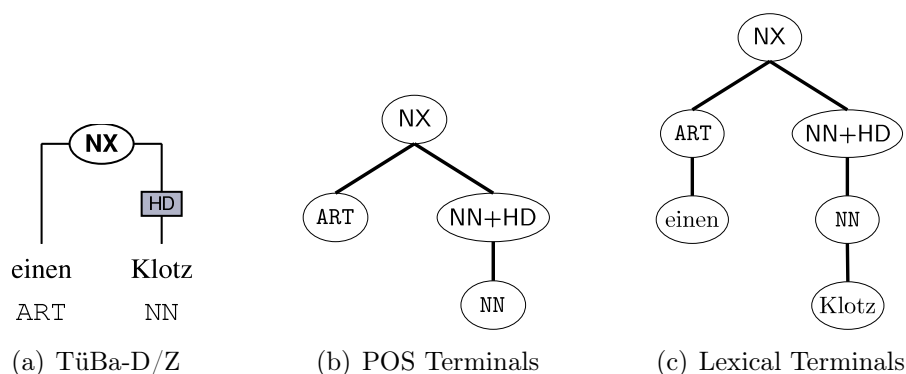


Figure 8.16: Extending Context to cover Lexical Items

tor, descendant, attribute, and lexical context, all overly general node labels modified to improve parsing performance by Schiehlen (2004) and by Klein and Manning (2003b) can also be found by Treebank Refinement, including node labels specialised by partial selection of functional edge labels and POS tag splitting based on high-frequency word forms.

### 8.3.1 Unrestricted Lexical POS Splitting

The difficulty in adding the appropriate amount of lexical information to POS tags is mainly caused by the high number of different word forms. Lexical items are known to have a very skewed frequency distribution, resulting in a high number of low-frequency types that are unlikely to be found frequently enough in new data to deserve individual treatment in order to improve parsing. There are methods for estimating probabilities for these problematic classes, and while we do not deny the usefulness of such approaches, we still stick to plain PCFG parsing nonetheless in order to examine its behaviour when it is unaffected by other methods that re-distribute probability mass. Any smoothing or backing-off being absent from our method of transforming the treebank prior to, and during parsing, there is of course the danger of introducing too many splits that do not generalise well to unseen test data. Table 8.7 indeed shows that straightforwardly applying Treebank Refinement to a treebank, so that words, and not POS, form the fringe of trees performs poorly: Parseval  $F_{lab}$  only reaches 67.5%, and almost 3% of all sentences fail to parse. This result is accompanied by a rather high number of 323 splits. Note that cross-perplexity is not directly comparable, because the vocabulary differs between the experiments.

The nodes that are split in this experiment tend to have high numbers of productions, and they have many productions that appear just once in

Context	ancest. & desc.	ancest./desc. & lexical
Pev $F_{lab}$	87.91	67.49
<i>perp</i>	10.64	118.24 <sup>a</sup>
Iterations	42	323
parent node contexts	30	29
parent attribute contexts	0	0
other ancestor contexts	2	232
maximum ancestors	2	3
child node contexts	9	45
child attribute contexts	0	0
other descendant contexts	1	17
maximum descendants	2	7
Focus tokens renamed	50 930	166 713
Context tokens renamed	1033	288 824
Grammar Rules	5142	14 742
Failed	0	88
Contexts	2 384 363	3 855 945

<sup>a</sup>Unique terminal vocabulary: unknown words in test and train set, and words seen three times or less in train set replaced by special new *unknown* symbol (section 5.2.4).

Table 8.7: Straightforward Lexicalisation via Lexical Context

the contexts that trigger the splits. Being expected even less, they sum up to a high overall deviance, making them top candidates for splitting. Classes containing many word form types, each with low frequency, are typically open word classes, like nouns or adjectives. These classes feature prominently among the lexically triggered splits. Table 8.8 shows the node label types that are split in this experiment with a context or a production covering lexical items, and adjectives and nouns are indeed by far most frequently the subject of a split (column *times split*). They make up for more than half of the overall 323 splits.

times split	label
109	NN
29	ADJA
27	NE
19	VVFIN
12	ART, VVPP
11	ADJD
10	ADV
7	VVINFL
6	ADVX
3	CARD, KON
2	APPR, VXINF
1	APPRART, KOUS, PIS, PPER

Table 8.8: Splits Triggered by Lexical Items for Unrestricted Lexicalisation

Sparsely populated classes are difficult to compare for similarity metrics, and while  $D_{10}^{SDP}$  proved relatively robust in this respect in the experiments presented in the previous chapter, it seems to give a large number of infrequent events too much importance for our purpose now. Note that closed-class POS are split as well, but much less frequently (e.g. ART in row 6). Splits seem to be triggered not by a few high-frequency lexical items (the declared target of our method), but rather by large sets of infrequent words. Accordingly, lexical items are part of the *context* of splits only 20 times, while they occur in *productions* an overwhelming 256 times. Note also that the influence of lexical items seems to extend over just the lowest levels of syntactic structure, because two splits of constituent labels are triggered by lexical items as well (ADVX and VXINF). Thus there seems to be a connection between lexical items and syntactic structure that is accessible to refinement despite these poor initial results.

A closer look at the lexically induced splits that are performed due to

a large number of low-frequency productions shows that many of the focus nodes have almost exclusively low-frequency productions. This is mostly true for the frequent open-class POS NN and ADJA, but table 8.9 shows that also other open-class POS tags show this behaviour. The table shows all POS

Focus POS	unique	all	% once
VVPP	1251	1622	77.13
ADJD	1151	1576	73.03
NN	23 643	33 842	69.86
ADJA	4405	6483	67.95
VVFIN	2723	4111	66.24
VVINFL	682	1070	63.74
NE	3928	7440	52.80
CARD	156	355	43.94
ADV	306	1621	18.88
PIS	12	106	11.32
APPR	25	991	2.52
KOUS	5	224	2.23
APPRART	3	213	1.41
PPER	7	668	1.05
ART	8	1484	0.54
KON	1	325	0.31

Table 8.9: Unique Productions of Lexically Induced Splits

tags that have been split due to productions covering lexical items, e.g. VVPP in the initial field that has the productions *Ergänzt*, *Alarmiert*, *Angedeutet*, *Angefangen*, and 25 others, where the first is seen twice, and all others once (the exact context is VF-VC-VXINF-VVPP+HD). The second column gives the number of unique types of productions for this focus POS, and the third column gives the number of all types of productions, including those where the production is seen more than once. The last column gives the fraction of unique productions from all productions. What is striking is that lexicalised adverb POS tags (ADV) have a rather low number of unique productions, more similar to the closed POS classes at the lower end of the table than to the open classes at the top. It seems that certain adverbs trigger splits because they frequently co-occur with distinct structure, making them interesting candidates in the current framework.

### 8.3.2 Restricted Lexical POS Splitting

A straightforward approach to excluding harmful classes of POS with many low-frequency productions would be to exclude sets of POS depending on whether their definition suggests that they include open-class words or not. The case of adverbs indicates, however, that there are word forms with POS labels covering mostly open-class words that still could be useful for parsing, because they correlate frequently to syntactic structure that is quite unexpected given the POS tag alone. A different approach to reducing the number of problematic low-frequency productions is to lexicalise a given set of (POS, word form) tuples, which may also include relevant open-class word forms. The selection should be guided by the frequency of the production, i.e. the frequency of the lexical item, and by the relevance of the production for the split, i.e. its contribution to the overall deviance leading to the split. Table 8.10 shows the five biggest contributors to the overall deviance among the productions that lead to splitting ADV the first and second time (splits #36 and #117). It shows that there are a few rather frequent productions

(a) split #36 in context VF-ADVX-ADV+HD

% deviance	obs	exp	production
9	47	6.52	<i>So</i>
6	31	3.36	<i>Da</i>
5	24	2.65	<i>Dann</i>
4	39	13.25	<i>dann</i>
4	20	2.65	<i>Jetzt</i>
687 occurrences in 175 productions			

(b) split #117 in context ADJX#13-ADVX-ADV+HD

% deviance	obs	exp	production
32	58	7.25	<i>so</i>
9	17	2.05	<i>ganz</i>
7	11	0.83	<i>viel</i>
6	11	1.15	<i>sehr</i>
5	7	0.49	<i>etwas</i>
210 occurrences in 61 productions			

Table 8.10: First Two ADV Splits Involving Lexical Items

that are also much more frequent than expected, making them large contributors to the overall deviance. The table also shows that capitalisation seems to be important. Most of the adverbs in split 36 are uppercase, so that the



split seems to be triggered by the fact that adverbs in the initial field are mostly uppercase (see context in table 8.10(a)). The fact that these adverbs are sentence-initial may seem surprising from the perspective of the **ADV** tag alone, but it probably does not contribute much to better parsing, because if the first input symbol for a parser is an adverb, it will likely be put into the initial field (**VF**) anyway. Since *Dann* and *dann* are both selected, there also seems to be a contribution of lexical items proper, underlined by split 117, where all adverbs are lowercase (see table 8.10(b); **ADJX#13** in the split's context prefers to occur in **MF**, **ADJX** or **VF**).

### 8.3.3 Selecting Lexicalised POS Before and During Parsing

On the one hand, the application of Treebank Refinement to lexicalisation should be restricted to splits triggered by mostly high-frequency word forms, i.e. it should be restricted mostly to closed-class POS. On the other hand, selecting closed-class POS a priori based on their definition misses those word forms tagged with an open-class POS that still have *mono-lexical* impact on syntactic structure, and will also fail to handle the uppercase feature, which seems to be rather orthogonal to the lexical item feature. We intend to stress the difference to *bilexical* dependencies between two words (e.g. the heads of phrases for attaching PPs) when using the term *mono-lexical*, which describes the dependence of structure on single lexical items.

Our strategy to solve the problem of selecting lexical items that could have an individual impact on syntactic structure is to ignore case distinctions by normalising case, reapply splitting, and select some splits that have POS tags as focus. We select those (POS, word form) tuples that have been triggered by descendant contexts, because minimal frequency and deviance requirements for splitting a node at all are definitely met for descendant context. Secondly, all ancestor context splits are included where the productions cover lexical items, and production frequency and contribution to deviance meet a certain minimal threshold. In order to allow all possibly interesting information to be considered for subcategorising the POS tags by lexical items, we use attribute context, which has the potential of connecting grammatical function edge labels with lexical items.

Parsing performance is influenced by splitting the treebank, but also by either annotating POS separately before parsing, or as part of assigning syntactic structure. All experiments apart from the one reported in Table 8.7 have used unmodified STTS POS tags as input symbols that completely hide all lexical items. Lexicalisation that is triggered by ancestor context results

in a modified set of POS tags that are selected depending on structure that dominates the tags, i.e. on information not available to the POS tagger. POS tag splits triggered by descendant context including just the lexical item are compatible with POS tagging as a preprocessing step, but only less than ten percent of all splits have such kind of context. The remaining splits cannot be performed on the basis of word forms and POS tags alone, because the POS tags are split lexically in ancestor contexts that depend on the parse, which is obviously not available prior to parsing, so that POS tagging as preprocessing is not an option. The parser can use word forms as input symbols as well as POS tags, so that the easiest way to use lexical information is splitting the treebank and letting the parser operate directly on word forms as input with a grammar derived from the lexically split treebank. Table 8.11 shows results for this approach and contrasts it with results for the pretagging approach as described in the previous section (row *presplit*).<sup>8</sup> In the third

		parse on words			pretag, parse		
TR	lexicalisation	struct	POS	<i>perp</i> <sup>a</sup>	struct	POS	<i>perp</i>
–	–	80.54	92.38	127.59	83.48	96.04	13.49
+	unrestricted	67.49	82.67	118.24	n/a		
+	no NN,NE,ADJA	76.45	86.98	94.06			
+	cc and ADV	<b>86.02</b>	94.77	98.99			
+	presplit	n/a			87.79	96.04	15.86
+	–	85.43	94.28	105.32	<b>87.91</b>	96.04	10.64

<sup>a</sup>Third and last rows with the same unique terminal vocabulary, all others are different and unique.

Table 8.11: POS Tagging & Parsing Performance vs. Lexicalisation

row, it shows performance on an untransformed treebank. The last row shows performance after Treebank Refinement with ancestor and descendant context, where the *pretag, parse* columns repeat results given before. The *parse on words* columns show results when the parser is fed with words instead of POS tags. The remaining four rows show results for unrestricted lexical POS splitting (*lexicalisation/unrestricted*), or when the three POS NN, NE, ADJA are excluded from splitting (*lexicalisation/no NN,NE,ADJA*), i.e. those POS with very many unique productions. The next row shows results when all closed-class POS plus ADV (everything but ADJA, ADJD, CARD, NE, NN, ADJD, VVFIN,

<sup>8</sup>Most experiments in table 8.11 use only ancestor and descendant context in order to reduce run-time. Additional attribute context is only used in the *presplit* row to determine subcategorised POS.

VVINF, VVPP) can be split (*cc and ADV*). These three experiments require the parser to select POS tags based on syntactic structure. They therefore cannot be used with pretagging and are only given in the *parse on words* columns. The remaining *presplit* row performs unrestricted lexical POS splitting with ancestor, descendant and attribute context, and selects POS for splitting that are either split in descendant context, or where productions occur at least 10 times, and their contribution to the sum according to equation 7.9 is at least 0.1.<sup>9</sup> We show POS tagging performance (*POS* columns) and Parseval  $F_{lab}$  over all node labels (*struct*), as well as cross-perplexity (*perp*). When we do not use lexicalisation at all, the difference between *parse on words* and optimal POS tagging performance (*pretag, parse*) is cut in half by applying Treebank Refinement with ancestor and descendant context (a difference of 3.66 in the first, and 1.76 points in the last row). In row *cc and ADV*, POS tagging accuracy is lifted by another 0.5%. The PCFG parser thus can take advantage of the additional lexical information in the POS tags. The difference to best Parseval performance also decreases, but parsing with words as input still stays clearly behind the pretagging approach (best Parseval for both strategies in bold). While absolute parsing performance is not of direct importance for the evaluation of relative parsing improvements achieved by splitting, lower absolute parsing performance is not desirable. The ability of the POS tagger to handle unknown words may be the reason for its better POS tagging performance. The parser does not use sophisticated smoothing to handle unknown words, but the method outlined in section 5.2.4. Cross-perplexity is only comparable between identical vocabularies. The first and last experiments in the *parse on words* column have the same vocabulary. They are also comparable to all other experiments that operate directly on words. All other experiments have individual vocabularies and therefore the cross-perplexities are not directly comparable (numbers in italics).

In the following, we will discuss some more details of the *presplit* experiment that gives highest Parseval  $F_{lab}$  performance, i.e. on the experiment that determines the POS to be split by lexical items prior to parsing. It turns out that refinement using contexts of unlimited size, and considering lexical items below all POS tags results in a very high number of contexts (17 281 628), which cannot be handled by our current implementation or Treebank Refinement. We therefore restrict contexts in two ways. First, descendant contexts, which make up the largest proportion of all contexts, are restricted to a maximum length of three. Second, we exclude some of the POS tags that are not blocked by our thresholds.

In the above *presplit* experiment, there are four kinds of open-class POS

---

<sup>9</sup>Both determined heuristically.

where at least one production exceeds the thresholds: **ADV**, **NN**, **NE** and **VVINF**. Adverbs (**ADV**) will be considered fully, given their many high-frequency productions observed in table 8.9. There are nine common nouns (**NN**), three proper names (**NE**), and one non-finite full verb (**VVINF**) that are involved in lexical splits in the same experiment (see table 8.12), in addition to fourteen adverbs. One can imagine a very specific usage for each of these lexical

POS	word form (frequency)
<b>NE</b>	<i>dpa</i> (66), <i>taz</i> (55), <i>Christoph</i> (16)
<b>NN</b>	<i>Kto.</i> (20), <i>Foto</i> (12), <i>Tel.</i> (12), <i>Volksbühne</i> (13), <i>Theater</i> (11), <i>Mai</i> (12), <i>Kommentar</i> (10), <i>Betr.</i> (10), <i>Uhr</i> (10)
<b>VVINF</b>	<i>tun</i> (19)

Table 8.12: Open-Class POS in Unrestricted Lexical POS Splitting

items, where they are potentially good candidates for predicting structure. For example, *dpa* and *taz* usually identify the source of a newspaper article in an unattached noun phrase preceding the newspaper article (denominating the article’s source: *Deutsche Presseagentur* and the *taz* newspaper, respectively). Similarly, there are (surprisingly many) authors whose first name is *Christoph*, which are also frequently mentioned at the beginning of an article in a similar way. The common noun *Foto* can often be found at the same position, denominating the source of photographs. Similarly, *Kommentar* (‘commentary’) and *Betr.* as abbreviation of *betrifft* (‘with respect to’) frequently introduce articles. Quite specific usage can also be attributed to the other common nouns, appearing in calls for donations (*Kto.*, for account number), contact information (*Tel.*, for telephone number), or announcements (the generic and proper names of theatres: *Theater*, *Volksbühne*; the time and date: *Uhr*, *Mai*). Frequencies for all these words are rather low (given in brackets), so that we do not lose too many cases of mono-lexical structural dependencies when the respective POS are excluded from splitting. We will consequently ignore all words given in table 8.12 and never split the POS **NE**, **NN**, and **VVINF** in order to reduce the search space for relevant contexts. It is likely that in more restricted domains a higher number of more frequent open-class words will emerge that could be interesting candidates for lexicalised POS, and including them would be more advisable. Despite being revised manually, note that all these items have been originally selected by fully automatic means.

Table 8.13 shows all remaining lexical splits, i.e. the POS tags and the lexical items that are assigned individual new subclasses of the original POS. The table specifies how we perform closed-class lexicalisation in all our remaining

POS	word form
\$ (	- ) ( /
\$.	:
ADV	<i>anders bis da dann dort ganz gar gestern heute hier jetzt mehr schließlich sehr so</i>
APPR	<i>als bei bis in nach neben über vor zu</i>
ART	<i>das den der des die eine einen</i>
KON	<i>aber denn doch</i>
PPER	<i>es ihm ihn ihr ihnen mir uns</i>

Table 8.13: (POS, word form) selected for Partial Lexicalisation

lexicalised experiments. There seem to be different reasons for splitting the POS tags. The punctuation marks tagged \$( are often found in unattached noun phrases (-) or in conjunction with proper names ((, ), /). The colon (:.) is found at the end of orphaned noun phrases much more frequently than in sentence-final position as the other members of \$., like the full stop or the exclamation mark. A larger class of the words shown in Table 8.13 is frequently assigned a wrong POS tag. Most adverbs are always tagged correctly (*anders, dann, dort, ganz, gar, gestern, heute, hier, jetzt, schließlich, sehr*). Of the remaining words, only the preposition *neben* and the personal pronouns *es, ihm, ihn* and *ihnen* are always tagged correctly.

Mistagging function words can have severe impact on parsing, which we try to illustrate with *bis*, which is POS-tagged rather unreliably. It has the gold POS tags ADV, APPR, KON, and KOUS and it is annotated with the accuracy shown in table 8.14 in tune train data. The tags label *bis* as a (subordinating)

gold	accuracy	mistagged as (% of all occurrences)
ADV	5.4 %	APPR (95 %)
APPR	71.7 %	ADV (13 %), KON (5 %), KOUS (2 %)
KON	17.8 %	APPR (57 %), ADV (3 %),
KOUS	63.6 %	APPR (46 %)

Table 8.14: POS Tagging Accuracy for *bis*

conjunction, a preposition or an adverb, and these analyses will obviously have different impact on syntactic structure. Splitting the POS tag of *bis* whenever it is (correctly or not) tagged as ADV or APPR gives the parser a chance to resolve this ambiguity on a higher level for most of the errors shown in table 8.14. Subclassifying POS tags manually has proven useful for some of these tags, too (Müller and Ule, 2002).

All definite articles in table 8.13 except for *des* are also sometimes tagged incorrectly as relative pronouns, so that assigning them lexicalised POS tags may help along the same lines. Giving the parser access to the lexical forms of articles also reveals some information about case. Most articles are not unambiguously correlated with case as is *einen*, which always marks accusative case, or *des* for genitive case. They do, however, show only restricted ambiguity like *das*, *die* or *eine*, which can be only accusative or nominative. The behaviour with respect to tagging errors and morphological ambiguity is rather similar for personal pronouns (PPER). Percolating these frequent function words beyond the level of POS tags may thus help using case information for parsing.

Subcategorising the POS tags of the conjunctions has yet another benefit for parsing. Normally, POS tags of conjunctions (KON) occur below noun phrases most of the time. The conjunctions *aber*, *denn* and *doch*, however, deviate from this behaviour, because they are mainly found in sentence-initial position, i.e. in the KOORD field (*aber*, *doch*), or the PARORD field (*denn*). Moreover, paratactic constructions differ from other coordinations only in the lexical element of the conjunction, so that the PARORD field or the P-SIMPX clause type (for paratactic construction) could never be assigned reliably without access to the conjunction (the prototypical paratactic conjunction is *denn*; Telljohann & al., 2003, pp. 88 and 106; note that we normalise all words to lowercase here).

The remaining lexicalised POS tags are mainly adverbs that are frequently found as modifiers of the verb (V-MOD: *gestern*, *heute*, *dort*, *jetzt*, *hier*) or as general modifiers (MOD: *mehr*, *gar*, *dann*, *da*, *so*). The adverb *anders* frequently modifies a predicate (PRED), as in *Wie immer war früher alles ganz anders*. ('As usual, everything used to be completely different.'). Still other adverbs modify adjective phrases (*so*, *ganz*, *sehr*), or are themselves the heads of complex phrases (*gestern* and *heute*) as in *erst seit gestern* ('only since yesterday').

Introducing lexicalisation to refinement, either by applying the parser directly to words, or by pre-splitting POS, does not beat results obtained without lexical context. This is rather surprising, given that words carry a wealth of information not available in plain POS tags. Some of the lexical information seems to be especially important in a language like German that has rather rich inflectional morphology, and rather free constituent order. The results that did not show too much improvement with lexicalisation, though, were results of evaluation metrics not specifically geared towards peculiarities of German, nor to those of the TüBa-D/Z annotation scheme. The next section tries to examine whether introducing more, and especially lexical, context into node labels has other more noticeable effects on parsing TüBa-D/Z.

## 8.4 Recursiveness

The small change to the worse that closed-class lexicalisation inflicts on the labelled Parseval  $F$ -score is somewhat surprising, given that lexical information is normally deemed quite relevant, which is confirmed by experiments that examine the usefulness of closed-class lexicalisation for parsing German (Schiehlen, 2004). A closer look beyond the standard Parseval evaluation reveals some differences between the data used in those experiments and ours, though. Parseval specifies a weighted average of precision and recall on the node labels given in the annotation of a treebank. We have argued so far that the node labels of a treebank may be overly specific or too general, and will extend this question on the evaluation of parser results.

Constituent labels in a generative framework capture the generalisation that a language allows recursion in all situations where a node can be embedded into a node with the same kind of label. In a treebank this means that noun phrases, whether complex or simple, always receive the same label. There will be, however, many more simple noun phrases (at least one for each NN and NE POS tag in TüBa-D/Z) than complex ones (only where an NX makes up a single complex constituent together with another constituent). When we count all noun phrases (NX) as *complex* that dominate at least one other NX, and the rest as *simple*, then the fraction of simple noun phrases from all NX is more than 75%. Looking at the performance of the baseline model on simple vs. complex NX, we find a Parseval  $F_{lab}$  of 93% on simple vs. 45% on complex NX. Performance on simple NX comes near POS tagging performance on the set of tags that are directly dominated by NX in the test set most of the time, which is 95.2% (NN, ART, NE, KON and PPER, covering 80% of all words dominated by simple NX, the rest being covered by 22 other POS tags). Noun phrases also often make up a rather large proportion of syntactic structure of a treebank (31% of the nodes in TüBa-D/Z tune test data). When Parseval metrics are optimised for a parser and a treebank, then the overall number of correctly annotated nodes with the respective label needs to be maximised. This figure will naturally be dominated by the 24% of all nodes that are simple NX in TüBa-D/Z. A closer inspection of the less frequent but more complex constituents thus seems in order to shed more light on the value added by the parser on top of the POS tagger.

