Unsupervised Learning of Word Order Rules

Christian Korthals

Diplomarbeit im Fach Computerlinguistik
Universität des Saarlandes, D-66123 Saarbrücken
Betreuer: Prof. Dr. Hans Uszkoreit, Dr. Geert-Jan Kruijff

2nd October 2003

Wahrheitsgemäße Erklärung

Ich erkläre, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, den 02. Mai 2003

Christian Korthals

Acknowledgments

This work emerged from the NEGRA project at Saarland University, which is part of SFB 378. I profited greatly from this research environment.

I am grateful for the extremely helpful discussions with Stephanie Becker, Ralph Debusmann, Péter Dienes, Amit Dubey, Denys Duchier, Anette Frank, Valia Kordoni, Marco Kuhlmann, Daniela Kurz, Sebastian Padó, Christoph Scheepers, Tatjana Scheffler, Silvia Hansen-Schirra, Markus Schulze, Anne Schwartz and Zdeněk Žabokrtsk'y. Joanna O'Driscoll and Bobbye Pernice, as native speakers, helped me in cases of doubt with my English; Marco