Table 8.15 gives the frequencies of occurrence of all node labels in the TüBa-D/Z tune test data set. Each node label is subdivided into two groups according to the number of nodes of the same type that it dominates directly or indirectly: *level 0* (or *simple*) when it does not dominate any node of the same type, and *not level 0* (or *complex*) if it does. The labels are also grouped into those belonging to the constituent domain, and those that make

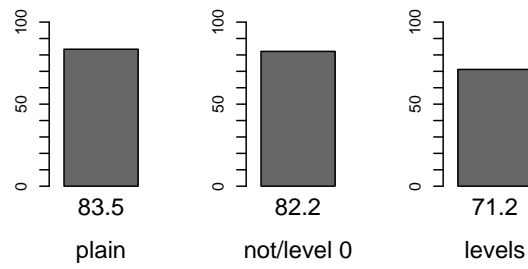
	TopF, Clause		Constituents	
	level 0	not level 0	level 0	not level 0
all	19 487	1584	33 405	5886
type	MF 4 423	SIMPX 1 230	NX 14 429	NX 4 558
freq	LK 3 291	VF 151	PX 4 283	PX 727
	SIMPX 3 100	NF 128	VXFIN 4 272	ADJX 326
	VF 2 914	MF 63	ADJX 4 119	ADVX 214
	VC 2 727	R-SIMPX 9	ADVX 3 226	EN-ADD 21
	C 992	FKOORD 2	VXINF 2 058	VXINF 18
	NF 869	FKONJ 1	EN-ADD 896	VXFIN 15
	FKONJ 397		FX 72	FX 7
	R-SIMPX 344		DM 33	
	FKOORD 205		DP 17	
	KOORD 168			
	PARORD 24			
	LV 20			
	P-SIMPX 13			
	VCE 0			

Table 8.15: Frequency of Recursive and Non-Recursive Node Label Usage

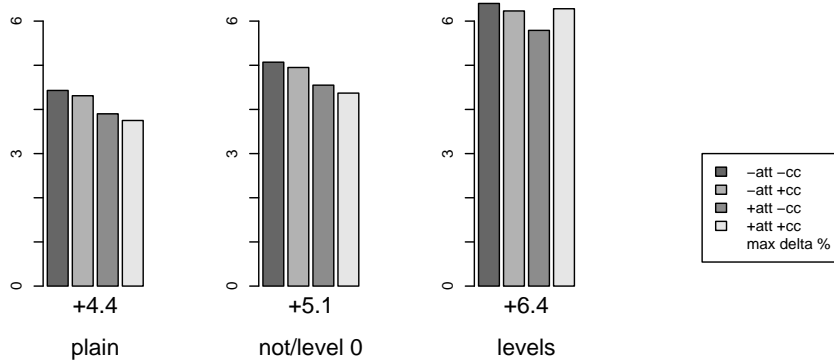
up topological field and clause structure. With the single information about recursion added, it becomes apparent that those nodes that form the higher levels of complex structure are considerably less frequent, and that they do not show the same frequency distribution as their level 0 counterparts. Especially for topological fields, these differences may give a false impression of performance when performance differs between the simple and the complex case. The middle field (MF) is an extreme example, where the complex case shows up in less than two percent of all cases, although this label is the most frequent topological field label overall.

By averaging over all occurrences of each label, the parseval metrics are likely to be a bit too optimistic about performance, assuming that performance on complex structure can be even more interesting than performance in the base case. Performance in the complex case touches phenomena such as coordination or clause attachment in TüBa-D/Z. Figure 8.17(a) shows baseline performance on the untransformed treebank in the leftmost bar (*plain*). The next two bars distinguish between simple and complex usage of node labels (*not/level 0*), and between all individual levels of recursion for all node labels (*levels*). It shows that a plain PCFG performs slightly worse when simple and complex usage of node labels is evaluated separately (a drop of 1.3%),





(a) baseline performance

(b)  $\Delta F_{lab}$  between Parseval baseline and split performanceFigure 8.17: Parseval  $F_{lab}$  Discriminating Levels of Recursion

and considerably worse when each level of recursion is evaluated separately (a drop of 12.3%). In contrast, plain Parseval performance of 83.5% does not seem too bad in absolute terms at first glance. It becomes evident that overall Parseval evaluation following the original node labels of TüBa-D/Z can be quite different from also evaluating the level of recursion for some, or all levels. Evaluating each level of embedding separately gives a more fine-grained impression of differences in performance between the levels.

Figure 8.17(b) shows the changes in baseline performance caused by parsing with a treebank grammar obtained from a refined treebank. The four bars correspond to four combinations of using ancestor and descendant context with or without closed-class lexicalisation (*cc*) or attribute context (*att*). While absolute performance is still best for just ancestor and descendant context, the improvements between the splitting regimes differ when evaluating all levels of recursion separately, and distinguishing recursive and non-recursive usage works almost as well as the annotation of plain node labels (87.9% vs. 87.3%).

### 8.4.1 High-Frequent Node Labels

Figures 8.18, 8.21 and 8.24 look closer into those node labels that represent labels of the most frequent non-recursive (*level 0*) or recursive (*not level 0*) types of fields/clauses, or constituents (first row of *type freq* for table 8.15). Together, they make up 72% of all nodes in the treebank. You cannot score above 43% Parseval  $F_{lab}$  if these fields are annotated with zero  $F_{lab}$ , and you cannot fall below 84% when they score perfectly at  $F_{lab} = 1$ , just to give a rough impression of their impact on plain Parseval  $F$ -score. The figures consist of a left part (b) that shows baseline performance of the untransformed treebank grammar, and a right part (a) that shows absolute difference in  $F_{lab}$  between the performance of the baseline grammar and the four grammars derived from the four different Treebank Refinement regimes, given by optional attribute context (+/- *att*) and optional closed-class lexicalisation (+/- *cc*). The baseline is given in bars and numbers below the bars. Evaluation is performed either on the original unchanged node labels with Parseval  $F_{lab}$  (*plain*), on node labels that distinguish between level zero and dominating some node of the same type (*not/level 0*), and last, on labels where each level of recursion is evaluated separately (*levels*). In addition to the overall performance for distinction of level 0, or every single level, which is given in the upper left, all individual *levels* are also evaluated separately in the centre. Finally, performance of either level 0 or on all other levels is given separately on the right-hand side. Frequencies for each subclass are also given below the bars. All these details are repeated for the split grammars, where

the numbers below the bars represent the maximum positive change after transformation. All Parseval  $F_{lab}$  relate to results on the tune test data set, and also frequencies, which are given below the baseline bars, are drawn from this set. Note that unattached nodes are represented by distinct node labels, so that sentential parentheses found embedded in a sentence do not count as recursion. Largest  $F$  values are marked bold for the middle parts.

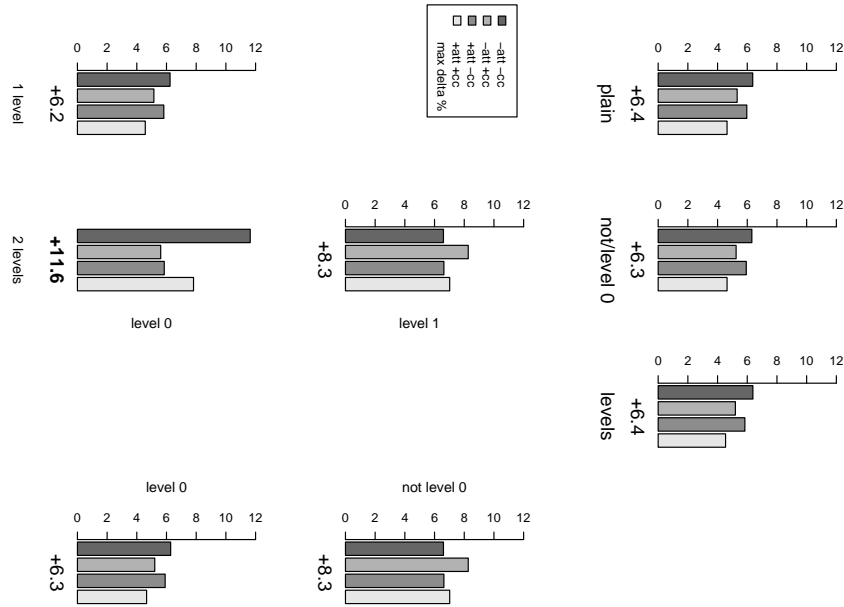
### 8.4.2 MF, Recursively

Figure 8.18 shows Parseval performance on middle field nodes (MF), which are the most frequent TopF/clause node labels overall. MF nodes show a maximum recursion of depth one, i.e. a MF node dominates at most a single other MF node. Almost all middle fields are non-recursive (4347 of 4486). Note that the FKOORD label groups coordinations of all kinds of field labels, in contrast to clausal or phrasal nodes, which are used recursively to represent coordination. Seven MF nodes have been split in the +att +cc experiment, and one of these splits connects recursive MF nodes directly via local context (i.e. either by context included directly in a node, or via the most likely production of such a node). This local context is shown in figure 8.19(a). 11 of the recursive MF nodes annotated by the parser are linked via the SIMPX#60 shown. The splitting regime –att –cc, which performs best overall, splits MF nodes just once, but also connects recursive MF usage via local context. Although this link between recursive MF is more direct (see figure 8.19(b)), it applies to only three parses. The sentential node connecting recursive MF is most frequently R-SIMPX, which is never split by any of the four splitting regimes. Relative clauses thus do not seem to behave unexpectedly when embedded in a middle field.

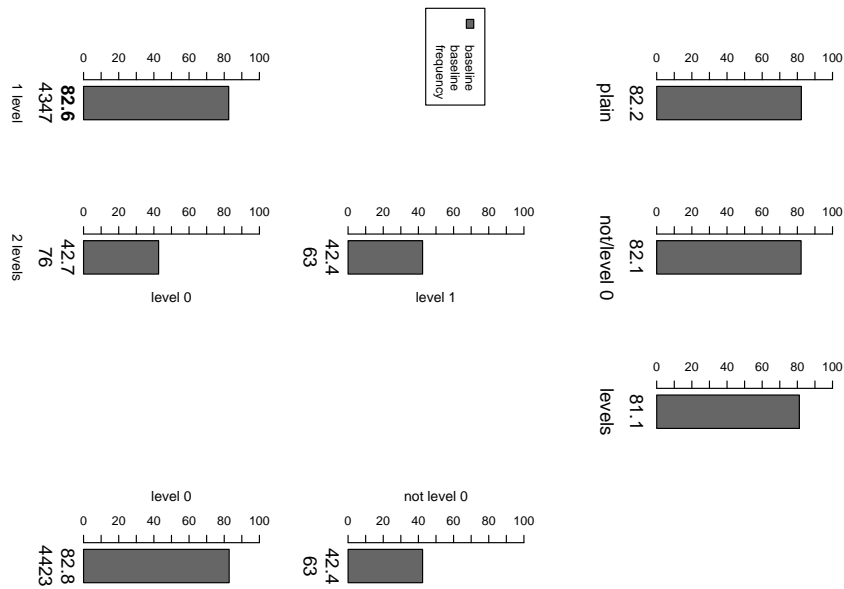
Figure 8.20 shows Parseval  $F_{lab}$  for MF nodes after each iteration of refinement for the +att +cc. Iterations where the focus of the split is an MF node are marked with small squares. MF nodes mostly seem to benefit from splits applying to other nodes: the biggest improvement is caused by split #2, which marks noun phrases in the complementiser field and thus allows more reliable annotation of verb-last sentences, including their MF.

### 8.4.3 SIMPX, Recursively

SIMPX nodes appear in a much greater variety of recursion levels – they are nested up to six levels deep (see figure 8.21). However, most of their population occurs in configurations of up to three levels (almost 99% of the overall 4330 occurrences in the test set). For all SIMPX nodes, splitting improves performance quite drastically. Even those parts of more frequent recursive



(a)  $\Delta F_{lab}$  between baseline and split performance



(b) baseline performance and frequency

Figure 8.18: Parseval  $F_{lab}$  for Levels of MF Recursion

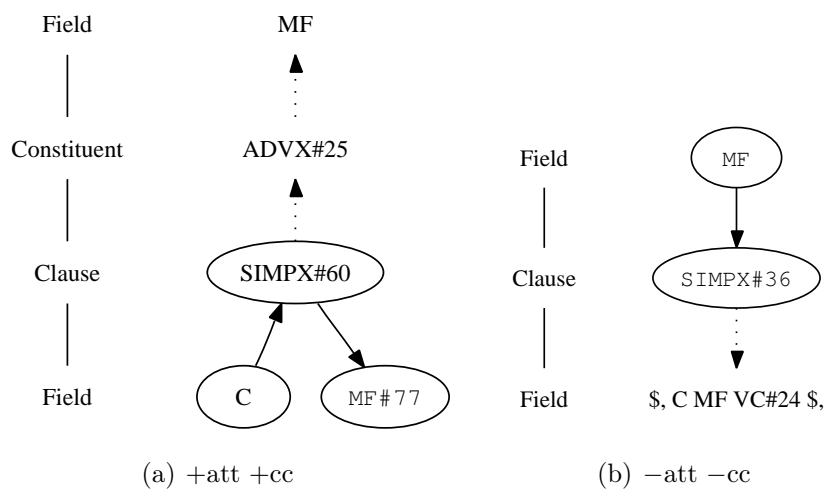


Figure 8.19: Local Context Connecting MF Recursion

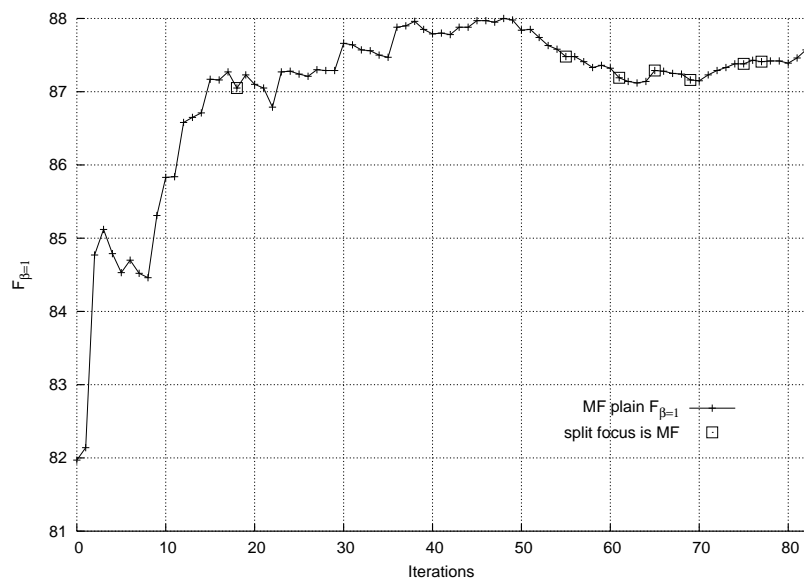


Figure 8.20: Iterative Performance on MF with +cc and +att Context

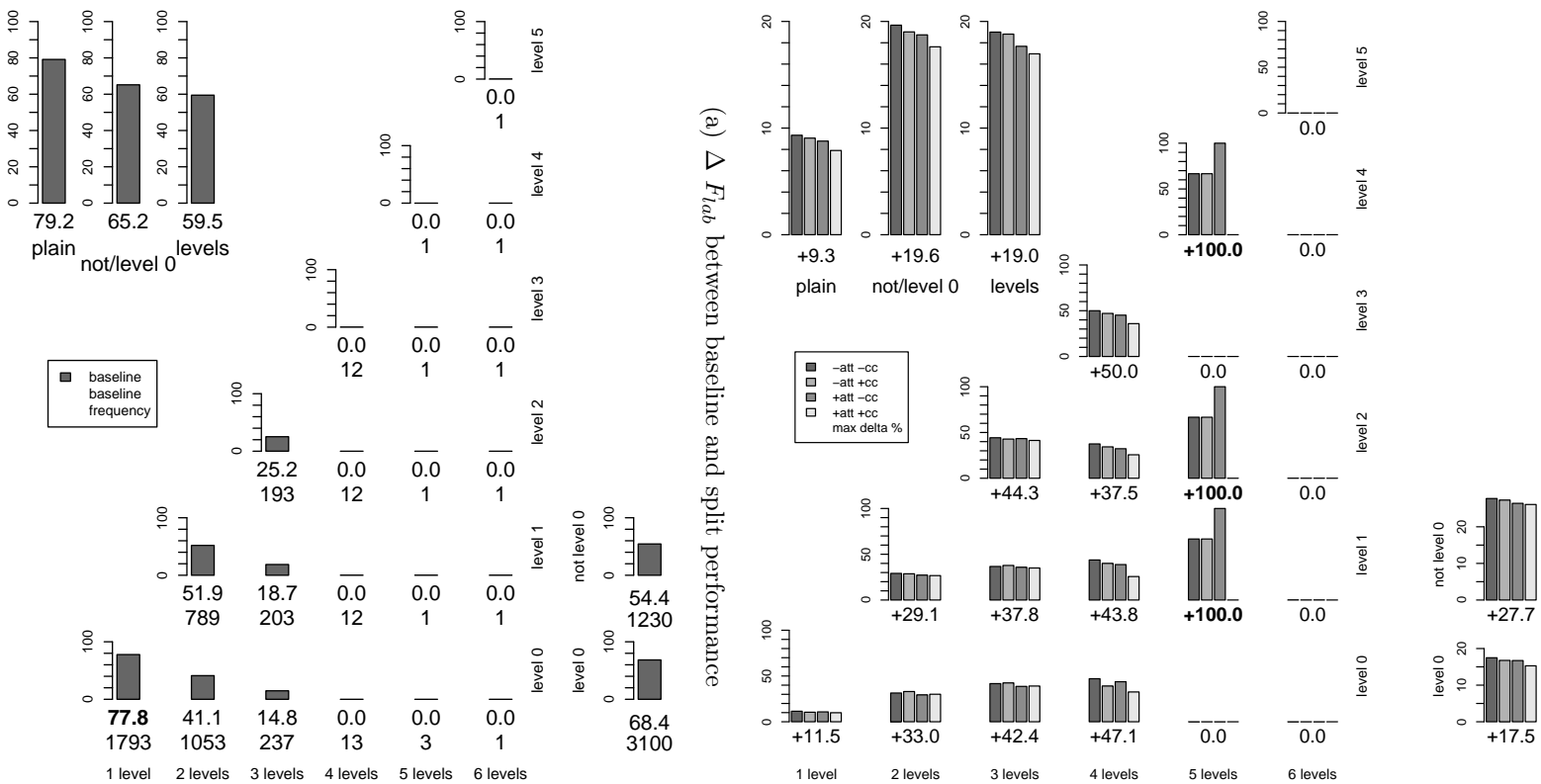


Figure 8.21: Parseval  $F_{lab}$  for Levels of SIMPX Recursion

structures that have a poor baseline performance come close to being acceptable after refinement. E.g. the lower node of a recursive **SIMPX** structure of depth two reaches 74%  $F_{lab}$  after refinement (baseline: 41%), and of depth three 57% (15%). Overall performance according to evaluation by levels of recursion accordingly gains more than *plain* evaluation: +19% for *levels* (over a baseline of 60%) vs. +9% for *plain* (over 79%). The increase for four levels of recursion is accordingly quite large (between 38 and 50 points) and in sharp contrast with the zero baseline. No **SIMPX** node seems to suffer from splitting. Even some low-frequent parts of structures nested five levels deep are annotated correctly, while not found at all in the baseline parses. Similar to **MF** nodes, **SIMPX** nodes do not seem to benefit from lexical, or attribute context. We think that this is due to complex sentence structure being less directly connected to lexical items than complex constituent structure.

Figure 8.22 shows *plain* performance of the refined treebank grammar on **SIMPX** nodes after each iteration of the +att +cc setting. Most of the overall

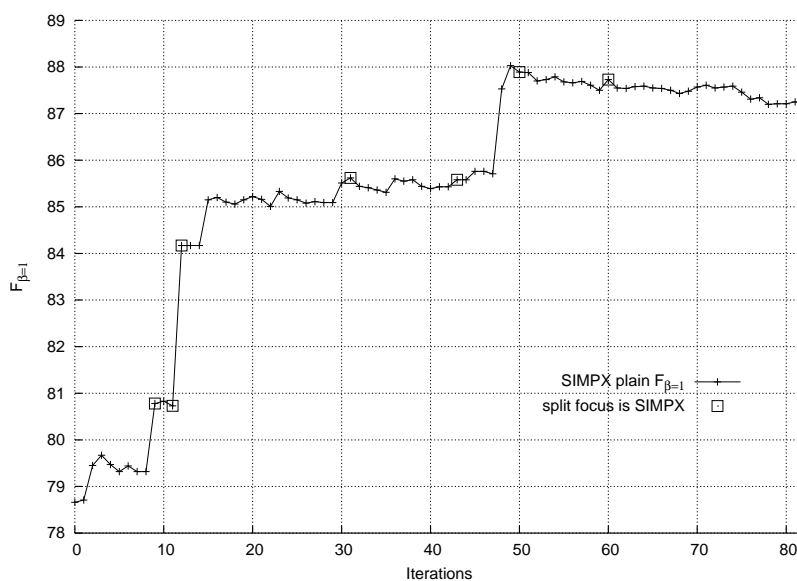


Figure 8.22: Iterative Performance on **SIMPX** with +cc and +att Context

improvement of 9%  $F$  originates in four splits (9, 12, 15, 48). The first two of these splits have **SIMPX** nodes as their focus (see figure 8.23(a)). These **SIMPX#9** and **SIMPX#12** make up 55% of all **SIMPX** nodes that appear in recursion of at least three levels in the parser's output. The context triggering the **SIMPX** splits shown in figure 8.23(a) covers relations between clause structure and topological fields: coordinated simplex clauses are found below another **SIMPX** node, and a **SIMPX** node in a final field (**NF**) preferably groups verb-last sentences. The fact that an **NX** label covers punctuation much more

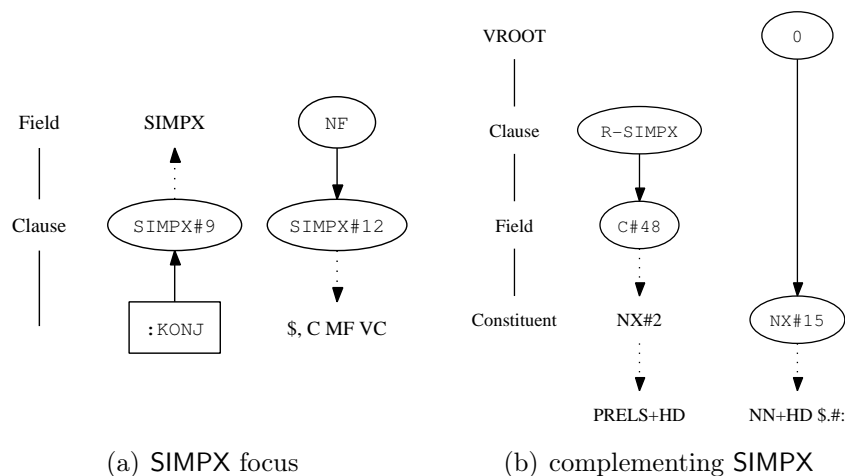


Figure 8.23: Splits Improving Annotation of SIMPX Nodes

frequently when found unattached than it does elsewhere (e.g. in a SIMPX) is another structural preference, as is the fact that the complementiser field holds subordinating conjunctions much more frequently in simplex clauses than in relative clauses (expressed via the complement set of C#48 that encodes a preference for relative pronouns for the C field of R-SIMPX). These local contexts are included into the other two node label splits that are most helpful for annotating SIMPX nodes given in figure 8.23(b). The #: expresses a lexical split of the POS tag \$. that occurs where it dominates a colon, as in sub-headings like *Kino:* (‘at the cinema’), *Premiere:* (‘first night’), which are often found at the beginning of certain newspaper articles. This split has been introduced by closed-class lexicalisation, but despite the name, you could also think of a colon as a structural indicator instead of a prototypical lexical item. Punctuation marks seem to be more closely connected to complex sentence structure than to semantics here. The splits shown in figure 8.23 have been considered most important in a splitting regime with access to closed-class lexical information and all edge labels (+att +cc). Still, among the contexts selected by Treebank Refinement, those that represent structural preferences seem to be most useful for improving the annotation of SIMPX nodes. Rather than lexical selection, these contexts express structural preferences.

#### 8.4.4 NX, Recursively

Noun phrases (NX) have the most frequent node label overall – they represent more than 30% of all nodes in TüBa-D/Z. They are found in rather



complex configurations of up to seven levels of embedding (see figure 8.24).<sup>10</sup> More than seventy percent of all their occurrences are found in rather flat noun phrases of depth one or two, but **NX** nodes are spread more evenly over recursive constructions of different complexity than **SIMPX** nodes: noun phrase nodes that are part of complex noun phrases of four or more levels make up more than ten percent of all occurrences. Performance varies quite drastically with nesting. On base noun phrases (*level 0 NX* with any number of dominating **NX**), Parseval  $F_{lab}$  is quite high for the baseline model already and reaches 92.8%. Base noun phrases make up more than three quarters of all **NX** nodes. Performance gains from splitting for all but the most complex noun phrases. While this improvement is rather low for base noun phrases (+0.5%), it reaches almost 20% for more complex configurations (where there are five levels of **NX**, maximum shown in bold). Despite this increase, only the less complex configurations with up to three levels of embedding are lifted beyond an  $F$  of 50% by splitting. Baseline performance even fails to exceed the level of 50%  $F$  for determining those **NX** that dominate another **NX** (*not level 0*). Splitting improves on this quite considerably, reaching more than 60% with attribute context and closed-class lexicalisation.

The results generally show a tendency of the more agnostic parameter sets (no attribute context, no closed-class lexicalisation) to prefer the high-frequency structures with lower complexity (cells to the lower left). Recursive structure gains through all kinds of splitting, while closed-class lexicalisation and attribute context both perform worse alone than in combination for noun phrases.

The types of context that are used to trigger splitting seem to have a different impact on assigning structure containing **NX** nodes than for structure containing **MF** and **SIMPX** nodes. For all but the most infrequent configurations of **NX** (i.e. for zero to five levels of recursion), attribute context wins in combination with closed-class lexicalisation. It seems to be here that closed-class lexicalisation has its most positive effect on parsing performance. On the other hand, the increase is very low for base noun phrases for this splitting regime, so that overall, +att +cc just reaches a similar increase in performance as ancestor and descendant context alone for *plain* evaluation.

Figure 8.25 shows performance per iteration on **NX** nodes for the +att +cc setting. Like the other two curves for performance per iteration, figure 8.25 first shows a steep increase up to near maximum performance, and then stays within a small corridor for the remaining iterations. Refinement does

<sup>10</sup>**NX** nest seven levels in e.g. *mit [Rußland [als [Ressourcenlieferant und [[militärischer Ausrüster [[Indiens] und [Chinas]], bis hin zu Atomwaffen]]]]* ('Russia delivering commodities and military goods to India and China, including nuclear weapons') in TüBa-D/Z sentence 2409.

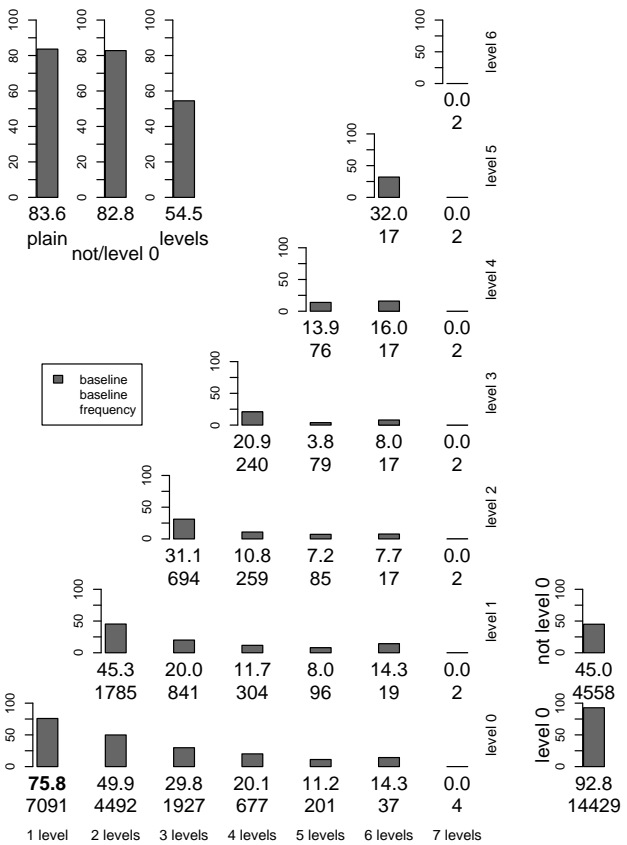
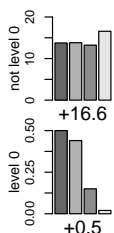
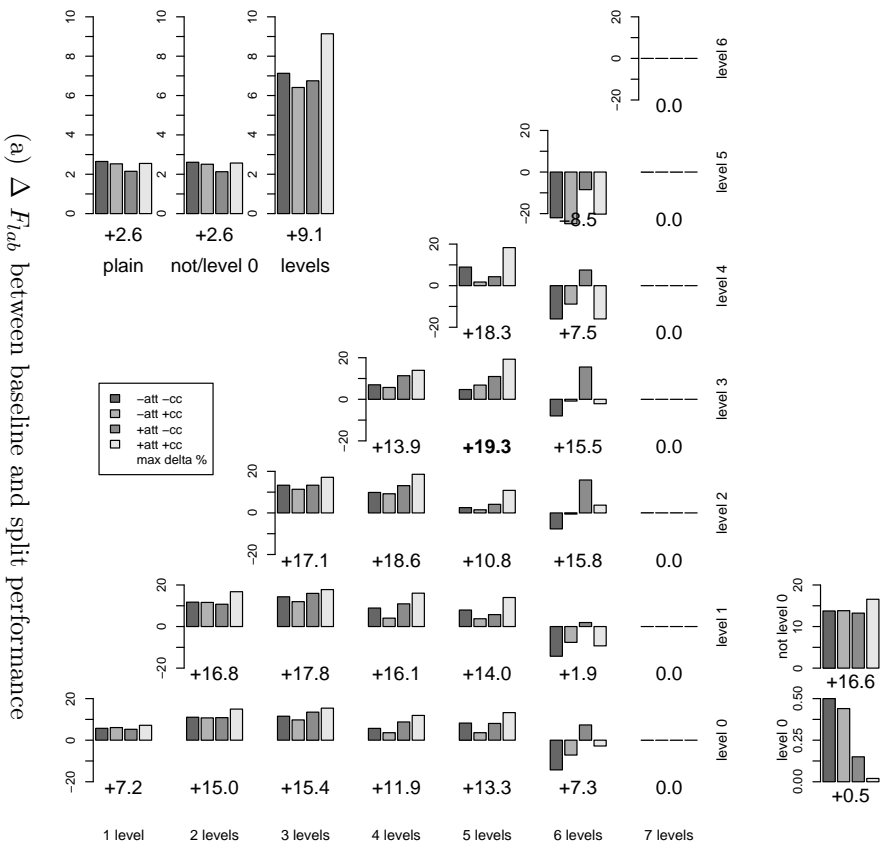


Figure 8.24: Parseval  $F_{lab}$  for Levels of NX Recursion



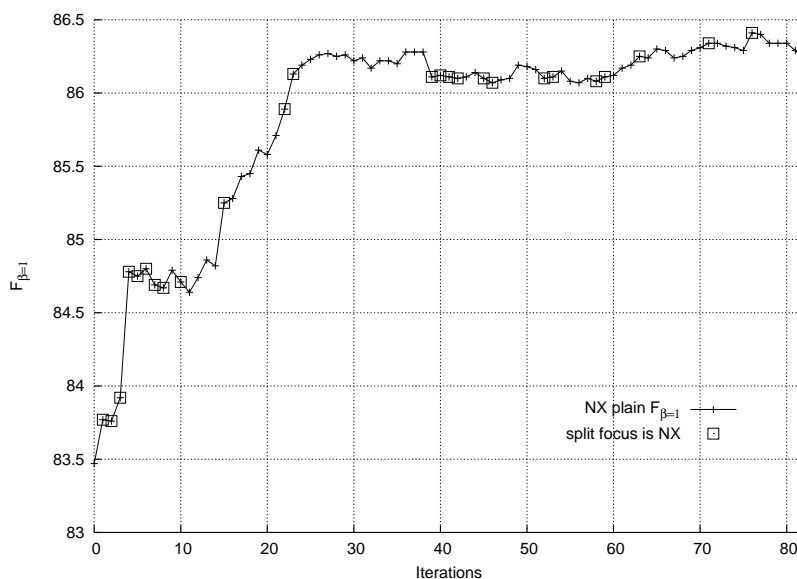
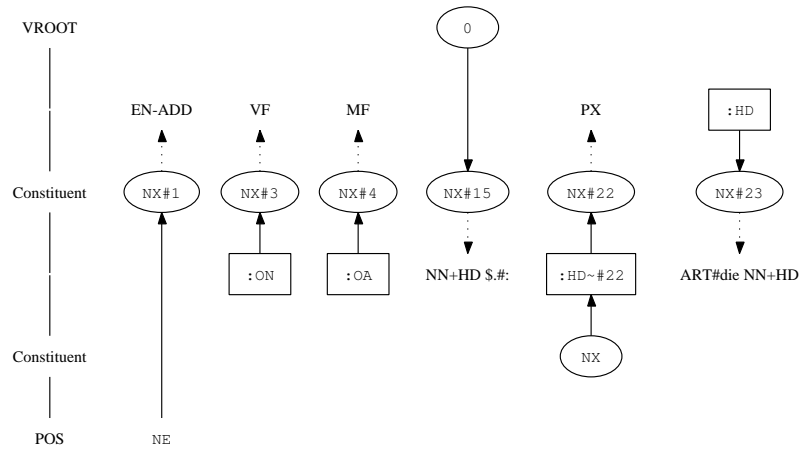
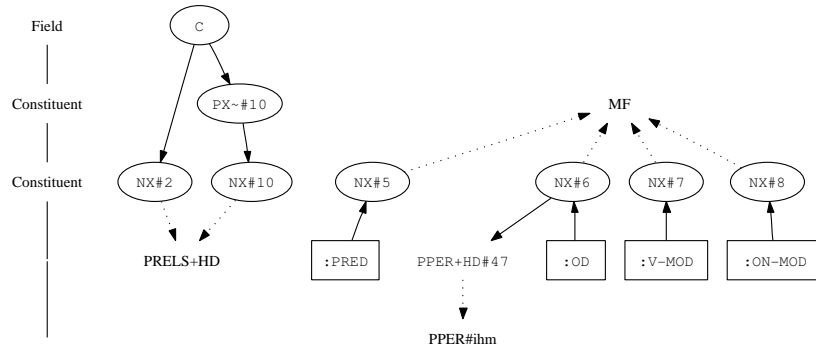


Figure 8.25: Iterative Performance on NX with +cc and +att Context

not seem to require an exact stopping condition judged by the curves for MF, SIMPX and NX. Quite a high proportion of all splits applies to focus NX nodes, which can be partly explained by their high overall frequency, in combination with the split metric  $D_{f_{min}}^{SDP}$ , which prefers high-frequency focus nodes in context. But it also seems that TüBa-D/Z's NX nodes are overly general for PCFGs. Figure 8.26(a) shows local context transformed into those NX nodes that are the main contributors to improvements in *plain* NX performance among those initial splits that lift performance on NX nodes to near its peak (iteration 1-30). Interestingly, there are as many NX focus nodes among these first iterations that do not seem to lift NX performance as there are NX focus nodes that lift performance (the former shown in figure 8.26(b)). The NX node splits that increase performance single out nominal constituents in proper names (split #1) and unattached NX (#15). They also include head information from edge labels to mark nested nominal constituents that tend to be part of prepositional phrases (#22), and to mark the nominal lower end of complex constituents to prefer *die* as an article followed by a single common noun (#23). NX#23 thus links closed-class lexicalisation directly to the constituent label via its most frequent production. The remaining two splits improving performance on NX do this as well, in effect, although the lexical information is not visible in the local context presented in the figure. The single most helpful split for NX, which includes the accusative object edge label OA into noun phrases (NX#4), encodes a preference for accusative objects to appear in the middle field, while nominative objects (ON; the subjects)



(a) improving NX annotation



(b) neutral with respect to NX annotation

Figure 8.26: Splits of NX Nodes Before Optimal Performance

prefer to occur in the initial field (NX#3). The resulting new nodes show a preference for different productions, including *der* as article followed by a common noun in the nominative case, and *den* and *einen* in the accusative case (not shown in the figures).

The same mix of structural and lexical contextual information is present in the NX splits that do not have a noticeable effect on NX performance (figure 8.26(b)). There is an explicit connection to lexical items via local context for dative objects (OD): NX nodes with an OD edge label are first split because they prefer to occur in the middle field, resulting in a new node label NX#6. This node label later triggers a split of dominated personal pronouns to prefer *ihm*. All splits of NX triggered by edge labels representing grammatical functions encode a preference for a certain linear position in the sentence. Those shown in figure 8.26(b) are less frequent than those improving NX performance shown in figure 8.26(a). Additionally, the splits in the lower part of the figure seem to connect the grammatical functions less often with anchors in closed-class lexicalisation. Table 8.16 shows the frequency of occurrence for NX#3 to NX#8 in tune train data. The splits of NX in context

node	NX#3	NX#4	NX#5	NX#6	NX#7	NX#8
has edge	ON	OA	PRED	OD	V-MOD	ON-MOD
frequency	8892	4554	798	692	261	206
cc-lex child	23.1 %	22.2 %	19.4 %	18.6 %	10.7 %	14.1 %

Table 8.16: NX Splits Linking Grammatical Functions and Lexical Items

ON and OA that increase performance on NX are much more frequent than the rest, and they also include more often at least one production containing POS tags that have been lexicalised (row *cc-lex child*). The split heuristics looks for a context in one direction (here: the attribute context) and looks for a correlation with the other direction (here: the dominating node). The correlation between the dominating field label nodes as context and the children of the NX as productions, which partly have been lexicalised, seems to be much weaker than the correlation between attribute descendant context and the field label productions. The preferences correlating with the more frequent grammatical functions ON and OA are local enough to be captured by Treebank Refinement. The preferences for the less frequent grammatical functions, however, seem to be too fine-grained to be encoded into node labels to improve performance. The splits #5 to #8 all encode a preference for appearing in the middle field. This more general preference may be useful if applied to all these nodes at once, generating a single new node label instead of four distinct labels (see discussion of merging node labels in chapter 9).

We would like to show finally that the overall performance curve of Parseval  $F_{lab}$  per iteration is again similar to the individual curves for MF, SIMPX and NX (all with +att +cc, see figure 8.27).

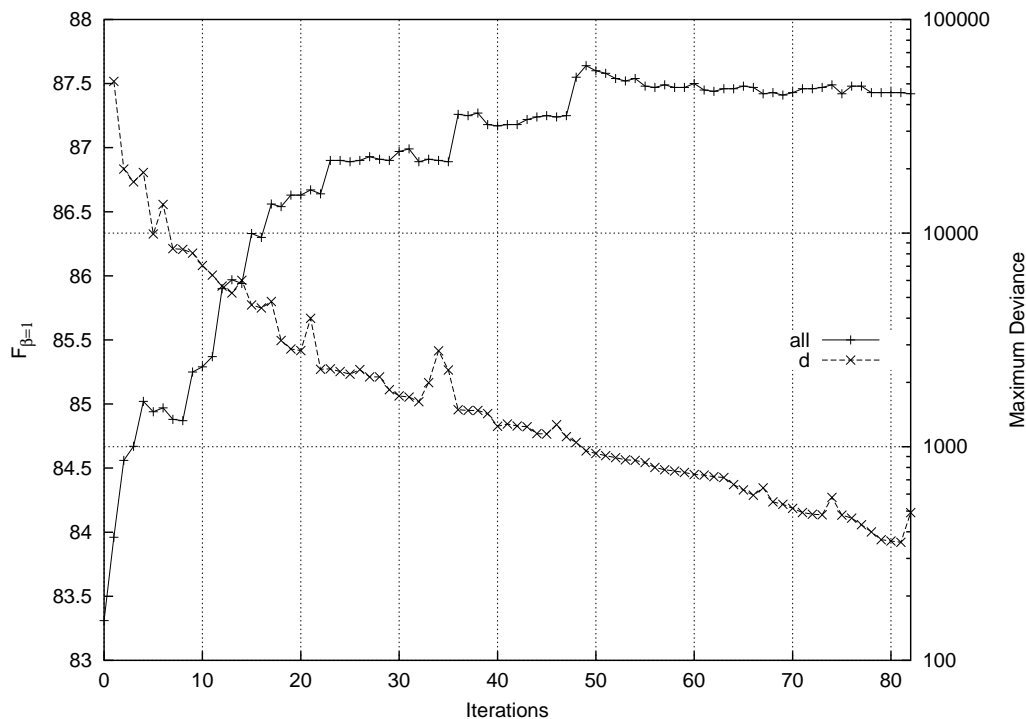
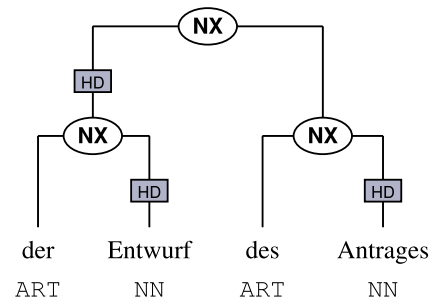


Figure 8.27: Iterative Behaviour with Extended Context

### 8.4.5 The Impact of Lexicalisation in TüBa-D/Z and negra

Closed-class lexicalisation has only access to articles, prepositions, personal pronouns, conjunctions, adverbs, and some punctuation marks, so that it can never disambiguate attachment ambiguities on the basis of the heads of the constituents to attach. Instead, certain configurations of closed-class word forms hint at a certain default modification structure. The article *des*/ART, e.g., prefers to be embedded into two NX like in figure 8.28. Most of the noun phrases including this article are analysed similarly, apart from about two percent that are part of a prepositional phrase (*während/trotz/anlässlich des* ...; ‘during’/‘in spite of’/‘on the occasion of the ...’). Figure 8.29 shows the distribution of grandparents of all articles (ART) that have been lexicalised in the tune train set according to closed-class lexicalisation. The figure clearly



Der Entwurf des Antrags  
 the draft of the proposal  
 ‘the draft of the proposal’

Figure 8.28: Default Genitive Modification Structure for *des/ART*

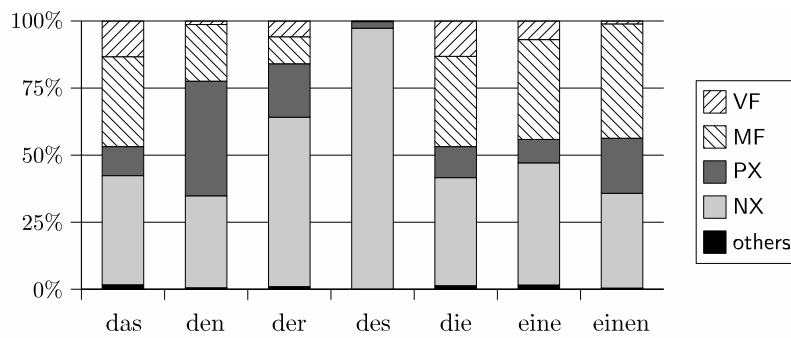


Figure 8.29: Proportion of Grandparents for some Articles (ART)

shows that *des*/ART tends to be embedded into a complex noun phrase much more frequently than *der*/ART or *den*/ART.<sup>11</sup> It is in this sense that richer structural information via attribute context allows annotating configurations of complex constituents more reliably. Parses cannot be chosen according to the lexical heads, but configurational preferences are adapted so that there is a more fine-grained default constituent structure.

Results of plain evaluation have been found to be dominated by *level 0* nodes for NX, MF and SIMPX node labels. Structural preferences are only anchored in closed-class lexicalised POS, which can only provide structural preferences as just outlined, but which cannot take advantage of full lexical information. More lexical information is accessible when the parser operates directly on words, without POS tagging as a preprocessing step. This approach has not come up to the same performance as using POS tags as input to the parser, though. It seems that the dominance of level 0 annotation and its close link to POS tags cause this behaviour.

The strength of the link between POS tags and the lowest layer of syntactic annotation is related to the amount of surprise that a certain node dominates a given POS tag, which is given by the conditional entropy  $H(N|T)$  of the distribution of pre-terminal node labels  $N \in \mathcal{N}$  given a POS tag terminal  $T \in \mathcal{T}$ .<sup>12</sup> Table 8.17 shows this conditional entropy, which can be calculated as  $H(N|T) = -\sum_T \sum_N p(T, N) \log_2 p(N|T)$  (Manning and Schütze, 1999, p. 64). The numbers are derived from the first 1000 sentences of *negra* and

Treebank	plain	lexicalised	$\Delta$
TüBa-D/Z	0.24	0.12	0.12
<i>negra</i>	2.41	0.88	1.53

Table 8.17: Conditional Entropy of Level 0 Node Labels Given POS

TüBa-D/Z using only gold POS tags (column *plain*), and alternatively after adding the words to the POS everywhere (column *lexicalised*). The last column shows the decrease in the entropy caused by adding the lexical information. The predicted sequence of POS tag parents is obviously not sufficient to fully construct the lowest level of annotation, which would need at least tags that mark the beginning of a constituent, and the following tags belonging to the same parent, and those that do not belong to the parent (e.g. IOB tags, Tjong Kim Sang and Veenstra, 1999). We still think that the figures highlight a difference between the treebanks even if they do not

<sup>11</sup>The direct parent of ART is always NX; counts include uppercase variants.

<sup>12</sup>We do not distinguish random variables and basic outcomes notationally here, because both refer to the same events.



measure exactly how predictable the full syntactic annotation is, given the POS tags. Given a POS tag, it seems to be much easier to predict the lowest level of syntactic annotation for TüBa-D/Z than for negra. Adding lexical information to all POS tags reduces the surprise for negra much more than for TüBa-D/Z, which is likely to explain the contrast between the positive effect of lexicalisation observed by Schiehlen (2004) for negra and the smaller impact on the results reported here for TüBa-D/Z.

Attempting to assign complex structure in TüBa-D/Z seems to be especially unrewarding with respect to Parseval  $F_{lab}$ , because an incorrect complex annotation is penalised twice (a missing constituent only penalises recall; a wrong additional constituent also penalises precision). A strategy that focuses only on high-frequent less-complex structure would just leave out the complex constituent (only a penalty on recall) and could moreover concentrate on that part of syntactic structure that is easily predictable from POS tags. Ignoring all complex (*not level 0*) structure amounts to ignoring just 12 % of all nodes. Ignoring just complex *constituents* only ignores 9.8 % of all nodes. A parser capable of annotating base (*level 0*) structure perfectly while ignoring all complex structure would achieve an Parseval  $F_{lab}$  of 93 %, or 95 % if recursive topological fields were also covered perfectly. The annotation scheme of TüBa-D/Z in combination with plain Parseval evaluation is thus especially geared towards less complex structure, and gives an arguably bad impression of performance on those parts of structure that are more difficult and more informative. Evaluating recursive structure separately as shown here may give a more balanced impression on parser performance.

## 8.5 Varying Detail of Input and Output

Table 8.18 summarises results on lexicalised and unlexicalised experiments considering edges as attribute context or not, and it adds an evaluation of the annotation of grammatical functions. It also adds results on including all edge labels a priori as an alternative to attribute context. The table tries to highlight the impact of different input information on PCFG performance before and after refinement, and adds evaluation of some structural relations that are an essential aspect of complex constituents.

The first two columns show which information is provided as input to parsing on top of labelled constituents, where no additional information means that labels are used as given in TüBa-D/Z, and the POS tags as provided by `tnt` according to the STTS. The *edge* column shows additional information from edge labels that is either provided as optional information to splitting via attribute context (*att*), or by affixing each node label with

input		unsplit performance				split performance			
edge	lex	parseval		functions		parseval		functions	
		plain	not/10	parsv	dep	plain	not/10	parsv	dep
		83.5	82.2	n/a		<b>87.9</b>	<b>87.2</b>	n/a	
	cc	83.3	82.0			87.8	87.1		
att		as above				87.4	86.7	51.1	16.2
att	cc					87.4	86.7	56.8	2.8
fix		85.0	78.8	54.0	55.1	86.8	80.0	56.3	59.1
fix	cc	85.2	78.7	57.0	57.5	86.8	79.7	<b>59.6</b>	<b>61.3</b>

Table 8.18: Performance on Structure and Functions

the dominating edge label a priori (*fix*). The second column (*lex*) shows a *cc* when closed-class lexicalisation is performed as introduced before. The next four columns under the heading *unsplit performance* show performance of a plain PCFG treebank grammar with node labels representing the respective input information. The two *parseval* columns show Parseval  $F_{lab}$  on constituents (*plain*), and on labelled constituents with those constituents being evaluated as a separate class that do not contain any other constituents of the same type (column *not/10*). The *functions* columns evaluate a subset of edge labels representing grammatical functions. Column *parsv* shows Parseval  $F_{lab}$  on all node labels embroidered with an edge label representing one of the grammatical functions accusative, dative, genitive, or nominative object (represented by the edge labels OA, OD, OG, ON), sentential or prepositional object (OS, OPP), or predicative (PRED). This column thus shows Parseval  $F_{lab}$  on constituents representing maximal projections of one of these grammatical functions in a sentence. It neither evaluates whether the head of the constituent has been determined correctly, nor which is the word that it depends on (both would require additional edge labels as head markers). This is what the next column (*dep*) shows. It gives the dependency evaluation on the same set of grammatical functions derived from the constituent structure, i.e.  $F_{lab}$  on the tuples representing dependency arcs between the heads of clauses and the objects being assigned one of these functions (as outlined in section 3.2). Note that all *functions* columns rely on information encoded in edge labels, because this is where TüBa-D/Z encodes head percolation and grammatical functions; the *functions* evaluation can thus only be performed when edge labels are always present in the output (*fix* in first column) or when edge labels are an optional part of the output (*att* after splitting). It cannot be performed where edge labels are not considered at all. The dependency evaluation moreover relies on an uninterrupted chain of

correct edge labels, while the parseval evaluation of edge labels relies on just a single correctly annotated constituent. The last four columns under the heading *split performance* show the same set of evaluation metrics as for the unsplit experiments after applying Treebank Refinement.

The top left cell represents the baseline performance. Unconditionally adding edge labels on input, or including local contextual information via splitting, always lifts performance over this baseline. Adding closed-class lexicalisation to POS labels mostly has a positive effect on annotating functions. The best performance without refinement is obtained by subcategorising all node labels by their edge labels (1.9% over baseline; bottom left cell). Interestingly, performance on embedded constituents does not increase equally, but drops below baseline to 78.7% (last row). Splitting, as an alternative, always increases performance over the unsplit models (*split performance* vs. *unsplit performance*). Especially embedded performance benefits from splitting. The third row (*no edge, no cc*) has the best Parseval results, and almost the highest increase:  $F_{lab}$  on plain node labels increases by 4.4%, and embedded performance even by 5% (best shown in bold). While plain parseval performance also benefits from splitting with hard-wired edge labels on input (last row), the increase is much lower, and also the absolute performance is lower than when splitting is applied to original node labels. Embedded performance is also lifted beyond the unsplit case, but turns out to be more than 7% lower than without hard-coded edge labels.

Plain Parseval performance using optional information for splitting (first column *att*) is between zero edge input and *fix* edge input. Performance on embedded elements is closer to the best performance than to performance for a priori adding all edge label information to the node labels. Attribute context, however, does not beat just ancestor and descendant contexts according to *plain* Parseval evaluation. This might be due to splitting being uni-directional, and attribute information being too fine-grained for supporting the annotation of constituent structure (too many cases like in figure 8.14, or 8.26(b); possible remedies will be discussed in chapter 9).

The benefit of using edge information via attribute context compared to using no edge information at all is that where attributes are included into node labels, the corresponding edge labels can be retrieved from the node labels. The *parsv* and *dep* columns of the middle *att* experiments show performance on grammatical functions that have been retrieved from the node labels after splitting and that have been selected by Treebank Refinement as attribute contexts. Performance on dependencies is very low, because a single missing edge label breaks all dependencies that rely on the percolation of the corresponding information up the constituent structure. Edge labels will be ignored for attribute context when Treebank Refinement does not de-

tect high deviance in the distribution of productions between their presence and absence, so that constituents without edge labels are likely to occur, producing gaps that break percolation of grammatical function information. The *parstv* column, however, evaluates only the correct detection of those maximal projections that have a grammatical function. These results are less than 3% below the best performing experiment in the last row (56.8% vs. 59.6% in table 8.18). They are slightly below the unsplit case with *fix* edges, but table 8.19 shows that attribute context often gives higher precision than recall. Again, recall is predictably worse because attributes will not always

GF	fix edge unsplit, cc lex			att edge split, cc lex		
	precision	recall	$F_{lab}$	precision	recall	$F_{lab}$
ON	70.4	74.4	72.3	73.2	70.7	71.9
OA	45.5	49.6	47.5	52.6	53.0	52.8
OD	28.7	15.1	19.8	49.4	30.6	37.8
OG	0.0	0.0	0.0	0.0	0.0	0.0
OS	47.7	35.0	40.4	0.0	0.0	0.0
OPP	26.9	30.5	28.6	26.4	30.3	28.2
PRED	39.2	29.8	33.9	31.0	12.6	17.9
all	56.9	57.1	57.0	61.1	53.1	56.8

Table 8.19: Precision and Recall on Constituents with Grammatical Functions

be present in the output (e.g. the function of a subject sentence OS cannot be retrieved at all because it has never been used as an attribute context).

Given that dependencies cannot be anchored in open-class word forms, but only in closed-class word forms that have an impact on constituent structure without edge labels, the rather low figures on disambiguating labelled dependencies are not surprising. The unlabelled dependencies reflect the degree in which structure and grammatical functions are correlated via those preferences that are anchored in single lexical items, as opposed to the disambiguation of labelled functions, which will normally be anchored in more than one lexical item. Table 8.20 shows performance on the same set of dependencies representing grammatical functions, but disregarding the labels. The results show the difference between selecting the proper dependents of the clausal heads (unlabelled) and assigning them the correct function (labelled). Again, splitting uniformly improves performance. The difference between labelled and unlabelled performance indicates that the correct heads of constituents are identified much more frequently than the labelled performance suggests, and that the heads of these constituents are often related to

input		unsplit					
edge	lex	unlabelled			labelled		
		Prec	Rec	$F$	Prec	Rec	$F$
fix		75.8	72.0	73.8	56.6	53.7	55.1
fix	cc	76.9	73.1	75.0	59.0	56.1	57.5
		split					
att		74.7	13.8	23.3	52.1	9.6	16.2
att	cc	33.1	2.1	4.0	23.2	1.5	2.8
fix		77.4	<b>77.0</b>	77.2	59.3	59.0	59.1
fix	cc	<b>78.1</b>	76.8	<b>77.4</b>	<b>61.7</b>	<b>60.8</b>	<b>61.3</b>

Table 8.20: Labelled and Unlabelled Grammatical Function Dependencies

the correct clausal heads. The claim that heads of constituents are often correctly identified is also backed by the *not/10* performance for splitting with attribute context in table 8.18. Joining the correct *level 0* nodes to complex structure is the prerequisite to selecting one of them as the head of the resulting complex constituent. Many of the resulting constituents will be complex in the sense of containing recursive node labels. The claim that these heads are related to the proper clausal heads is also backed by the ability to assign the proper clause structure. Clause structure in TüBa-D/Z is given by topological field structure, and performance on topological fields parsing reaches 90%  $F_{lab}$ , which is an increase of 10% over the baseline (table 8.21).

input		unsplit			split		
edge	lex	Prec	Rec	$F_{lab}$	Prec	Rec	$F_{lab}$
		79.1	80.9	80.0	<b>89.7</b>	<b>90.8</b>	<b>90.2</b>
	cc	82.5	83.9	83.2	89.3	90.4	89.9
att		as above			89.6	89.9	89.7
att	cc				89.1	89.4	89.3
fix		87.5	82.4	84.9	89.2	84.3	86.7
fix	cc	87.3	82.3	84.7	88.9	84.0	86.4

Table 8.21: Topological Field Parsing Parseval Performance

The evaluation of grammatical functions annotation gives a more favourable impression of the utility of closed-class lexicalisation: all Parseval evaluations of grammatical functions annotation benefit from it. Splitting also never fails to improve performance of the assignment of grammatical functions. Attribute context never beats pure ancestor and descendant context for *plain* evaluation, and it never beats *fix* inclusion of all edge labels in con-

junction with splitting on functions evaluation. It does, however, beat both alternative combinations, i.e. it beats *fix* inclusion of edges on *plain* Parseval, and no additional information on functions evaluation. It thus constitutes a compromise that can be achieved fully automatically. Best performance with respect to a certain evaluation is achieved when input information is most similar to the desired output information in conjunction with Treebank Refinement.

## 8.6 Evaluation on Final Data Sets

Evaluation so far has heavily explored the tune data set. We will therefore continue with experiments that are performed on the final test data that has remained untouched so far. We repeat the experiments that performed best overall according to Parseval  $F_{lab}$  (no edge,  $-cc$ ), those that scored best according to dependency evaluation (fix edge,  $+cc$ ), and which showed a small drop in the former plus acceptable unlabelled precision on grammatical functions (att edge,  $-cc$ ). Lexicalisation is kept identical for the tune and final data sets of the (fix edge,  $+cc$ ) experiments, which has the advantage that perplexities are more comparable. All results are quite stable between experiments. While Parseval  $F$  tends to be lower on the final data set, and more so on the final data set of unrestricted length, the dependency evaluation seems to benefit from more training data and increases over the results obtained for the tune data set (recall that the final data sets include the full tune train and test data). Perplexities drop only for attribute context. Interestingly, baseline performance seems to suffer more for parsing the final data set and longer sentences than all results obtained after Treebank Refinement. While the number of rules grows for the larger final data sets, the coverage is about constant, and stays almost complete for all but the fix edges experiments.

## 8.7 Conclusion

In this chapter we have shown how Treebank Refinement introduces various kinds of local context into the node labels of a treebank. We have extended the search space of contexts beyond parent context, using the same objective function as introduced in the previous chapter. We have evaluated the effect of Treebank Refinement on parsing performance with Parseval and dependency metrics. Parseval evaluation following the original node label distinctions has been complemented by separately evaluating the subclasses of node label types that are more frequent and easier to predict.

data set	tune $\leq$ 40 words				final $\leq$ 40 words				final, unrestricted length			
experiment	base	-e/-c	fe/+c	ae/-c	base	-e/-c	fe/+c	ae/-c	base	-e/-c	fe/+c	ae/-c
iterations	n/a	42	47	76	n/a	56	74	97	n/a	62	75	110
Pev $F_{lab}$	83.48	87.91	86.76	87.38	83.12	87.77	87.22	87.33	82.37	87.34	86.57	87.10
Pev $F_{unl}$	87.15	90.02	89.20	89.58	86.89	89.86	89.52	89.44	86.21	89.43	88.90	89.24
GF dep $F_{lab}$	n/a	n/a	61.25	16.24	n/a	n/a	63.31	17.93	n/a	n/a	62.49	18.84
GF dep $F_{unl}$	n/a	n/a	77.43	23.29	n/a	n/a	79.80	25.18	n/a	n/a	79.03	26.75
<i>perp</i>	13.49	10.64	16.55 <sup>a</sup>	11.18	13.85	10.69	16.42 <sup>a</sup>	11.06	13.83	10.64	16.45 <sup>a</sup>	10.86
failed	0	0	13	1	0	0	20	0	0	0	14	0
# rules	3432	5142	10 068	8102	4160	6613	13 632	10 744	4557	7489	14 902	12 390
10 <sup>6</sup> contexts	n/a	2.384	2.384	6.288	n/a	3.367	3.367	8.893	n/a	3.862	3.862	11.050

<sup>a</sup>Terminal POS vocabulary obtained through identically restricted lexical POS splitting.

Table 8.22: Parsing Performance on Tune and Final Data Sets

We have shown that Treebank Refinement improves performance over the baseline in all modes of evaluation, whether according to the plain or more detailed Parseval analysis, or according to dependency analysis of major grammatical functions. It also improves performance when input information varies in detail. The benefit of Treebank Refinement is not limited to performance improvements, though. It produces results that are accessible to human judgement and that can therefore help understanding parsing problems that are caused by the interplay of treebank design and parser. It has become apparent for the annotation scheme of TüBa-D/Z that only a few node label types dominate Parseval results, and that these nodes are closely linked to POS tags. Visualisation of the splits performed by Treebank Refinement also reveals preferences for certain configurations of nodes that would otherwise remain hidden.

Treebank Refinement strengthens connections between nodes by including local context on the one hand, and by changing production probabilities of these new nodes on the other hand. It is an iterative process that can result in chains of connected nodes. These chains of nodes have been presented as *local context* that can be depicted by graphs covering the nodes that are connected via contexts, or preferred productions. The contexts that are included into the nodes of TüBa-D/Z via Treebank Refinement rarely exceed the size of two nodes for those parameter sets that perform best. Including context into node labels means that the modified focus nodes never occur without their context, and it seems that the size of such a context is limited. On the one hand, the search space of Treebank Refinement could be extended even further to detect connections between more distant nodes. On the other hand, limiting the amount of context seems preferable when the resulting treebank is only considered as cleaner with respect to overly general node labels, and not as a perfect parsing model in its own right. The application of Treebank Refinement only alters node labels and leaves the nodes and the edges that connect them unchanged, so that we prefer to see it as a preprocessing step to those approaches to parsing that are able to handle more distant relations and unseen events more gracefully than a plain PCFG.



## Chapter 9

# Handling Sparse Data

Sparse data is often a problem when natural language is processed. Data is sparse when examples for different types of events are rare, so that with a given number of examples, you cannot predict reliably which events to expect in the future. Approaches that infer occurrence probabilities from manually augmented data like we do in Treebank Refinement have to invest considerable labour into producing such data. The natural preference thus is to reduce the required amount of training data as much as possible.

If the number of different types of events is limited, and the distribution of probabilities is even, then it is rather easy to predict reliably how often a type of event will be observed in the future. Natural language, however, often shows a high, or even a theoretically unlimited number of different types of events. Moreover, the probability of a few types of events is typically very high, while very many types of events occur only rarely (the distribution is *skewed*). The distribution of word forms is a typical case in point. Each day, thousands of new word forms are being coined that will find their way into training data for machine learning only slowly, if at all (see e.g. Lemnitzer, 2003, for German). On the other hand, function words like articles or pronouns will occur in virtually any sentence of German, with little variation over time. These differences have been our motivation for parsing on partly lexicalised POS tags, where the small (closed) class of high-frequent function words is partly made available to the syntactic parser, while the large (open) classes of e.g. nouns and adjectives are handled by the POS tagger, which assigns them general POS tags. The open word classes are the major source of sparse data on the lexical level, and these problematic classes appear only as a closed class of more general word-class information to the parser.

The problem of sparse lexical data is thus moved out of direct scope of our syntactic parser. Problems of data sparseness still remain, though. Treebank Refinement specialises a treebank for PCFG parsing by incorporating con-

textual information into node labels. PCFGs consider node labels as atomic pieces of information (as opposed to e.g. unification grammars; Johnson, 2003), so that the number of different types of node labels has to increase to accommodate additional contextual information. Treebank Refinement as we have seen it so far only splits node labels, so that the number of examples per focus node in context necessarily decreases for all modified nodes, and the number of different rewritings of a non-terminal is increased when any of its children is modified. The number of rules, which always increases after Treebank Refinement, reflects these changes.

The motivation for Treebank Refinement indeed is to find subclasses of node labels that are distinguished by their behaviour in a certain context, which is distinct from that of the original more general node label. The curves of either overall Parseval performance (as shown in e.g. figure 7.13) or for individual node labels (e.g. in figure 8.25) all show a characteristic levelling out after an initial increase. Parseval metrics average over correctness of all nodes in a treebank, so that nodes that are annotated correctly more frequently have a higher impact than those which are less frequent. The deviance metric  $D_{10}^{SDP}$ , which we generally employ, prefers more frequent deviant node labels. The curves thus level out, because the frequency of more specialised subclasses of node labels decreases, so that the positive effect of each new node label on parsing performance also decreases. At some stage, new node labels will become so specific that they cannot be applied in parses of unseen data, so that another, less appropriate node label has to be used instead.

All curves of Parseval performance that we have shown do not drop sharply in performance during the iterations performed until our stopping condition is met. It is likely, though, that at some stage, tree fragments are created via connected contexts that formulate conditions on input symbols that are met only rarely in novel data. The experiments carried out on words as input symbols in section 8.3.1 may serve as an extreme example, where specialised noun phrase labels were created for sets of words that were observed only once in the majority of cases (see table 8.9). These over-specific node labels seem to be a reason for poor performance, which in this experiment is also accompanied by reduced coverage. Most other experiments show just below, or exactly 100% coverage, however, so that specialisation is not excessive and at least one parse can be produced. We can therefore concentrate on evaluation metrics judging the quality of annotation to judge the benefit of Treebank Refinement for most experiments.

Still, there is the latent danger of splitting nodes too often, as exemplified by those splits of NX that are neutral with respect to their annotation performance (figure 8.26(b) and repeated below). This chapter tries to show

how to undo or prevent those splits that are harmful for parser performance. We will first discuss a merge operator that merges nodes originating in previous splits, and a variant of generating contexts, so that splits can look into the splitting history of contextual nodes, which also can help reducing the number of different node labels. Both approaches try to attack the problem of sparse data arising from continuously splitting node labels.

## 9.1 Merging Similar Nodes

Bockhorst and Craven (2001) propose to refine a PCFG so that it reflects observed probabilities of strings more faithfully. They do not only propose a split operator, which has been the inspiration for our own Treebank Refinement split operator. They also propose to complement splitting by a merge operator. Splits add node labels where a single node label covers distinct usages which can be distinguished by a larger context. The motivation for a complementary merge operator is to merge node labels that behave similarly, and thus to reduce the number of different types of nodes in order to increase the number of examples per class. Although Bockhorst and Craven suggest to use such a merge operator, they do not specify or discuss it in detail.

We will now describe such a merge operator to complement splitting in Treebank Refinement. Following the abstract definition of the split operator, we are looking for a function that merges two types of nodes that behave identically or very similarly despite their differing labels. We have already seen examples for splits that are likely to be too fine-grained to improve parsing performance on our task of annotating all bracketings correctly in data labelled with node labels unadorned by edge labels. Figure 9.1 repeats details from an experiment we have seen in section 8.4.4, where attribute descendant contexts have triggered splits of NX nodes (see figure 8.26). Each

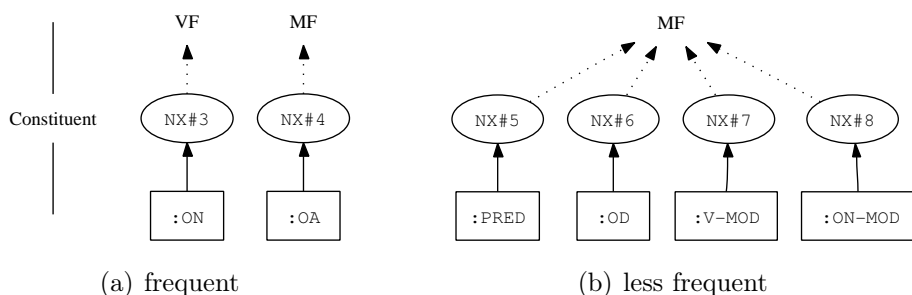


Figure 9.1: NX Splits Triggered by Grammatical Functions

subclass of NX originating in a split has the same property of appearing

preferably in either the middle or initial fields (MF/VF). All four splits shown in figure 9.1(b) do not improve performance on NX (see figure 8.25), and they also do not improve overall Parseval performance (figure 8.27), despite being performed in rather early iterations. The two splits shown in figure 9.1(a), in contrast, improve both Parseval performance on NX and overall Parseval performance. A major difference between these two sets of splits is their frequency, which is between five and forty times higher in the first set than in the second set (see table 8.16). The second set thus seems to include good candidates for merging when edge labels are not considered in evaluation. Performance on plain node labels may be increased by merging nodes NX#5 to NX#8 into a single new node that generalises their property to appear preferably in the middle field.

### 9.1.1 Similar Parents and Unmergeable Nodes

A problem appears when the example just outlined is compared with our short definition of merging just given, which just requires similar behaviour, which could translate into similarity in either ancestor or descendant context. We would like to let a merge operator see non-terminal symbols as atomic symbols, just like the split operator and PCFGs do, where node labels do not carry any further information besides being identical to, or different from other node labels. However, other nodes can be imagined to show a similar preference for middle field parents as the NX presented above, and thus may also be considered to be similar. The last five prepositional phrase labels (PX) shown in figure 9.2 produced in the same experiment as those in figure 9.1, e.g., show a similar preference for appearing below MF nodes. Similarly, PX

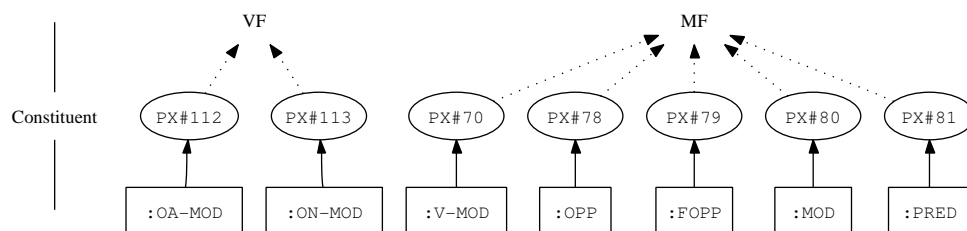


Figure 9.2: PX Splits Triggered by Grammatical Functions

with ON-MOD and OA-MOD edge labels resemble NX with ON in their preference for appearing in the initial field (VF). There are only few field labels, and most constituents normally occur either in the initial or middle fields, as just seen for PX and NX nodes. As a consequence, many rather different classes of

nodes may be merged, including most other classes of non-verbal constituents carrying grammatical function edge labels, when we use the distribution of parents of nodes with a grammatical function as a basis to determine similarity. It does not seem to be desirable from a linguistic point of view to merge such a wide variety of nodes, and it will not help us to recreate labelled gold annotation, either. We thus need to restrict our notion of similarity driving merging in some way. A rather simple restriction is to impose the same perspective as ancestor context, and judge similarity of nodes by the similarity between the distributions of sequences of their children, which show greater variety than the distributions of parents of a focus node in descendant context. Similarity of nodes is then based on distributions with more types of different events.

The untransformed treebank gives us a set of node labels that have been chosen in accordance with linguistic assumptions about the syntactic categories of constituents, fields, and clauses. We accept that these distinctions should not be undone, because we assume that the gold standard defines our parsing goal. We would thus like to prevent merging of any initial categories. This can be achieved by restricting merges to only those nodes that have been split before. This restriction complies with our declared goal of undoing the detrimental effects of overly fine-grained splits.

Our deviance metrics compare pairs of distributions. When more than two nodes are tested for similarity, we establish an overall similarity of a set of nodes as the average similarity between all pairs in the set of nodes. We also require that the deviance of no single pair exceeds a certain threshold. We thus also require that the set of similar node labels contains the full transitive closure of mutually similar node labels.

### 9.1.2 Similarity of Split Nodes

We have chosen Skew Divergence as a deviance metric for splitting, which has the advantage over Kullback-Leibler Divergence that it is also defined when the domains of the distributions that we seek to compare are not identical, and even when neither is a subset of the other. Even though the Kullback-Leibler Divergence may be defined to be either zero or infinite when one or the other domain is undefined (see section 7.3.5), Skew Divergence has the advantage that differences caused by gaps in either domain are smoothed and do not cause infinite or zero difference. We have used the  $D_{fmin}^{SDP}$  metric for determining divergence between distributions, which weighs Skew Divergence with the frequency of focus nodes in context, so that those that are both very frequent and very deviant receive highest scores. For merging, we are looking for most similar nodes instead. When we use  $D_{fmin}^{SDP}$  to determine

very similar nodes instead of very different nodes, and lower values of the metric translate to higher similarity, then lower frequency also translates to higher similarity. Our main target is to find merges between nodes that improve context-free probabilistic parsing according to our evaluation metrics. Nodes that show a similar distribution of sequences of children, and that are moreover very frequent, have a high impact on parsing performance. On the other hand, our main motivation for merging similar nodes is that these nodes may be low-frequent by-products of our split metric, which uni-directionally increases the number of node types and thus reduces their average frequency. It is thus unclear whether frequencies should have the same impact on the deviance metric for merging as they have in  $D_{f_{min}}^{SDP}$ , or none at all. We therefore merge using both  $D_{10}^{SDP}$  in the same way as we use it for splitting, and the corresponding unweighted metric  $D^{SD}$ , and compare results. A third option would be to let higher frequencies further reduce the deviance metric, which we do not test here.

We choose those experiments as our starting point that have been split extensively and that have shown best performance, which includes the experiments with or without attribute context, and with or without lexical POS splitting. We do not consider the experiments based on node labels decorated with edge information before splitting, because our merge operator cannot undo the inclusion of these edge information, as opposed to edge information added during splitting via attribute context. We find that for these four experiments, the pair with the lowest deviance has 0.06 for the *edge -*, *lex cc* experiment (rows *first avg. merge*  $D^{SD}/D_{10}^{SDP}$  in tables 9.1 and 9.2). The other experiments all start with a value closer to the deviance of the last split (column *last split* vs. *last avg. merge*). Unweighted final  $D^{SD}$  decreases from left to right, because the attribute context and lexical POS splitting seem to detect more instances of deviant nodes in context, so that lower  $D^{SD}$  still beats the threshold for splitting due to higher numbers of occurrences, as splitting is performed with  $D_{10}^{SDP}$ , where higher frequencies increase deviance. The search effort is always equal for all experiments in tables 9.1 and 9.2, because the number of nodes that have to be mutually compared is 217 221 for all of them.

We definitely do not intend to undo exactly the splits we have introduced before, so that we only merge sets of nodes with an average similarity that is lower than the divergence of the last split performed on the underlying data set. The threshold for splitting that we have used in the underlying experiments is 350, which we use as a threshold for merging in the experiments using the same metric ( $D_{10}^{SDP}$ ) as well. The experiments using the unweighted metric  $D^{SD}$  use thresholds derived from the last split  $D_{10}^{SDP}$  values given in the row *last split*  $D^{SD}$  in table 9.1.

There are only few iterations in all experiments where sets of similar

edge	–	–	att	att
lex	–	cc	–	cc
last split $D^{SD}$	4.15	2.89	1.22	0.96
first avg. merge $D^{SD}$	0.82	0.06	0.31	0.31
last avg. merge $D^{SD}$	4.11	2.33	1.15	0.84
iterations	7	6	7	5
nodes merged	15	13	20	16
# rules	5066 (–76)	5917 (–52)	7720 (–298)	8985 (–155)
<i>perp</i>	10.64 (±0)	15.84 (–0.02)	11.25 (+0.07)	16.38 (+0.05)
Pev $F_{lab}$	87.9 (±0)	87.8 (±0)	87.2 (–0.2)	87.4 (+0.2)
failed	0 (±0)	13 (±0)	1 (±0)	14 (±0)

Table 9.1: Merging with  $D^{SD}$  after Splitting

edge	–	–	att	att
lex	–	cc	–	cc
last split $D_{10}^{SDP}$	353	354	353	492
first avg. merge $D_{10}^{SDP}$	117	21	144	82
last avg. merge $D_{10}^{SDP}$	193	117	303	303
iterations	2	3	3	6
nodes merged	4	6	12	18
# rules	5134 (–8)	5958 (–11)	8049 (–53)	9048 (–82)
<i>perp</i>	10.64 (±0)	15.86 (±0)	11.19 (+0.01)	16.34 (+0.01)
Pev $F_{lab}$	87.9 (±0)	87.8 (±0)	87.4 (±0)	87.4 (+0.2)
failed	0 (±0)	13 (±0)	1 (±0)	14 (±0)

Table 9.2: Merging with  $D_{10}^{SDP}$  after Splitting

nodes are merged. Performance after the first iteration (not shown here) as measured by cross-perplexity, Parseval  $F_{lab}$ , and the number of rules, shows the same tendency as the results after all iterations according to the tables. Changes relative to the underlying experiments are shown in brackets and small numbers. We note that apart from the reduction in the number of rules, no other evaluation criteria show major changes. Reduction in the number of rules is higher after all iterations than after the first iteration for all experiments. Reduction is also always higher for experiments that employ the unweighted metric  $D^{SD}$ .

The two pairs that show highest similarity among all sets of merged nodes are both in the *edge -*, *lex cc* column. For the  $D^{SD}$  metric, verbal complexes (VC) are merged that have been previously split in two different contexts. The two splits have marked verbal complexes either dominated by a relative clause (R-SIMPX) node, where the verbal complex contains a finite verb except for some annotation errors and ellipses, or verbal complexes dominating a finite verb, where the verb complexes are usually part of a clause in the final or the initial fields (NF/VF). Thus, both subtypes of VC mostly occur in subordinate clauses. Merging these two subtypes of verbal complexes accomplishes our goal of undoing splits generating distinct labels for nodes with similar behaviour. The second most similar pair of labels that is merged are subcategories of adverbial (ADVX) and prepositional constituent labels. The PX subtype dominates almost exclusively words tagged as adverbial interrogative or relative pronouns marked as heads (PWAV+HD) when appearing in the complementiser (C) field. Constituents labelled ADVX are found in the C field very often when they contain a PWAV marked as head. Merging these ADVX and PX seems to be defensible looking at their behaviour, but we do not desire merging any categories defined in the original treebank.

There is only one other pair of originally different node labels that is merged. Both finite and non-finite verbal chunks (VXFIN/VXINF) often dominate a single finite verb when they are part of a coordination (KONJ edge label; last merge of the *edge att* experiments using the  $D_{10}^{SDP}$  metric). Note that POS tags have been assigned automatically, and that the difference between some finite and non-finite verbs often confuses automatic POS taggers in German (Müller and Ule, 2002). Coordination poses a further problem, so that the corresponding distribution of finite and non-finite POS tags shown in table 9.3 probably results from the inability of the POS tagger to disambiguate verbal finiteness in certain situations (sorted by overall frequency; only frequencies above zero shown). Despite being originally different categories, merging VXINF and VXFIN can probably be justified in this specific situation, because both categories obviously behave rather similarly in an automatically POS-tagged treebank. The table also highlights another as-



	VXFIN	VXINF
VVPP+HD	2	34
VVFIN+HD	17	7
VVINF+HD	3	15
VVINF+HD	0	15
PTKZU+HD VVINF	0	6
VMFIN+HD	5	0
ADJD+HD	0	4
NN+HD	1	3
VAFIN+HD	4	0
VVIZU+HD	2	2

Table 9.3: Distribution of Children of Coordinated VXFIN or VXINF

pect of specialising the node labels of a treebank. The ability to specialise on errors of the new combined VXFIN/VXINF node label can be used as a goal in itself to spot errors in the annotation of the treebank. We will try to examine this behaviour of the approach for spotting annotation errors in more detail in chapter 10.

Merging these verbal chunks is performed last in both experiments, and the similarity between the nodes is rather low according to the metric (row *last avg. merge* in table 9.2). Looking at the many gaps in both distributions of VXFIN and VXINF, table 9.3 also indicates that similarity is not very high anymore, despite the low number of merged nodes (row *nodes merged* in table 9.2), so that merging seems to apply only rarely.

Algorithm 4 shows pseudocode of the merging algorithm. The function  $\text{TRANSITIVECLOSURE}(f, \mathcal{M}, d)$  returns the transitive closure of nodes in  $\mathcal{M}$  including  $f$  that are mutually at least as similar as  $d$  (see algorithm 6). If the transitive closure is not a subset of  $\mathcal{M}$ , then it returns the empty set. Two nodes  $f_j$  and  $f_k$  are similar if  $\text{MAXDEVIANCE}(f_j, f_k) \leq \check{d}$  (see algorithm 5). The function  $\text{RENAMEFOCUSINCONTEXT}(\mathcal{M})$  in algorithm 4 is called to assign new labels to all merged nodes. It is redefined from the function in algorithm 1 so that it assigns all nodes in  $\mathcal{M}$  a single new unique node label. We do not claim that the algorithms shown here are optimal. They closely resemble our current implementation.

We have shown that merging nodes is feasible, and that the number of rules, which corresponds to the degree of fragmentation of training data, can be reduced by merging. However, the reduction achieved in experiments is rather small, and other evaluation metrics show only minimal changes, and not always to the better.

**Algorithm 4** Iteratively Merging Similar Nodes

---

```

1: repeat
2:    $\check{d} \leftarrow$  maximum deviance threshold
3:   for all  $f_1 \in \mathcal{F}$  do ▷ See text for initial restrictions on  $\mathcal{F}$ 
4:     for all  $f_2 \in \mathcal{F} \setminus \{f_1\}$  do
5:        $d \leftarrow$  MAXDEVIANCE( $f_1, f_2$ )
6:       if  $d \leq \check{d}$  then ▷ Find all pairs below threshold
7:          $\mathcal{M} \leftarrow \mathcal{M} \cup \{f_1, f_2\}$ 
8:       end if
9:     end for
10:  end for
11:   $d_0 \leftarrow \check{d}$ 
12:  for all  $f \in \mathcal{M}$  do
13:     $\mathcal{M}_f \leftarrow$  TRANSITIVECLOSURE( $f, \mathcal{M}, \check{d}$ )
14:    if  $\mathcal{M}_f \neq \emptyset$  then
15:       $n \leftarrow \frac{1}{2} |\mathcal{M}_f| (|\mathcal{M}_f| - 1)$  ▷ Number of different pairs in  $\mathcal{M}_f$ 
16:       $d \leftarrow \frac{1}{n} \sum_{f_1 \in \mathcal{M}_f, f_2 \in \mathcal{M}_f \setminus f_1}$  MAXDEVIANCE( $f_1, f_2$ )
17:      if  $d \leq d_0$  then ▷ Determine lowest average deviance
18:         $(d_0, f_0) \leftarrow (d, f)$ 
19:      end if
20:    end if
21:  end for
22:  if  $\bigcup_{f \in \mathcal{M}} \mathcal{M}_f \neq \emptyset$  then ▷ At least one mergeable set found
23:    RENAMEFOCUSINCONTEXT( $\mathcal{M}_{f_0}$ )
24:  end if
25: until  $\bigcup_{f \in \mathcal{M}} \mathcal{M}_f = \emptyset$ 

```

---

**Algorithm 5** Maximum Pairwise Deviance

---

```

1: function MAXDEVIANCE( $f_1, f_2$ )
2:    $d_{12} \leftarrow D(f_1 \| f_2)$ 
3:    $d_{21} \leftarrow D(f_2 \| f_1)$  ▷  $D()$  may be asymmetric
4:   return  $\max\{d_{12}, d_{21}\}$ 
5: end function

```

---

**Algorithm 6** Transitive Closure of Similar Nodes

---

```

1: function TRANSITIVECLOSURE( $f, \mathcal{M}, d$ )
2:    $\mathcal{M}_f \leftarrow f$ 
3:    $\mathcal{M}_{new} \leftarrow \emptyset$ 
4:   repeat ▷ Find all nodes in  $\mathcal{M}$  (transitively) similar to  $f$ 
5:      $\mathcal{M}_f \leftarrow \mathcal{M}_f \cup \mathcal{M}_{new}$ 
6:     for all  $f_1 \in \mathcal{M}_f$  do
7:       for all  $f_2 \in \mathcal{M} \setminus \mathcal{M}_f$  do
8:         if MAXDEVIANCE( $f_1, f_2$ )  $\leq d$  then
9:            $\mathcal{M}_{new} \leftarrow \mathcal{M}_{new} \cup \{f_2\}$ 
10:        end if
11:       end for
12:     end for
13:   until  $\mathcal{M}_{new} = \emptyset$ 
14:   for all  $f_1 \in \mathcal{M}_f$  do ▷ Check all mutual similarities
15:     for all  $f_2 \in \mathcal{M}_f \setminus \{f_1\}$  do
16:       if MAXDEVIANCE( $f_1, f_2$ )  $> d$  then
17:         return  $\emptyset$ 
18:       end if
19:     end for
20:   end for
21:   return  $\mathcal{M}_f$ 
22: end function

```

---

## 9.2 Splitting with Context History

Most restrictions that we impose on the merge operator try to put the most interesting group of candidates into focus. These interesting candidates are those that have been split previously from a common type of node for different reasons, mainly because splitting as we use it considers a single context at a time, so that it cannot detect several contexts at once that have a similar effect on the focus node's behaviour. After these individual splits, the new node labels are not related any more for PCFG treebank grammars generated from the transformed treebank, and examples are unnecessarily sparse for the subclasses of the original node label. Undoing splits is a dangerous business, because it may cross boundaries we would like to keep (hence our restrictions), but also because merging of nodes and undoing previous splits is a combination that could be desirable after each iteration, so that new splits are based on the largest possible classes of node labels. This closer combination is dangerous, however, because it is immediately clear that alternating splits and merges can easily enter an infinite loop without further restrictions.

We propose another solution to handle overly split node labels that avoids these problems. Instead of breaking up merging and splitting into two separate operations, we base splits on the treebank in several earlier states in addition to the current state. We allow the splits to be triggered by contexts which have been modified by previous splits as we have done so far, or also by more general contexts, by undoing some of the splits *during searching* for contexts. The only requirement on the generalisation of the context is that changes to the context that have occurred *before* the creation of the focus node cannot be undone. All changes to the context that have occurred *after* the creation of the focus node can be undone without harming the guaranteed stopping condition. In the beginning, most nodes are still unsplit, so that each context which is considered during the search for deviant nodes in context is only accompanied by other contexts from the context history as soon as nodes change in it.

When we have a focus node in context  $(c_n, c_{n-1}, \dots, c_2, c_1, f)$  as we know it from chapter 8, then  $\mathcal{C}_f = (c_n, c_{n-1}, \dots, c_2, c_1)$ , as before, is the first context for  $f$  for which the distribution of productions over the whole treebank is collected in order to judge its deviance from  $f$  without a context. This context is the only context that we have considered so far. The *context history* now strips the most recent change from  $\mathcal{C}_f$  and again records the distribution of productions over the whole treebank for  $f$  in this context, or in a context where the context history contains the context we are looking for. The history of the context only goes back as far as the iteration in which  $f$  has been last

modified. Otherwise, the same splits could be repeated infinitely, which we want to avoid.

Given  $f$  has been split in iteration  $i_f$ ,  $c_1$  in iteration  $i_{c_1}$ , and so on, and  $c_j^i$  is the context node  $c_j$  with all splits undone that are more recent than iteration  $i$ , then the set of contexts  $\mathcal{C}_f$  as we have used it in algorithm 2 is extended to  $\mathcal{C}_f^h$  with new contexts from context history, so that

$$\mathcal{C}_f^h = \{ (c_n^i, c_{n-1}^i, \dots, c_2^i, c_1^i) \mid i_f < i \leq \max \{ i_{c_1}, \dots, i_{c_n} \} \}$$

Each context taken from the context history (i.e. where  $i < \max \{ i_{c_1}, \dots, i_{c_n} \}$ ) gives possibly access to more examples from the treebank hidden by later splits. The remaining algorithm can be used as before.

Experiments confirm that initially, the splits are identical to those obtained without looking into the context history, but after a number of iterations, there seem to be more general splits when context history is considered. We repeat the same experiments as we have used as a basis for merging, and that have shown most promising results for Parseval and dependency  $F_{lab}$  before, now considering context history (see table 9.4). We do not repeat here those experiments where edge labels are immutably given, because they cannot be undone by context history. We notice that performance is better for most experiments and most metrics with context history, including *perp* between the resulting models and test data, with least impact on Parseval  $F_{lab}$ .<sup>1</sup> We notice a considerable increase in dependency  $F_{lab}$  for the experiment using attribute context and closed-class lexicalisation (column *edge att, lex cc*), where data sparseness is greatest according to the number of rules. The increase in *dep*  $F_{lab}$  in contrast to the rather constant *GF Parseval*  $F_{lab}$  can be explained by the percolation of information necessary for establishing dependencies in constituent trees, which requires an uninterrupted chain of edges including marks for heads, conjuncts, and appositions (HD, KONJ, APP). The individual grammatical functions gain most on recall. Generally, precision is much higher than recall for them (see table 9.5, and also table 8.20). Low-frequent grammatical functions are still out of scope of Treebank Refinement: OG, OPP and OS make up only about 10% of all functions; OG occurs only four times in test data (see frequencies in column *gold*). High-frequent grammatical functions like ON (more than 50% of all functions) benefit most from splitting with context history. There are only two undesirable effects. First, experiments disregarding edges (*edge -*) show an increase in the number of rules. Second, the number of contexts increases that has to be searched. The most important result in the present context, however, is the reduction in

<sup>1</sup>Note that the stopping condition has been set according to the iterative behaviour of Parseval of the *edge -*, *lex -* experiment as described in section 7.4.

edge	–	–	att	att
lex	–	cc	–	cc
iterations	43 (+1)	94 (–5)	71 (–5)	125 (–10)
# rules	5278 (+136)	6378 (+409)	7086 (–1016)	8106 (–1024)
<i>perp</i>	10.63 (–0.01)	15.86 (±0)	10.99 (–0.19)	15.97 (–0.36)
Pev $F_{lab}$	87.8 (–0.1)	88.0 (+0.2)	87.8 (+0.4)	87.5 (+0.1)
GF Pev $F_{lab}$	n/a	n/a	52.0 (+0.9)	57.3 (+0.5)
dep $F_{lab}$	n/a	n/a	15.5 (–0.7)	16.6 (+13.8)
dep $F_{unl}$	n/a	n/a	22.2 (–0.9)	22.3 (+18.3)
failed	0 (±0)	13 (±0)	1 (±0)	14 (±0)
contexts $\leq$	4 232 650 (+1 848 287)	4 521 594 (+2 137 231)	7 792 852 (+1 505 288)	7 723 835 (+1 436 271)

Table 9.4: Relative Changes with Context History

GF	precision	recall	$F_{lab}$	gold
ON	68.6 (+65.1)	9.6 (+9.5)	16.9 (+16.6)	4985
OA	56.0 (+11.0)	17.3 (+11.6)	26.4 (+16.3)	2051
OD	50.0 (+37.0)	9.3 (+8.3)	15.7 (+13.9)	312
OG	0.0 (±0.0)	0.0 (±0.0)	0.0 (±0.0)	4
OS	0.0 (±0.0)	0.0 (±0.0)	0.0 (±0.0)	420
OPP	0.0 (-6.8)	0.0 (-0.9)	0.0 (-1.6)	553
PRED	38.1 (+22.5)	2.7 (+2.1)	5.0 (+3.9)	896

Table 9.5: Precision and Recall of Grammatical Function Dependencies for the *edge att*, *lex cc* Experiment Including Context History

the number of rules in those experiments that originally led to the highest number of rules (*edge att*).

The *contexts* column shows that many more contexts have to be searched than without context history. The numbers refer to the last iteration. They increase per iteration, being identical in the first iteration. Note that while the search effort is higher according to the set of inspected contexts, the experiments often require fewer iterations to complete.

Context history inspects statistics from old states of the treebank in addition to those emerging from splits. Contexts without split nodes from context histories are more frequent, so that they can score higher for deviance metrics that take frequencies into account. An example for fragmentation of data that can be undone is the connection between lexicalised POS occurring in the middle field and their grammatical functions as modifiers (MOD). These modifiers are quite frequently the adverbial phrases *nicht mehr* (‘not anymore’) and *gar nicht* (‘not at all’), as in *Ich kann mir das nicht mehr erklären.* (‘I cannot explain this anymore.’) or *In Frankfurt dagegen wird gar nicht erst abgestimmt.* (‘In Frankfurt, however, elections will not be held in the first place.’). Other less frequent constituents also behave similarly, like *ganz nebenbei* (‘by the way’), *noch nicht* (‘not yet’), *nun wirklich* (‘sure enough’), *immer mehr* (‘more and more’), *noch nie* (‘never so far’), *überhaupt nicht*

(‘not at all’), and others. Including specialised node labels for these special adverbial phrases that are rather frequently used in German allows annotating the function of the corresponding constituents rather easily. Before these modifiers are annotated, a number of other labels are generated by splits, including middle fields in various positions. Middle field nodes are split when they are part of a clause in the final field, or of a coordinated or relative clause, or when coordinated with another middle field. Of all 105 modifiers in all these middle fields in our training data, only 55 occur in middle fields that have not been split before we try to mark *gar nicht* etc. as modifiers. All split middle fields contain about equally many of our modifiers (10 to 17). With context history, the context MF-MOD-ADVX-ADVX > ADV+HD covers all of them in the MF symbol, because statistics for each instance of e.g. MF#117 in MF#117-MOD-ADVX-ADVX > ADV+HD (where MF#117 is part of a relative clause) are all subsumed under MF after going back into the split history and stripping #117. The same experiment without context history fails to state this connection directly. We do not generally include MOD in the evaluation of grammatical functions. For the *edge att*, *lex cc* experiment, MOD achieves a dependency  $F_{lab}$  of 2.7 % without, and 16.8 % with context history (precision jumps from 25 % to 61 %).

Figure 9.3 shows another example with more local consequences. It is

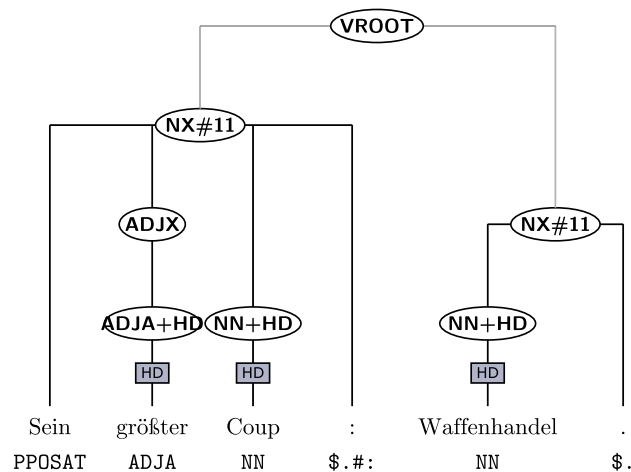


Figure 9.3: Sentence-Final Punctuation Marks Unattached NX

the first split where the experiments with and without context history differ (the eleventh split of non-terminals). The split of NX that dominates a POS tag for sentence-final punctuation (\$.) almost exclusively marks unattached noun phrases. Both experiments with and without context history are based on lexically split POS, where colons (:.) receive their own subclass of the



$\$. \text{ tag}$  (see table 8.13). The experiment using context history can undo this split when gathering the statistics and therefore finds more examples, leading to the split, whereas the experiment without context history suffers from fragmentation of the data, so that it cannot find sufficient examples for a split. Both unattached  $\text{NX}$  in figure 9.3 contribute to statistics for the context  $\$. < \text{NX}$  when context history is considered, whereas  $\$. < \text{NX}$  and  $\$. \# : < \text{NX}$  are strictly different without context history. With context history, the example sentence increases the number of events for  $\$. < \text{NX}$  by two, and for  $\$. \# : < \text{NX}$  by one, so that the special behaviour of the  $\$. \# :$  tag is recorded along with the more general behaviour of the  $\$. \text{ tag}$ .

Our initial problem in this chapter were  $\text{NX}$  and  $\text{PX}$  that have been split according to their grammatical functions, which were not frequent enough individually to improve parsing. Context history cannot undo these splits, but it can help handling phenomena equally in all children of these new nodes. If there was, e.g., a certain adjectival modifier depending on the position of the  $\text{NX}$  in the middle field and common to all  $\text{NX}\#4$  to  $\text{NX}\#8$  in figure 9.1, then with context history a single context  $\text{MF-NX} > \text{ADJX}$  would cover all focus  $\text{ADJX}$  at once despite occurring below different subtypes of  $\text{NX}$ .

## 9.3 Conclusion

We have concentrated in this chapter on sparse data problems that arise specifically where node labels in a PCFG are split to reflect more fine-grained usage patterns. We have given two approaches that can help reducing the number of rules in the treebank grammar read off the modified treebank, where the number of rules is correlated to the number of different node labels. The first approach tries to collect several node labels that behave similarly and merges them to form a single new label. Merging may also apply to originally distinct node label types, so that we restrict it to apply only to previously split nodes. Together with other restrictions on the set of nodes to be merged, the merge operator applies much less frequently than splitting, and achieves a small decrease in the number of rules.

The second approach to reducing the detrimental effect of overly splitting node labels tries to undo earlier splits of node labels already during the splitting process. The splits are undone only for the collection of statistics of upcoming splits, and they are undone only in the contexts, so that each step back into the context history offers larger classes unharmed by splitting, where the steps correspond to earlier iterations. At the same time, more informative but smaller classes produced in later iterations are also available. Using context history during splitting gives a greater decrease in grammar

size in experiments originally leading to larger grammars, and using context history also has an overall positive effect on other evaluation criteria.

We thus think that a more extensive search for contexts to be included into node labels is more promising than undoing splits by merging nodes after splitting them first, especially where extrinsically defined distinctions between node labels should not be undone. Merges between nodes that belong to different linguistically motivated categories usually indicate subclasses of these node labels that tend to behave similarly, or plain annotation errors.

## Chapter 10

# Trebank Refinement Aiding Supervised Annotation

Setting up language resources involves considerable effort, because human intervention is inevitable and costly. Human annotators are essential, because they usually outperform automatic methods in terms of annotation accuracy, but they still make their own kind of errors. In addition to genuine mistakes, they do not always behave identically each time when presented with the same infrequent problem. Thus one can expect a number of errors to be present in any hand-built language resource.<sup>1</sup>

We divide these errors into the following categories: *violations of the annotation guidelines* and *violations of language principles not covered by the annotation guidelines*. Additionally, following Blaheta (2002), errors can also be: *detectable* – errors that are easy to spot and fix by using queries over the annotation that define impossible configurations and transformations for correction; *fixable* – errors which can be found automatically, but that require human intervention for correction; *systematic inconsistencies* – errors which are not covered by the annotation guidelines, or errors not described precisely enough in the guidelines. These two classifications of errors in annotated corpora are orthogonal, but not independent: we can expect errors that are violations of the annotation guidelines to be usually detectable and fixable, and those that are a violation of language principles, but not covered by the annotation guidelines, to be more frequently systematic inconsistencies.

Each class of errors requires a specific way for detection and correction. Detectable errors covered by the guidelines are the easiest in this respect. They can be addressed by encoding the guidelines in a formal way and by testing the corpus for consistency. Detecting the other types of errors requires ad-

---

<sup>1</sup>This chapter is a revised version of joint work with Kiril Simov (Ule and Simov, 2004).

ditional linguistic knowledge. Such knowledge is not always available or easy to acquire, so that other mechanisms are desirable for error detection. We divide those methods into *symbolic* and *non-symbolic* approaches. The *symbolic* approaches are based on (linguistically motivated) pattern matching selecting possible deviations from linguistically correct occurrences. Patterns can be devised by human annotators, or they can be extracted (semi-)automatically from the corpus itself. The *non-symbolic* approaches use statistical methods to find rare events in the annotated corpus, where an event is a certain fragment of the annotation. In general, these methods can find errors in each of the above categories, but they are especially useful when pattern-based approaches are not easily applicable, because patterns are difficult to find.

We present such a non-symbolic method that attacks errors and inconsistencies in structural annotation, and that shows good performance across languages and annotation schemes. We detect errors and inconsistencies that appear as unexpected events in a corpus using Treebank Refinement on artificially introduced errors and apply machine learning to produce fully automatically a list of likely error candidates.

## 10.1 Methods and Data

### 10.1.1 Unexpected Productions

Treebank Refinement aims at spotting productions of nonterminal nodes in treebanks that behave not as expected when they appear in certain contexts. Treebank Refinement looks at all types of nonterminal nodes  $f$  in a treebank and compares the distribution of each of  $f$ 's productions over the whole treebank with its productions when appearing under a certain parent node (the *context* type  $c \in \mathcal{C}_f$ ; we use only parent context here). Treebank Refinement is applied iteratively, and in each iteration it delivers a single type of *focus* node  $f$  that has the most unexpected distribution of productions in a certain context  $c$ . As before, we compare the distributions of the different production types  $p \in \mathcal{P}_f$  of node  $f$ . In order to compare these distributions, we employ the  $\chi^2$  metric, which computes the sum of squared differences between expected and observed frequencies of node type  $f$  having production type  $p$  in context type  $c$ , normalised by the expected frequency (repeated here from equation 7.3):

$$D^{CS} = \sum_{p \in \mathcal{P}_f} \frac{(\text{freq}(c > f \rightarrow p) - \text{exp}(c > f \rightarrow p))^2}{\text{exp}(c > f \rightarrow p)}$$

The  $\chi^2$  statistical test prescribes minimal values for expected and observed frequencies (see section 7.3.1). With lower values, the test yields higher significance than it should, making it statistically unsound. We use this as a feature, and (mis-)employ the  $\chi^2$  test for spotting errors: very unexpected events ( $\text{exp}(c > f \rightarrow p) \ll 1$ ) are rated high even when occurring few times (e.g. when  $\text{freq}(c > f \rightarrow p) = 1$  and  $\text{exp}(c > f \rightarrow p) = 1/1000$ , then  $D^{CS} \approx 1000$ ). An event thus is a (context, focus, production) triple:  $(c, f, p)$ . We argue that very unexpected events, which moreover occur rarely in a corpus, may well be errors. We call these unexpected events *error candidates*.

### 10.1.2 Ranking Error Candidates

Trebank Refinement typically involves several hundred iterations, and each iteration covers a focus node in context with many different productions, yielding a high number of possibly erroneous corpus positions. In order to focus on the most likely candidates, we choose to employ a supervised ML regime using memory-based learning as implemented in Timbl (Daelemans & al., 2003).<sup>2</sup> We introduce artificial errors into the corpus and generate the following features that characterise error candidates from the  $(c, f, p)$  triples, resulting in positive training data:

$\text{freq}(c > f \rightarrow p)$  occurrences observed in the corpus

$\text{exp}(c > f \rightarrow p)$  the expected number of occurrences

$\frac{(\text{freq}(c > f \rightarrow p) - \text{exp}(c > f \rightarrow p))^2}{\text{exp}(c > f \rightarrow p)}$  its contribution to  $D^{CS}$

$D^{CS}$  the overall  $D^{CS}$  over all  $p \in \mathcal{P}_f$

*termratio* fraction of overall  $D^{CS}$  contributed by this triple

*rank* triple is *rank* highest contributor to  $D^{CS}$

*rankratio* the relative position as contributor:  $\text{rank}/|\mathcal{P}_f|$

*alt* the number of other contributors to  $D^{CS}$ , i.e.  $|\mathcal{P}_f| - 1$

$\text{freq}(c > f)$  the number of times  $c$  dominates  $f$

*iter* the iteration of Trebank Refinement detecting  $(c, f, p)$

*iterratio* fraction of *iter* from all Trebank Refinement iterations

---

<sup>2</sup>We use Timbl version 4.3.1 for our experiments.

The motivation for the above list of features is to present all non-symbolic information to Timbl that could be relevant for identifying errors. Output of the ML stage is four classes: error in  $f$ , error in  $p$ , error in  $c$ , or no error. Combinations are not represented as separate classes:  $f$  is also used when an additional error occurs in  $p$  or  $c$ , and  $p$  is also the class for errors in  $p$  and  $c$ . Having more classes or just binary classes did not improve precision but harmed recall on the most reliable focus node.

Our method of error detection is based on the ranking of the error candidates with respect to the parameters provided by Treebank Refinement. We apply ML techniques to support ranking the error candidates because the actual ranking is hard to define explicitly, as there are many dependencies among the parameters. However, recall of ML is rather low overall, so that we rejoin the ML output with all other error candidates. We sort the resulting list of triples so that generally triples marked as errors by ML and occurring less frequently are given first. The sort keys we use are (with matching list items sorted first):

- |   |                                     |   |                                    |
|---|-------------------------------------|---|------------------------------------|
| 1 | $freq(c > f \rightarrow p) \leq 3$  | 6 | smaller $exp(c > f \rightarrow p)$ |
| 2 | ML says focus node error            | 7 | higher <i>rankratio</i>            |
| 3 | ML predicts some error              | 8 | higher <i>termratio</i>            |
| 4 | $rankratio = termratio$             | 9 | lower <i>iterratio</i>             |
| 5 | smaller $freq(c > f \rightarrow p)$ |   |                                    |

The sort keys 2 and 3 account for the machine learner’s reliable classification. Key 4 prefers focus nodes that have few but equally unexpected productions. The other keys generally rank those events higher that are less expected, and that occur infrequently. This sorting combines all information acquired by Treebank Refinement and the machine learner in a single ordered list. The resulting list of error candidates is presented to a human annotator, who has to judge whether the errors are true positives.

### 10.1.3 Early and Revised Annotation

We apply error detection to two manually annotated treebanks: BulTreeBank and TüBa-D/Z (see chapter 4). BulTreeBank (BTB) is an HPSG-based treebank for Bulgarian annotated with detailed syntactic information (Simov & al., 2001). It contains more than 10000 sentences that have been extracted from grammars of the Bulgarian language and from electronic texts. Its annotation scheme is constituency-based. However, each constituent is additionally classified with respect to head-dependant relations like: head-complement, head-subject, etc. Keeping the original word order unchanged, we have introduced discontinuous constituents. The reference interaction

among the constituents is expressed by coreferential relations. In the experiments reported below we use the most elaborated part of the treebank, which consists of 580 sentences.

We use four data sets for our experiments, one from BulTreeBank and three from TüBa-D/Z, consisting of less revised data (*early*), almost finished data (*late*), both representing snapshots of the treebank during development, and release data (*release*), which includes the sentences of the *early* and *late* data sets, but which has undergone more extensive revisions corresponding to the first public release of the treebank (see table 10.1 below for the sizes of the data sets; TüBa-D/Z is further abridged to TZ).

We operate on the *export* data model for both treebanks (see chapter 4), representing linguistic annotation as directed acyclic graphs with labelled nodes and edges, only ignoring secondary edges (see section 5.2). It is significant how the structure of the annotation in a treebank is represented in the export model, because this representation determines the distribution of the relevant events. When generating the export representation of the data sets, we decided to ignore information about grammatical functions in order to overcome sparse data problems caused by infrequent lexical information.

#### 10.1.4 Evaluation via Artificial Errors

Evaluation of methods like ours is a challenge, because the original training material is meant to be error free, and the results can only be evaluated indirectly by manually checking whether the method discovers some errors in the treebank, which only yields precision, but not recall of the method. Thus we need a corpus of errors for training and testing. In order for the resulting corpus to be objective, we decided to introduce artificial errors automatically and randomly by permuting node labels in a given percentage of all nodes. This procedure has the advantage of introducing a set of errors with given properties, such as the number of the introduced errors, their nature (via changing the list of the categories involved), or the places to introduce them (via patterns for selecting a subset of nodes in the treebank).

We are aware that randomly changing node labels does not resemble all kinds of errors equally well, but will be more similar to typing errors (where a wrong label is accidentally selected) than to misinterpreted larger structures. Randomly changed node labels may even be correct according to the annotation guidelines when the guidelines do not prescribe a single solution. It will also be useful, though, when those parts of the guidelines become apparent that allow to annotate a single phenomenon in several ways.

## 10.2 Experiments and Results

We perform two sets of experiments that concentrate on the ability of error detection to spot artificial errors, and on its ability to spot proper errors in the original data. For evaluating the ability of error detection to detect artificial errors, we inject errors into 1%, 0.1% and 0.01% of the nodes in the BulTreeBank, and TüBa-D/Z *early/late/release* datasets. Treebank Refinement is applied to these twelve data sets with the stopping condition of  $\alpha < 1$ .<sup>3</sup> Each resulting list of error candidates is classified by ML using ten-fold cross-validation and then sorted to produce a ranked list as explained above. Table 10.1 shows the actual number of artificial errors introduced into the datasets and the overall number of these errors covered by the full list produced by error detection, i.e. all errors present in some part of the  $(c, f, p)$  triples in the list. Table 10.2 shows precision and recall of the ML stage for the *focus* error class.

	BTB	TZ <i>late</i>	TZ <i>early</i>	TZ <i>release</i>
sent.	580	3074	7398	15260
nodes	15013	56601	132640	318596
1.00%	155/44	579/362	1279/856	3168/2641
0.10%	12/5	57/38	125/90	306/246
0.01%	2/0	10/6	10/8	35/26

Table 10.1: Artificial Errors Introduced/Detected

	BTB	TZ <i>late</i>	TZ <i>early</i>	TZ <i>release</i>
1.00%	0.42/0.35	0.72/0.72	0.60/0.65	0.68/0.69
0.10%	0.0/0.0	0.49/0.59	0.44/0.61	0.73/0.69
0.01%	0.0/0.0	0.0/0.0	0.0/0.0	0.30/0.30

Table 10.2: ML Precision/Recall for *focus* Errors

In addition to the number of errors present in the error detection list, it is most relevant how much human labour is needed to decide whether an error candidate is an actual error.<sup>4</sup> Given that human labour involved in finding an error is proportional to the number of corpus positions that have to be checked to find a true error, figure 10.1 shows the amount of labour

<sup>3</sup>In a statistically valid experiment,  $\alpha$  is the certainty that the null hypothesis of two distributions being equal can be rejected.

<sup>4</sup>Note that a single  $(c, f, p)$  can occur many times in a corpus.



necessary to find a certain fraction of the artificial errors when 1%, 0.1% and 0.01% of the nodes are relabelled incorrectly. They plot the number of wrongly proposed error candidates per correctly identified error, i.e. the number of corpus positions without an error you have to check manually until you find a true error. The x-axis shows the percentage of all artificial errors covered by the top ranks of the list so far when going through the list top-down. Figure 10.1 shows that for all combinations of size and language of treebanks, and for all relative numbers of artificial errors, error detection points to many true artificial errors first. It shows good performance in that for spotting up to 25% of all errors, you have to check at most ten corpus positions per error. In most cases you find more than 50% of the errors by looking at two corpus positions per error. The method seems to be applicable already to relatively small corpora (BulTreeBank), and it performs well for unfinished (TüBa-D/Z *early*) as well as for highly edited data (BulTreeBank and TüBa-D/Z *release*). As expected, it seems to be easier to spot errors in cleaner and larger data sets.

Figure 10.2 shows the more standard ROC curve representation for ranking classifiers (Fawcett, 2003). ROC curves plot true positives on the Y axis against false positives on the x-axis. A perfect classifier starts from the origin and proceeds to 100% true positives without ever producing any false positives. ROC curves are especially useful to check whether a classifier can distinguish reliable from less reliable answers. When it knows which answers are reliable, it will give them first, and return a high number of true positives per false positive. Only later, when the classifier has exhausted all its reliable answers, it goes on with the less reliable classifications, resulting in more false positives per true positive. Such a conservative behaviour corresponds to a steep initial increase, and is especially desirable when there are large numbers of negative instances (Fawcett, 2003, p. 5). This is the case for the problem at hand, where we have up to 99.99% negative instances, and we are looking for a classifier that can distinguish the few positive instances (i.e. the errors) from the large number of negative instances (all other nodes). The curves in Figure 10.2 show very good performance initially. They also show a tendency to allow spotting more of the introduced errors for larger data sets.

The second set of experiments tries to evaluate the relevance of error detection for detecting proper errors in the original data sets. For one of the above experiments (BulTreeBank, 0.1% artificial errors) we checked whether there are genuine errors in the original data among the first candidates of artificial errors. We found that among the first 27 error candidates, there were 11 such errors. The remaining 16 candidates include five examples where the guidelines are unclear, and eleven productions that are correct, but rare. We

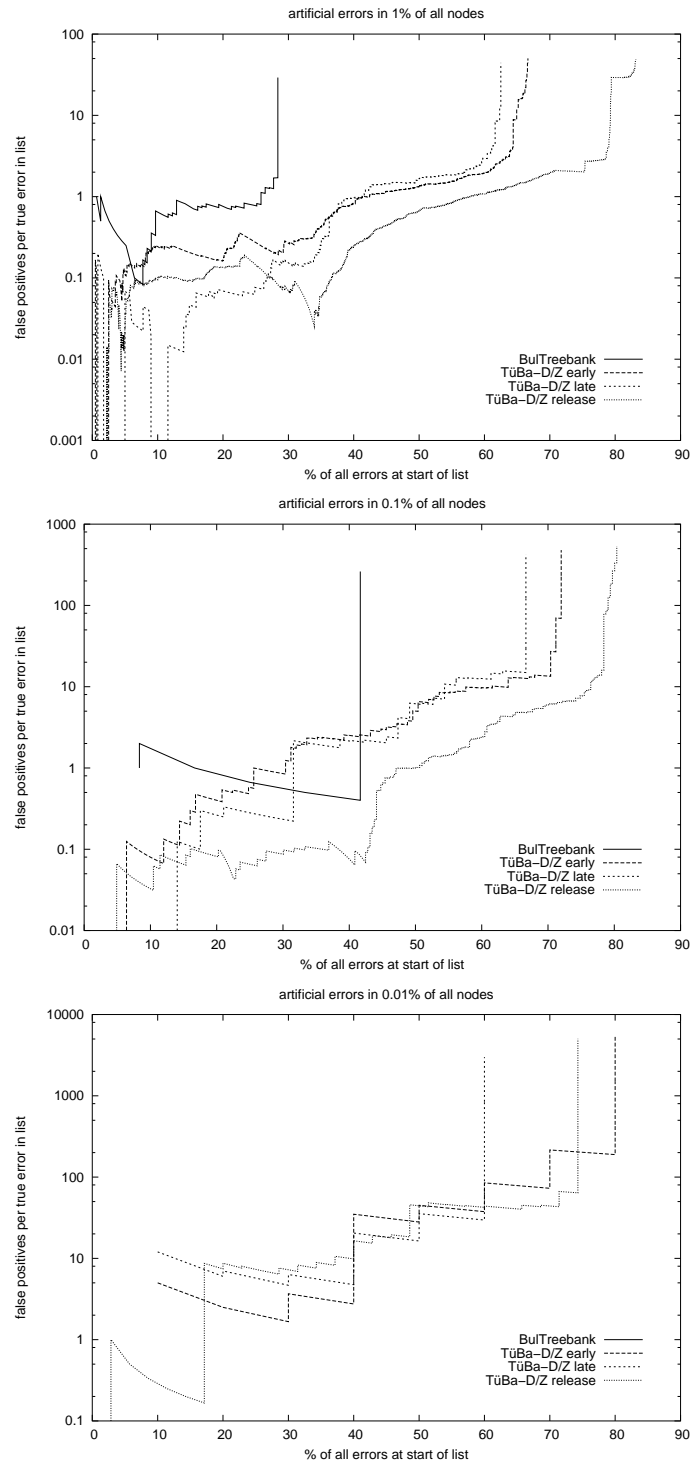


Figure 10.1: False Positives per True Positive

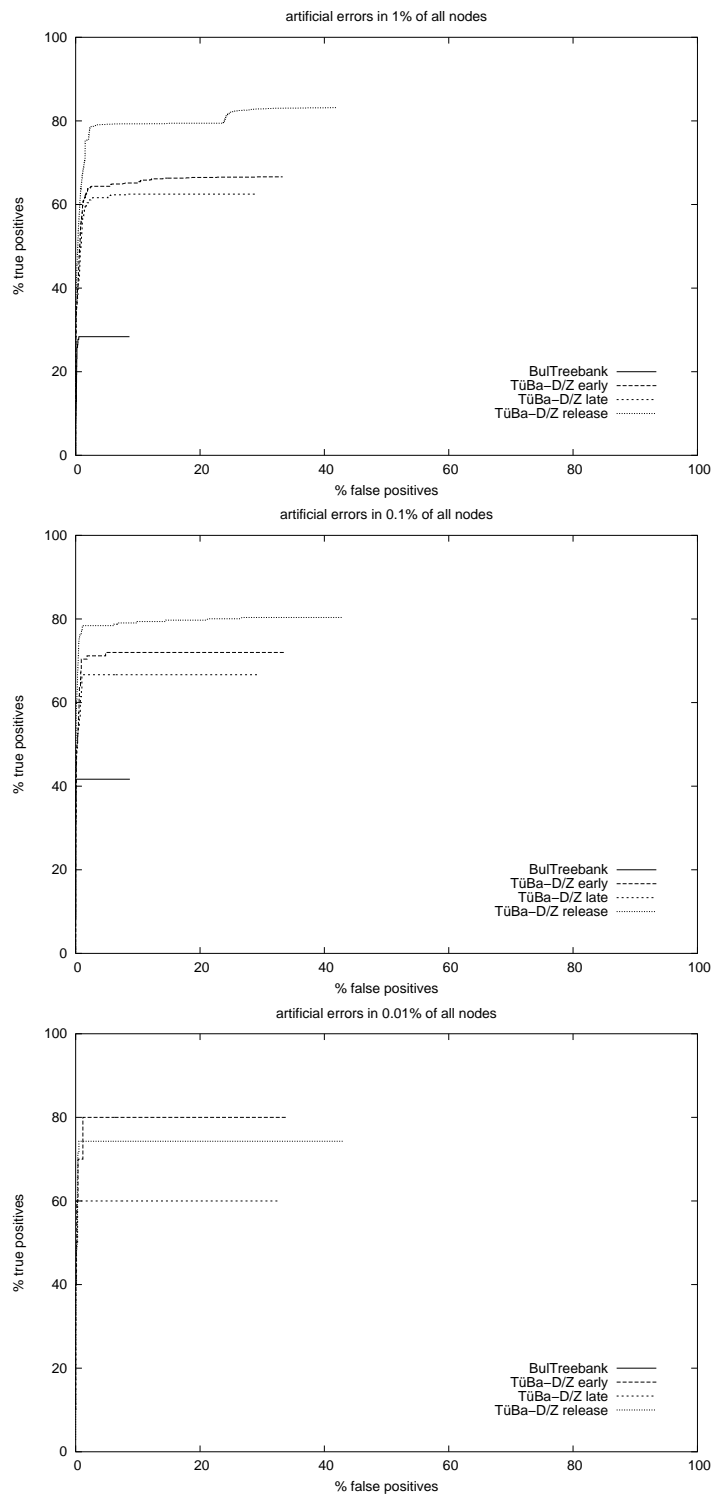


Figure 10.2: ROC Curves for Classification of Errors

performed a similar experiment for TüBa-D/Z, checking the highest ranked candidate errors of the *release* data set that are found by training Timbl on data with 0.01% artificial errors and applying it to the original, clean data. The resulting error detection list shows 12 true errors and 3 unclear cases among the first 20 candidates. Errors included e.g. a finite verb within an infinite verbal phrase, and a missing field node between sentential and phrasal nodes. Rerunning the same experiment with Timbl trained on data with 0.1% artificial errors results in 2 errors and 9 unclear cases among the first 30 candidates, indicating that the training data should resemble the rather clean target data.

It is likely that randomly changing node labels does not resemble well the distribution of naturally occurring errors. We are optimistic, though, that only few kinds of natural errors cannot be detected at all, because figure 10.1 shows that more than 75% of all errors can be found for large and clean datasets.

### 10.3 Discussion and Related Work

Error detection as presented here focuses on errors that distort the probability distribution of context-free productions. While these errors may only be a subset of all errors, we believe that they are very relevant for improving the usefulness of a corpus as a training resource for parsers, because probabilistic parsers usually condition the probability of a node's production fully or partly on the node label. Error detection can thus be seen as a means to clean a corpus from errors particularly harming performance of parsers that rely heavily on node labels, like those based on PCFG models. The abstract units considered in this work are defined as a context-free grammar, i.e. as productions in the context of a parent node. But the method is not restricted to this definition of  $(c, f, p)$  triples. It could equally be defined in terms of e.g. dependency relations, where  $f$  is the syntactic category of some word in a sentence, and  $c$  is the category of the head on which it depends, and  $p$  the set of categories representing its children.

There are several lines of related research. Dickinson and Meurers (2003) use the notion of *variation n-grams* — a sequence of word form tokens with different annotations in different occurrences in the corpus. The variations between n-grams are likely to indicate errors in the corpus. Their method is similar to ours in that potential errors need to be inspected by humans. However, in our case there is a measure which helps us to rank candidate errors. In their method, the context of potential errors is defined by (lexical) word form tokens, whereas we use syntactic categories rather than word

forms. Květoň and Oliva (2002) show how errors can be detected in POS-tagged corpora. Their approach is based on searching for *impossible n-grams* in a corpus. They directly point to the occurrences of errors, but at the same time their method depends on hand-crafted definitions of relevant n-grams. An advantage of this method over ours is that it is in principle able to detect errors that occur systematically in certain contexts; however, it requires more linguistic knowledge.

Each corpus contains two types of linguistic information, which we call *explicit* and *implicit* information. The former is usually given in the documentation of a corpus, and the latter is based on the annotators' intuition encoded in the particular annotation; both can be erroneous. As pointed out before, explicit errors are usually easy to spot via clear rules. Spotting implicit errors requires at least a description of the places where these errors may occur, a description of the context on that the errors depend, and a method for recognising potential errors in a context. Defining errors relative to implicit linguistic information to a great extent requires linguistic intuition and also experiments for verification. The advantage of our method is that it is not limited to certain definitions of errors and contexts. Moreover, the model generated in the ML stage of error detection abstracts away from language-specific details and thus allows training on a larger and better developed treebank of one language and applying the resulting model to a treebank of a different language for which less training data is available. Similar to the other methods for detecting errors, our approach will be most useful in an interactive environment. We have therefore incorporated an interface for the error list into the CLaRK interactive annotation tool that navigates to the candidate error positions, and also allows to apply symbolic pattern matching techniques to find errors.<sup>5</sup>

## 10.4 Conclusion

We have presented a method for detecting errors and inconsistencies in the structural annotation of treebanks. The method is based on the observation that the productions of nonterminals should behave consistently across all contexts in a corpus. We generalise from the output of a statistical test by applying machine-learning to features extracted from its output. The method performs well across different languages and sizes of corpora, and evaluation shows that the method is reliable, independent of corpus size, annotation quality, and target language, so that it seems to be well suited to validate corpora that still undergo annotation at any level of completion.

---

<sup>5</sup>Available at <http://www.bultreebank.org/clark>.



# Chapter 11

## Conclusions

We have presented Treebank Refinement, which is a method that tunes the representation of syntactic analyses in a treebank to the specific needs of probabilistic context-free parsers. We have shown that the choice of representations of syntactic analyses in a treebank determines the performance of PCFG parsers but is rarely chosen to satisfy their specific needs. Their known weakness to model contextual information can be alleviated by choosing representations that introduce more contextual information into node labels. Treebank Refinement defines a function that selects and modifies node labels in the original representation that lack this kind of contextual information. From a more general point of view, the success of this method shows that it is useful to choose more appropriate representations for different tasks, and to consider the design decisions that determine the shape of the original representation of analyses of any treebank.

We have applied Treebank Refinement to a German treebank and found that the changes in the annotation that it suggests are easy to understand. They lead to a considerable improvement in performance of syntactic parsing via PCFGs that compares favourably with an alternative treebank transformation, which at the same time introduces more changes to the treebank. Being easy to follow, the individual changes proposed by Treebank Refinement can be evaluated manually and lead to a more detailed understanding of the annotation in the treebank. The judgements about the individual changes show characteristic patterns for annotation errors and can accordingly be employed to direct annotators to inconsistencies in a treebank.

Treebank Refinement builds on previous work that examines the connections between PCFG parsing performance and the shape of the context-free grammar, first and foremost on the idea of *Grammar Refinement* by Bockhorst and Craven (2001). It adds to related work in several respects. First, it competes with other methods to transform treebank grammars in order to

improve parser performance. Most successful methods carefully analyse the data at hand and propose changes accordingly, either in a framework of strict PCFG parsing that is similar to ours (Schiehlen, 2004), or as a pre-processing step for more powerful parsers (Collins, 1999). Treebank Refinement does not require the same effort. Quite to the contrary, it can be used to automate the search process and spot candidate changes. In contrast to other automatic methods (Belz, 2002b; Johnson, 1998), the motivation for changes proposed by Treebank Refinement is easier to understand, because each modification is described by its area of application (via *focus* and *context*), and also motivated by differences in behaviour (i.e. in *production* frequencies). The other methods often apply more changes and thus lead to larger grammars, or have a smaller search space for information to include into node labels. The second obvious class of competitors to Treebank Refinement are more powerful syntactic parsers. The main difference to them is that these parsers are usually geared towards a specific language or to a specific treebank by extending contextual sensitivity appropriately (Collins, 1999; Dubey and Keller, 2003). Instead, Treebank Refinement does not define a parser but changes the input to a parser. It is thus rather complementing more powerful parsing methods than competing with them, as it specialises node labels if they are used for very different purposes. The complexity of parsing can thus be kept low where the parsers try to cope with simple structural preferences that can equally be expressed by more appropriate node labels. Using Treebank Refinement in conjunction with plain PCFG parsing focuses on these structural preferences and defines a baseline that can be reached by automatic means and that can be utilised efficiently by standard PCFG parsers.

There are several obvious areas for future research. Extending the search space for candidate modifications has proved useful already, but we had to restrict descendant and lexical contexts for considerations of efficiency. A more efficient implementation will allow us to search all ancestor and descendant contexts, and possibly also allow searching for relevant contexts across branching paths. Attribute context has so far only been used to offer parts of the original annotation as optional information to Treebank Refinement. Any other information could be introduced as attributes as well, including underspecified morphological analyses or the depth of embedding. The former will be useful when certain open-class words prefer to appear in distinct positions depending on their ambiguity class, and the latter can distinguish the behaviour of node labels depending on their usage as chunk or complex phrase labels. The application of Treebank Refinement to other treebanks is a further interesting area for future research. Preliminary experiments on the Penn Treebank have shown that Treebank Refinement can be combined successfully with parent annotation and manual optimisations,



and some of the modifications proposed by Treebank Refinement were in fact unintentional omissions. Preliminary experiments on the Italian Syntactic-Semantic Treebank (Montemagni *et al.*, 2003) have also resulted in improved PCFG parsing performance, moreover indicating that the original POS tag set should be more fine-grained for better performance. Treebank Refinement thus seems to be widely applicable for parsing, understanding, and correcting syntactically annotated data.



# Appendix A

## Label Sets used in negra and TüBa-D/Z

### A.1 STTS POS Tags

POS tags according to Schiller & al. (1995), which are common to TüBa-D/Z and negra except for the aberrations mentioned in section 5.2.4. English descriptions follow Telljohann & al. (2003).

Tag	Description
ADJA	attributive adjective
ADJD	adverbial or predicative adjective
ADV	adverb
APPR	preposition; left circumposition
APPRART	preposition plus article
APPO	postposition
APZR	right circumposition
ART	definite or indefinite article
CARD	cardinal number
FM	foreign language material
ITJ	interjection
KOUI	subordinating conjunction with <i>zu</i> plus infinitive
KOUS	subordinating conjunction with clause
KON	coordinating conjunction
KOKOM	particle of comparison, no clause
NN	noun
NE	proper noun

continued on next page

Tag	Description
PDS	substituting demonstrative pronoun
PDAT	attributive demonstrative pronoun
PIS	substituting indefinite pronoun
PIAT	attributive indefinite pronoun without determiner
PIDAT	attributive indefinite pronoun with determiner
PPER	irreflexive personal pronoun
PPOSS	substituting possessive pronoun
PPOSAT	attributive possessive pronoun
PRELS	substituting relative pronoun
PRELAT	attributive relative pronoun
PRF	reflexive personal pronoun
PWS	substituting interrogative pronoun
PWAT	attributive interrogative pronoun
PWAV	adverbial interrogative or relative pronoun
PAV	pronominal adverb
PTKZU	<i>zu</i> plus infinitive
PTKNEG	negative particle
PTKVZ	separated verb particle
PTKANT	answer particle
PTKA	particle with adjective or adverb
TRUNC	first part of truncated word
VVFIN	finite main verb
VVIMP	imperative, main verb
VVINFINF	infinitive, main verb
VVIZU	infinitive plus <i>zu</i> , main verb
VVPP	past participle, main verb
VAFIN	finite auxiliary verb
VAIMP	imperative, auxiliary verb
VAINFINF	infinitive, auxiliary verb
VAPP	past participle, auxiliary verb
VMFIN	finite modal verb
VMINFINF	infinitive, modal verb
VMPP	past participle, modal verb
XY	non-word containing special characters

continued on next page

Tag	Description
\$,	comma
\$.	sentence-final punctuation
\$(	other sentence internal punctuation

Table A.1: STTS POS Tags

## A.2 negra Node and Edge Labels

The tables follow the list and description of labels that comes with the negra corpus (see Albert & al., 2003, for a detailed description of the similar label set of the Tiger treebank).

Label	Description
AA	superlative phrase with <i>am</i>
AP	adjective phrase
AVP	adverbial phrase
CAC	coordinated adposition
CAP	coordinated adjective phrase
CAVP	coordinated adverbial phrase
CCP	coordinated complementiser
CH	chunk
CNP	coordinated noun phrase
CO	coordination
CPP	coordinated adpositional phrase
CS	coordinated sentence
CVP	coordinated verb phrase (non-finite)
CVZ	coordinated infinitive with <i>zu</i>
DL	discourse level constituent
ISU	idiosyncratic unit
MPN	multi-word proper noun
MTA	multi-token adjective
NM	multi-token number
NP	noun phrase
PP	adpositional phrase
QL	quasi-language

continued on next page

Label	Description
S	sentence
VP	verb phrase (non-finite)
VZ	infinitive with <i>zu</i>

Table A.2: negra Node Labels

Label	Description
AC	adpositional case marker
ADC	adjective component
AMS	measure argument of adjective
APP	apposition
AVC	adverbial phrase component
CC	comparative complement
CD	coordinating conjunction
CJ	conjunct
CM	comparative conjunction
CP	complementiser
DA	dative
DH	discourse-level head
DM	discourse marker
GL	prenominal genitive
GR	postnominal genitive
HD	head
JU	junctor
MC	comitative
MI	instrumental
ML	locative
MNR	postnominal modifier
MO	modifier
MR	rhetorical modifier
MW	way (directional modifier)
NG	negation
NK	noun kernel modifier
NMC	numerical component
OA	accusative object
OA2	second accusative object

continued on next page

Label	Description
OC	clausal object
OG	genitive object
PD	predicate
PG	pseudo-genitive
PH	placeholder
PM	morphological particle
PNC	proper noun component
RC	relative clause
RE	repeated element
RS	reported speech
SB	subject
SBP	passivised subject (PP)
SP	subject or predicate
SVP	separable verb prefix
UC	(idiosyncratic) unit component
VO	vocative

Table A.3: negra Edge Labels

### A.3 TüBa-D/Z Node and Edge Labels

The tables follow Telljohann & al. (2003).

Label	Description
NX	noun phrase
PX	prepositional phrase
ADVX	adverbial phrase
ADJX	adjectival phrase
VXFIN	finite verb phrase
VXINF	non-finite verb phrase
FX	foreign language phrase
DP	determiner phrase
LV	resumptive construction (Linkversetzung)
VF	initial field (Vorfeld)
LK	left sentence bracket (Linke Satzklammer)
MF	middle field (Mittelfeld)

continued on next page

Label	Description
VC	verb complex (Verbkomplex)
NF	final field (Nachfeld)
C	complementiser field (C-Feld)
KOORD	field for coordinating particles
PARORD	field for non-coordinating particles
FKOORD	coordination consisting of conjuncts of fields
MFE	middle field between VCE and VC
VCE	VC with split finite <i>Ersatzinfinitiv</i> verb
FKONJ	conjunct consisting of more than one field
SIMPX	simplex clause
R-SIMPX	relative clause
P-SIMPX	paratactic construction of simplex clauses
DM	discourse marker
EN-ADD	named entity

Table A.4: TüBa-D/Z Node Labels

Label	Description
HD	head
-	non-head
KONJ	conjunct
ON	nominative object (=subject)
OD	dative object
OA	accusative object
OG	genitive object
OS	sentential object
OPP	prepositional object
OADVP	adverbial object
OADJP	adjectival object
PRED	predicate
OV	verbal object
FOPP	optional prepositional object
VPT	separable verb prefix
APP	apposition
MOD	ambiguous modifier
ON-MOD	modifier of the nominative object

continued on next page



Label	Description
OD-MOD	modifier of the dative object
OA-MOD	modifier of the accusative object
OG-MOD	modifier of the genitive object
OPP-MOD	modifier of the prepositional object
OS-MOD	modifier of the sentential object
PRED-MOD	modifier of the predicate
FOPP-MOD	modifier of the optional prepositional object
OADJP-MOD	modifier of the adjectival object
V-MOD	modifier of the verb
MOD-MOD	modifier of another modifier
ONK	second split-up nominative object conjunct
ODK	second split-up dative object conjunct
OAK	second split-up accusative object conjunct
FOPPK	second split-up optional prepositional object conjunct
OADVPK	second split-up adverbial object conjunct
PREDK	second split-up conjunct of the predicate
MODK	ambiguous second split-up modifier
V-MODK	second split-up conjunct of the verb modifier

Table A.5: TüBa-D/Z Edge Labels



# Appendix B

## PP-Attachment in negra and TüBa-D/Z with Edges

We show how PCFG estimates of high and low PP attachment change when all edge labels are appended to the node labels in the example given in section 5.1.

High and low attachment according to the annotation scheme of TüBa-D/Z is shown in figure B.1.

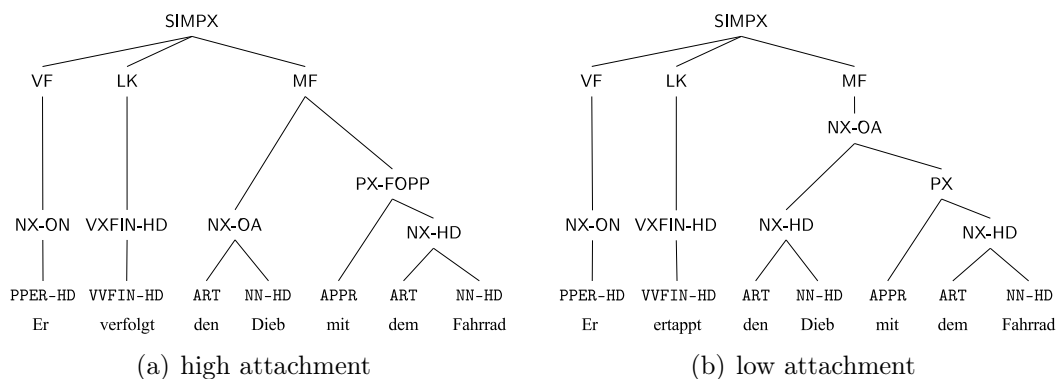


Figure B.1: German PP Attachment in TüBa-D/Z with all Edge Labels

Maximum-likelihood estimates of the rule probabilities are accordingly:

$$\begin{array}{ll}
 P( VF \rightarrow NX-ON ) = 1 & P( SIMPX \rightarrow VF LK MF ) = 1 \\
 P( NX-ON \rightarrow PPER ) = 1 & P( VXFIN-HD \rightarrow VVFIN-HD ) = 1 \\
 P( PPER-HD \rightarrow Er ) = 1 & P( PX-FOPP \rightarrow APPR NX-HD ) = 1 \\
 P( LK \rightarrow VXFIN-HD ) = 1 & P( PX \rightarrow APPR NX-HD ) = 1
 \end{array}$$

$$\begin{aligned}
P(\text{NX-HD} \rightarrow \text{ART NN-HD}) &= 1 \\
P(\text{ART} \rightarrow \text{den}) &= 1/2 \\
P(\text{NN-HD} \rightarrow \text{Dieb}) &= 1/2 \\
P(\text{APPR} \rightarrow \text{mit}) &= 1 \\
P(\text{ART} \rightarrow \text{dem}) &= 1/2 \\
P(\text{NN-HD} \rightarrow \text{Fahrrad}) &= 1/2 \\
P(\text{VFIN-HD} \rightarrow \text{verfolgt}) &= \frac{h}{h+l} \\
P(\text{MF} \rightarrow \text{NX-OA PX-FOPP}) &= \frac{h}{h+l} \\
P(\text{NX-OA} \rightarrow \text{ART NN-HD}) &= \frac{h}{h+l} \\
P(\text{VFIN-HD} \rightarrow \text{ertappt}) &= \frac{l}{h+l} \\
P(\text{MF} \rightarrow \text{NX-OA}) &= \frac{l}{h+l} \\
P(\text{NX-OA} \rightarrow \text{NX-HD PX}) &= \frac{l}{h+l}
\end{aligned}$$

so that the high attachment parse of the sentence in figure B.1(a) has the probability

$$\begin{aligned}
P(\phi^a | w^a) &= P(\text{VFIN-HD} \rightarrow \text{verfolgt}) \\
&P(\text{MF} \rightarrow \text{NX-OA PX-FOPP}) P(\text{NX-OA} \rightarrow \text{ART NN-HD}) \\
&P(\text{ART} \rightarrow \text{den}) P(\text{NN-HD} \rightarrow \text{Dieb}) \\
&P(\text{ART} \rightarrow \text{dem}) P(\text{NN-HD} \rightarrow \text{Fahrrad})
\end{aligned}$$

and the low attachment parse of the same sentence has the probability

$$\begin{aligned}
P(\phi^b | w^a) &= P(\text{VFIN-HD} \rightarrow \text{verfolgt}) \\
&P(\text{MF} \rightarrow \text{NX-OA}) P(\text{NX-OA} \rightarrow \text{NX-HD PX}) \\
&P(\text{ART} \rightarrow \text{den}) P(\text{NN-HD} \rightarrow \text{Dieb}) \\
&P(\text{ART} \rightarrow \text{dem}) P(\text{NN-HD} \rightarrow \text{Fahrrad})
\end{aligned}$$

so that the first parse is selected when

$$\begin{aligned}
 & P(\phi^a | w^a) > P(\phi^b | w^a) \\
 \Leftrightarrow & P( MF \rightarrow NX-OA \ PX-FOPP ) \\
 & P( NX-OA \rightarrow ART \ NN-HD ) > P( MF \rightarrow NX-OA ) \\
 & P( NX-OA \rightarrow NX-HD \ PX ) \\
 \Leftrightarrow & \frac{h}{h+l} \frac{h}{h+l} > \frac{l}{h+l} \frac{l}{h+l} \\
 \Leftrightarrow & h > l \tag{B.1}
 \end{aligned}$$

Accordingly, both attachments can be represented according to the negra annotation scheme as shown in figure B.2.

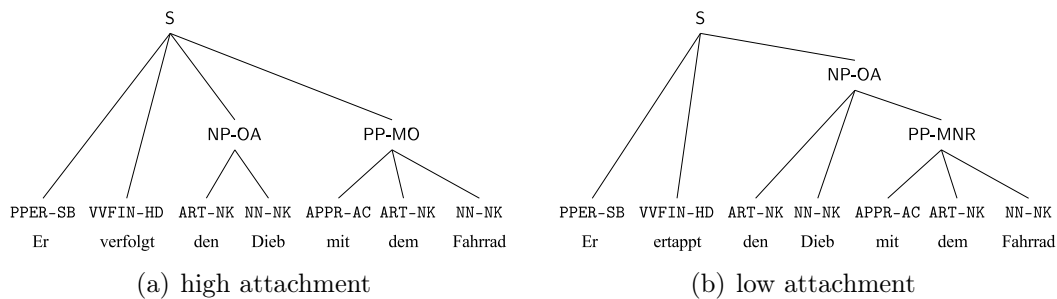


Figure B.2: German PP attachment in negra with all Edge Labels

Maximum-likelihood estimates of the rule probabilities are accordingly:

$$\begin{aligned}
P(\text{PPER-SB} \rightarrow \textit{Er}) &= 1 \\
P(\text{ART-NK} \rightarrow \textit{den}) &= 1/2 \\
P(\text{NN-NK} \rightarrow \textit{Dieb}) &= 1/2 \\
P(\text{APPR-AC} \rightarrow \textit{mit}) &= 1 \\
P(\text{ART-NK} \rightarrow \textit{dem}) &= 1/2 \\
P(\text{NN-NK} \rightarrow \textit{Fahrrad}) &= 1/2 \\
P(\text{PP-MO} \rightarrow \text{APPR-AC ART-NK NN-NK}) &= 1 \\
P(\text{PP-MNR} \rightarrow \text{APPR-AC ART-NK NN-NK}) &= 1 \\
P(\text{S} \rightarrow \text{PPER-SB VVFIN-HD NP-OA PP-MO}) &= \frac{h}{h+l} \\
P(\text{VVFIN-HD} \rightarrow \textit{verfolgt}) &= \frac{h}{h+l} \\
P(\text{NP-OA} \rightarrow \text{ART-NK NN-NK}) &= \frac{h}{h+l} \\
P(\text{S} \rightarrow \text{PPER-SB VVFIN-HD NP-OA}) &= \frac{l}{h+l} \\
P(\text{VVFIN-HD} \rightarrow \textit{ertappt}) &= \frac{l}{h+l} \\
P(\text{NP-OA} \rightarrow \text{ART-NK NN-NK PP-MNR}) &= \frac{l}{h+l}
\end{aligned}$$

so that the high attachment parse of the sentence in figure B.2(a) has the probability

$$\begin{aligned}
P(\phi^a | w^a) &= P(\text{VVFIN-HD} \rightarrow \textit{verfolgt}) \\
&P(\text{S} \rightarrow \text{PPER-SB VVFIN-HD NP-OA PP-MO}) \\
&P(\text{NP-OA} \rightarrow \text{ART-NK NN-NK}) \\
&P(\text{ART-NK} \rightarrow \textit{den}) P(\text{NN-NK} \rightarrow \textit{Dieb}) \\
&P(\text{ART-NK} \rightarrow \textit{dem}) P(\text{NN-NK} \rightarrow \textit{Fahrrad})
\end{aligned}$$

whereas the low attachment parse of the same sentence has the probability

$$\begin{aligned}
 P(\phi^b|w^a) &= P(\text{VVFİN-HD} \rightarrow \textit{verfolgt}) \\
 &P(\text{S} \rightarrow \text{PPER-SB VVFİN-HD NP-OA}) \\
 &P(\text{NP-OA} \rightarrow \text{ART-NK NN-NK PP-MNR}) \\
 &P(\text{ART-NK} \rightarrow \textit{den}) P(\text{NN-NK} \rightarrow \textit{Dieb}) \\
 &P(\text{ART-NK} \rightarrow \textit{dem}) P(\text{NN-NK} \rightarrow \textit{Fahrrad})
 \end{aligned}$$

so that the high attachment parse is selected when the following inequation holds:

$$\begin{aligned}
 &P(\phi^a|w^a) > P(\phi^b|w^a) \\
 \Leftrightarrow &P(\text{S} \rightarrow \text{PPER-SB VVFİN-HD NP-OA PP-MO}) \\
 &P(\text{NP-OA} \rightarrow \text{ART-NK NN-NK}) \\
 &> \\
 &P(\text{S} \rightarrow \text{PPER-SB VVFİN-HD NP-OA}) \\
 &P(\text{NP-OA} \rightarrow \text{ART-NK NN-NK PP-MNR}) \\
 \Leftrightarrow &\frac{h}{h+l} \frac{h}{h+l} > \frac{l}{h+l} \frac{l}{l+h} \\
 \Leftrightarrow &h > l
 \end{aligned}$$

We see that both annotation schemes provide treebank grammars for which the ratio between the likelihoods of both parses equals the ratio between the occurrence frequencies in the treebank when we use edge information as additional decoration on all node labels.





# Bibliography

- Abeillé, Anne, ed. (2003). *Treebanks: Building and using Parsed corpora*, vol. 20 of *Text, Speech and Language Technology*. Dordrecht: Kluwer.
- Albert, Stefanie, Jan Anderssen, Regine Bader, Stephanie Becker, Tobias Bracht, Sabine Brants, Thorsten Brants, Vera Demberg, Stefanie Dipper, Peter Eisenberg, Silvia Hansen, Hagen Hirschmann, Juliane Janitzek, Carolin Kirstein, Robert Langner, Lukas Michelbacher, Oliver Plaehn, Cordula Preis, Marcus Pußel, Marco Rower, Bettina Schrader, Anne Schwartz, George Smith, and Hans Uszkoreit (2003). TIGER Annotationschema. Universität des Saarlandes. FR 8.7 Computerlinguistik. Universität Stuttgart. Institut für Maschinelle Sprachverarbeitung. Universität Potsdam. Institut für Germanistik.
- Becker, Markus and Anette Frank (2002). A Stochastic Topological Parser for German. In: *Proceedings of the 19th International Conference on Computational Linguistics*, vol. 1, pp. 71–77. Taipei, Taiwan.
- Beil, Franz, Glenn Carroll, Detlef Prescher, Stefan Riezler, and Mats Rooth (1999). Inside-Outside Estimation of a Lexicalized PCFG for German. In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, Maryland, USA.
- Beil, Franz, Detlef Prescher, Helmut Schmid, and Sabine Schulte im Walde (2002). Evaluation of the Gramotron Parser for German. In: *Proceedings of the Workshop ‘Beyond PARSEVAL — Towards improved evaluation measures for parsing systems’ at the 3rd International Conference on Language Resources and Evaluation*, pp. 52–59. Las Palmas, Spain.
- Belz, Anja (2002a). Learning grammars for different parsing tasks by partition search. In: *Proceedings of the 19th International Conference on Computational Linguistics*. Taipei, Taiwan.
- Belz, Anja (2002b). PCFG Learning by Nonterminal Partition Search. In:

- Proceedings of the 6th International Colloquium on Grammatical Inference*, pp. 14–27. Amsterdam, the Netherlands.
- Bies, Ann, Mark Ferguson, Karen Katz, and Robert MacIntyre (1995). Bracketing Guidelines for Treebank II Style Penn Treebank Project. Technical report, University of Pennsylvania.
- Bikel, Daniel M. (2004). A Distributional Analysis of a Lexicalized Statistical Parsing Model. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pp. 182–189. Barcelona, Spain.
- Bird, Steven and Mark Liberman (2001). A formal framework for linguistic annotation. *Speech Communication*, 33(1,2):23–60.
- Black, Ezra, Steven Abney, Dan Flickinger, C. Gdaniek, Ralph Grishman, P. Harrison, Donald Hindle, Robert Ingria, Frederik Jelinek, Judith Klavans, Mark Liberman, Mitchell Marcus, Salim Roukos, Beatrice Santorini, and T. Strzalkowski (1991). A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In: *Proceedings of the DARPA Workshop on Speech and Natural Language*. Pacific Grove, California.
- Blaheta, Don (2002). Handling noisy training and testing data. In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pp. 111–116. Philadelphia, PA, USA.
- Bockhorst, Joseph and Mark Craven (2001). Refining the Structure of a Stochastic Context-Free Grammar. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. Seattle, Washington, USA.
- Bod, Rens (1995). *Enriching Linguistics with Statistics: Performance Models of Natural Language*. Ph.D. thesis, University of Amsterdam.
- Bod, Rens (2000). Parsing with the Shortest Derivation. In: *Proceedings of the 18th International Conference on Computational Linguistics*, vol. 1. Saarbrücken, Germany, Nancy, France, and Luxembourg, Luxembourg.
- Böhmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Hladká (2003). The Prague Dependency Treebank: A Three-Level Annotation Scenario. In: Abeillé (2003), ch. 7.
- Bosco, Cristina and Vincenzo Lombardo (2004). Dependency and relational structure in treebank annotation. In: *Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar*, pp. 1–8. Geneva, Switzerland.

- Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith (2002). The TIGER Treebank. In: *Proceedings of the First Workshop on Treebanks and Linguistic Theories*. Sozopol, Bulgaria.
- Brants, Sabine and Silvia Hansen (2002). Developments in the TIGER Annotation Scheme and their Realization in the Corpus. In: *Proceedings of the Third International Conference on Language Resources and Evaluation*, pp. 1643–1649. Las Palmas, Spain.
- Brants, Thorsten (1997). The NeGra Export Format for Annotated Corpora. Technical report, Universität des Saarlandes. Computerlinguistik.
- Brants, Thorsten (1999). TnT user’s manual. Technical report, Universität des Saarlandes.
- Brants, Thorsten (2000). TnT – A Statistical Part-of-Speech Tagger. In: *Proceedings of the 6th Applied Natural Language Processing Conference*. Seattle, WA, USA.
- Carroll, Glenn and Mats Rooth (1998). Valence induction with a head-lexicalized PCFG. In: *Proceedings of the Third Conference on Empirical Methods in Natural Language Processing*. Granada, Spain.
- Carroll, John, Ted Briscoe, and Antonio Sanfilippo (1998). Parser evaluation: a survey and a new proposal. In: *Proceedings of the First International Conference on Language Resources and Evaluation*, pp. 447–454. Granada, Spain.
- Charniak, Eugene (1996). Tree-bank Grammars. Technical report CS-96-02, Brown University, Department of Computer Science.
- Charniak, Eugene (1997). Statistical Parsing with a Context-free Grammar and Word Statistics. In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. Menlo Park.
- Charniak, Eugene (1999). A Maximum-Entropy-Inspired Parser. Technical report CS-99-12, Brown University.
- Charniak, Eugene (2001). Immediate-Head Parsing for Language Models. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France.
- Charniak, Eugene and Glenn Carroll (1992). Two Experiments on Learning Probabilistic Dependency Grammars from Corpora. In: *Workshop Notes for Statistically-Based NLP Techniques*, pp. 1–13.

- Chi, Zhiyi (1999). Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.
- Chomsky, Noam (1956). Three Models for the Description of Language. *IRE Transactions of Information Theory*, IT-2(3):113–124.
- Collins, Michael (1997). Three Generative, Lexicalised Models for Statistical Parsing. In: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*. Madrid, Spain.
- Collins, Michael (1999). *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.
- Collins, Michael (2000). Discriminative Reranking for Natural Language Parsing. In: *Proceedings ICML-2000*. Stanford, CA.
- Collins, Michael, Jan Hajic, Lance Ramshaw, and Christoph Tillmann (1999). A Statistical Parser for Czech. In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, Maryland, USA.
- Corazza, Anna, Alberto Lavelli, Giorgio Satta, and Roberto Zanolli (2004). Analyzing An Italian Treebank with State-of-the-art Statistical Parsers. In: *Proceedings of the Third Workshop on Treebanks and Linguistic Theories*, pp. 39–50. Tübingen, Germany.
- Cover, Thomas M. and Joy A. Thomas (1991). *Elements of Information Theory*. New York et al.: John Wiley & Sons.
- Daelemans, Walter, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch (2003). TiMBL: Tilburg Memory Based Learner, version 5.0. Reference Guide. Technical report ILK 03-10, Tilburg University.
- Dickinson, Markus and Walt Detmar Meurers (2003). Detecting Inconsistencies in Treebanks. In: *Proceedings of the Second Workshop on Treebanks and Linguistic Theories*. Växjö, Sweden: Växjö University Press.
- Dipper, Stefanie, Michael Götze, and Stavros Skopeteas (2004). Towards User-Adaptive Annotation Guidelines. In: *Proceedings of the COLING 2004 5th International Workshop on Linguistically Interpreted Corpora*, pp. 23–30. Geneva, Switzerland.
- Doran, Christine (2000). Punctuation in a Lexicalized Grammar. In: *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms*. Paris, France.

- Drach, Erich (1937). *Grundgedanken der Deutschen Satzlehre*. Frankfurt/Main: Diesterweg.
- Dubey, Amit and Frank Keller (2003). Probabilistic Parsing for German using Sister-Head Dependencies. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan.
- Earley, Jay (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.
- Eisenberg, Peter, Jörg Peters, Peter Gallmann, Cathrine Fabricius-Hansen, Damaris Nübling, Irmhild Barz, Thomas A. Fritz, and Reinhard Fiehler, eds. (2005). *Duden. Die Grammatik*, vol. 4. Mannheim et al.: Dudenverlag.
- Engel, Ulrich (1970). Regeln zur Wortstellung. In: *Forschungsberichte des Instituts für deutsche Sprache*, (Ed.) Ulrich Engel, vol. 5, pp. 7–148. o.O.
- Erdmann, Oskar (1886). *Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung*. Erste Abteilung. Stuttgart: Cotta.
- Fawcett, Tom (2003). ROC Graphs: Notes and Practical Considerations for Researchers. Technical report HPL-2003-4, HP Laboratories, Palo Alto, CA, USA.
- Frank, Anette, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schäfer (2003). Integrated Shallow and Deep Parsing: TopP meets HPSG. In: *Proceedings of ACL-2003*. Sapporo, Japan.
- Gaizauskas, Robert (1995). Investigations into the Grammar Underlying the Penn Treebank II. Research Memorandum CS-95-25, Department of Computer Science, Univeristy of Sheffield, Sheffield, United Kingdom.
- Gildea, Daniel (2001). Corpus Variation and Parser Performance. In: *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pp. 167–202.
- Goodman, Joshua (1996a). Efficient Algorithms for Parsing the DOP Model. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain.
- Goodman, Joshua (1996b). Parsing Algorithms and Metrics. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pp. 177–183. San Francisco.

- Goodman, Joshua (2003). Efficient Parsing of DOP with PCFG-reductions. In: *Data-Oriented Parsing*, (Eds.) Rens Bod, Remko Scha, and Khalil Sima'an, Center for the Study of Language and Information - Studies in Computational Linguistics. Chicago University Press.
- Grishman, Ralph, Catherine Macleod, and John Sterling (1992). Evaluating parsing strategies using standardized parse files. In: *Proceedings of the Third Conference on Applied Natural Language Processing*. Trento, Italy.
- Grune, Dick and Criel J.H. Jacobs (1990). *Parsing Techniques - A Practical Guide*. Chichester, England: Ellis Horwood.
- Harrison, Philip, Steven Abney, Ezra Black, Dan Flickenger, Claudia Gdaniec, Ralph Grishman, Donald Hindle, Robert Ingria, Mitch Marcus, Beatrice Santorini, and Tomek Strzalkowski (1991). Evaluating syntax performance of parser/grammars of English. In: *Proceedings of the Workshop On Evaluating Natural Language Processing Systems*.
- Herling, S. H. A. (1821). Über die Topik der deutschen Sprache. In: *Abhandlungen des frankfurtischen Gelehrtenvereins für deutsche Sprache*, Drittes Stück, pp. 296–362. Frankfurt/Main.
- Hindle, Donald and Mats Rooth (1993). Structural Ambiguity and Lexical Relations. *Computational Linguistics*, 19(1):103–120.
- Hinrichs, Erhard W. and Julia S. Trushkina (2003). N-gram and PCFG Models for Morpho-Syntactic Tagging of German. In: *Proceedings of the Second Workshop on Treebanks and Linguistic Theories*. Växjö, Sweden: Växjö University Press.
- Höhle, Tilman N. (1985). Der Begriff ‚Mittelfeld‘. Anmerkungen über die Theorie der topologischen Felder. In: *Kontroversen, alte und neue. Akten des 7. Internationalen Germanistenkongresses*, (Ed.) A. Schöne, pp. 329–340. Göttingen.
- Hopcroft, John E. and Jeffrey D. Ullman (1996). *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Bonn et al.: Addison-Wesley, 3rd ed.
- Johnson, Mark (1998). PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24(4):613–632.

- Johnson, Mark (2003). Learning and parsing stochastic unification-based grammars. In: *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop (COLT/Kernel'03)*, (Eds.) Bernhard Schölkopf and Manfred K. Warmuth, vol. 2777 of *Lecture Notes in Computer Science*. Springer.
- Jones, Bernard E. M. (1994). Exploring the Role of Punctuation in Parsing Natural Text. In: *Proceedings of the 15th International Conference on Computational Linguistics*, vol. 1. Kyoto, Japan.
- Joshi, Aravind K. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In: *Natural Language Parsing. Psychological, Computational, and Theoretical Perspectives*, (Eds.) David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, *Studies in Natural Language Processing*, ch. 6, pp. 206–250. New York: Cambridge University Press.
- Joshi, Aravind K. and Leon S. Levy (1982). Phrase Structure Trees Bear More Fruit than You Would Have Thought. *American Journal of Computational Linguistics*, 8(1).
- Karlsson, Fred (2004). Constraints on Clausal Embedding Complexity. In: *Proceedings of the Third Workshop on Treebanks and Linguistic Theories*. Tübingen, Germany. Invited Talk.
- Klein, Dan and Christopher D. Manning (2001a). An  $O(n^3)$  Agenda-Based Chart Parser for Arbitrary Probabilistic Context-Free Grammars. Technical report, Stanford University.
- Klein, Dan and Christopher D. Manning (2001b). Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pp. 330–337. Toulouse, France.
- Klein, Dan and Christopher D. Manning (2003a). A\* Parsing: Fast Exact Viterbi Parse Selection. In: *Proceedings of the Human Language Technology Conference and the 4th Meeting of the North American Association for Computational Linguistics*. Edmonton, Canada.
- Klein, Dan and Christopher D. Manning (2003b). Accurate Unlexicalized Parsing. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pp. 423–430. Sapporo, Japan.

- Krotov, Alexander, Robert Gaizauskas, and Yorick Wilks (1994). Acquiring a Stochastic Context-Free Grammar from the Penn Treebank. In: *Proceedings of Third Conference on the Cognitive Science of Natural Language Processing*, pp. 79–86. Dublin.
- Květoň, Pavel and Karel Oliva (2002). (Semi-)Automatic Detection of Errors in PoS-Tagged Corpora. In: *Proceedings of the 19th International Conference on Computational Linguistics*. Taipei, Taiwan.
- Kübler, Sandra (2002). *Memory-Based Parsing of a German Corpus*. Ph.D. thesis, Universität Tübingen, Tübingen, Germany.
- Kübler, Sandra and Heike Telljohann (2002). Towards a Dependency-Based Evaluation for Partial Parsing. In: *Proceedings of the Workshop ‘Beyond PARSEVAL — Towards improved evaluation measures for parsing systems’ at the 3rd International Conference on Language Resources and Evaluation*. Las Palmas, Spain.
- Lee, Lillian (1999). Measures of Distributional Similarity. In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 25–32. College Park, Maryland, USA.
- Lee, Lillian (2001). On the Effectiveness of the Skew Divergence for Statistical Language Analysis. In: *Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics*, pp. 65–72. Key West, Florida.
- Lemnitzer, Lothar (2003). Ist das nicht doch alles das Gleiche? Regeln und Distanzmaße zur Berücksichtigung orthographischer Idiosynkrasien bei der Abbildung von Textsegmenten auf lexikalische Einheiten. In: *Sprache zwischen Theorie und Technologie. Festschrift für Wolf Paprotté zum 60. Geburtstag*, (Eds.) Lea Cyrus, Hendrik Feddes, Frank Schumacher, and Petra Steiner, pp. 135–148. Wiesbaden: DUV.
- Lin, Dekang (1995). A Dependency-based Method for Evaluating Broad-Coverage Parsers. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1420–1427. Montréal, Québec, Canada.
- Malouf, Robert and Gertjan van Noord (2004). Wide Coverage Parsing with Stochastic Attribute Value Grammars. In: *Proceedings of Workshop “Beyond Shallow Analyses – Formalisms and statistical modeling for deep analyses” at the First International Joint Conference on Natural Language Processing*. Sanya City, Hainan Island, China.



- Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Mass. et al.: MIT Press.
- Marcus, M., S. Santorini, and M. Marcinkiewicz (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.
- Montemagni, Simonetta, Francesco Barsotti, Marco Battista, Nicoletta Calzolari, Ornella Corazzari, Alessandro Lenci, Antonio Zampolli, Francesca Fanciulli, Maria Massetani, Remo Raffaelli, Roberto Basili, Maria Teresa Pazienza, Dario Saracino, Fabio Zanzotto, Nadia Mana, Fabio Pianesi, and Rodolfo Del Monte (2003). Building The Italian Syntactic-Semantic Treebank. In: Abeillé (2003), pp. 189–210.
- Müller, Frank Henrik (to appear). *A Finite State Approach to Shallow Parsing and Grammatical Functions Annotation of German*. Ph.D. thesis, Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany.
- Müller, Frank Henrik and Tylman Ule (2002). Annotating topological fields and chunks – and revising POS tags at the same time. In: *Proceedings of the 19th International Conference on Computational Linguistics*. Taipei, Taiwan.
- Nivre, Joakim (2003). Theory-Supporting Treebanks. In: *Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, pp. 117–128. Växjö, Sweden: Växjö University Press.
- Osenova, Petya and Kiril Simov (2004). BulTreeBank Stylebook. Technical report BTB-TR05, Linguistic Modelling Laboratory, Institute for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria.
- Plaehn, Oliver (1998). Annotate Bedienungsanleitung. Technical report, Universität des Saarlandes. Computerlinguistik, Saarbrücken.
- Plaehn, Oliver (1999). *Probabilistic Parsing with Discontinuous Phrase Structure Grammar*. Master’s thesis, Universität des Saarlandes, Saarbrücken, Germany.
- Prescher, Detlef (2002). *EM-basierte maschinelle Lernverfahren für natürliche Sprachen*. Ph.D. thesis, IMS, Universität Stuttgart, Stuttgart.
- Ratnaparkhi, Adwait (1999). Learning to Parse Natural Language with Maximum Entropy Models. *Machine Learning*, 34:151–175.

- van Rijsbergen, C.J. (1979). *Information Retrieval*. London: Butterworths, 2nd ed.
- Rinne, Horst (1997). *Taschenbuch der Statistik*. Verlag Harri Deutsch, 2nd ed.
- Sampson, Geoffrey (2000). A Proposal for Improving the Measurement of Parse Accuracy. *International Journal of Corpus Linguistics*, 5(1):53–68.
- Sampson, Geoffrey (2001). *Empirical Linguistics*. London and New York: Continuum International.
- Sampson, Geoffrey and Anna Babarczy (2003). A test of the leaf-ancestor metric for parse accuracy. *Natural Language Engineering*, 9(4).
- Schiehlen, Michael (2004). Annotation Strategies for Probabilistic Parsing in German. In: *Proceedings of the 20th International Conference on Computational Linguistics*, pp. 390–396. Geneva, Switzerland.
- Schiller, Anne, Simone Teufel, Christine Stöckert, and Christine Thielen (1999). Guidelines für das Taggen deutscher Textcorpora mit STTS (Kleines und großes Tagset). Technical report, IMS and SfS, Stuttgart and Tübingen, Germany.
- Schiller, Anne, Simone Teufel, and Christine Thielen (1995). Guidelines für das Taggen deutscher Textcorpora mit STTS. Technical report, IMS and SfS, Stuttgart and Tübingen, Germany. Draft.
- Schmid, Helmut (2000). Lopar: Design and Implementation. Arbeitspapiere des Sonderforschungsbereichs 340 149, IMS, Universität Stuttgart, Stuttgart.
- Schmid, Helmut (2004). Efficient Parsing of Highly Ambiguous Context-Free Grammars with Bit Vectors. In: *Proceedings of the 20th International Conference on Computational Linguistics*, pp. 162–168. Geneva, Switzerland.
- Sima'an, Khalil (1996). Computational Complexity of Probabilistic Disambiguation by means of Tree-Grammars. In: *Proceedings of the 16th International Conference on Computational Linguistics*, vol. 2, pp. 1175–1180. Copenhagen, Denmark.
- Simov, Kiril, Gergana Popova, and Petya Osenova (2001). HPSG-based syntactic treebank of Bulgarian (BulTreeBank). In: *A Rainbow of Corpora: Corpus Linguistics and the Languages of the World*, (Eds.) Andrew Wilson, Paul Rayson, and Tony McEnery, pp. 135–142. Munich: Lincom-Europa.

- Skut, Wojciech, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit (1998). A Linguistically Interpreted Corpus of German Newspaper Texts. In: *ESS-LLI Workshop on Recent Advances in Corpus Annotation*. Saarbrücken, Germany.
- Smith, George (2003). A Brief Introduction to the TIGER Treebank. Version 1. Technical report, Universität Potsdam, Potsdam, Germany.
- Stegmann, Rosmary, Heike Telljohann, and Erhard W. Hinrichs (2000). Stylebook for the German Treebank in VERBMOBIL. Verbmobil-Report 239, Eberhard-Karls-Universität Tübingen, Tübingen.
- Taskar, Ben, Dan Klein, Michael Collins, Daphne Koller, and Christopher D. Manning (2004). Max-Margin Parsing. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pp. 1–8. Barcelona, Spain.
- Telljohann, Heike, Erhard W. Hinrichs, and Sandra Kübler (2003). Stylebook for the German Treebank of Written German (TüBa-D/Z). Technical report, Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany.
- Telljohann, Heike, Erhard W. Hinrichs, and Sandra Kübler (2004). The TüBa-D/Z Treebank – Annotating German with a Context-Free Backbone. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Lisbon, Portugal.
- Tjong Kim Sang, Erik F. and Jorn Veenstra (1999). Representing Text Chunks. In: *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics*. Bergen, Norway.
- Trushkina, Julia and Erhard W. Hinrichs (2004). A Hybrid Model for Morpho-syntactic Annotation of German with a Large Tagset. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain.
- Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind, New Series*, 59(236):433–460.
- Ule, Tylman (2002). DEREKO Linguistic Markup. Technical report, SfS, Universität Tübingen, Tübingen, Germany.
- Ule, Tylman and Sandra Kübler (2004). From Constituent Structure to Dependencies, and Back. In: *Pre-Proceedings of The International Conference on Linguistic Evidence*. Tübingen, Germany.

- Ule, Tylman and Frank Henrik Müller (2004). KaRoPars: Ein System zur linguistischen Annotation großer Text-Korpora des Deutschen. In: *Automatische Textanalyse. Systeme und Methoden zur Annotation und Analyse natürlichsprachlicher Texte*, (Eds.) A. Mehler and H. Lobin. Opladen: Westdeutscher Verlag.
- Ule, Tylman and Kiril Simov (2004). Unexpected Productions May Well be Errors. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*, vol. 5, pp. 1795–1798. Lisbon, Portugal.
- Ule, Tylman and Jorn Veenstra (2004). Iterative Treebank Refinement. In: *Proceedings of the 14th Meeting of Computational Linguistics in the Netherlands*. Antwerp, Belgium.
- Veenstra, Jorn, Frank Henrik Müller, and Tylman Ule (2002). Topological Fields Chunking for German. In: *Proceedings of the Sixth Conference on Natural Language Learning*. Taipei, Taiwan.
- Wall, Larry, Tom Christiansen, and Jon Orwant (2000). *Programming Perl*. Beijing et al.: O'Reilly, 3rd ed.
- Yoshinaga, Naoki, Kentaro Torisawa, and Jun'ichi Tsujii (2003). Comparison between CFG filtering techniques for LTAG and HPSG. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan.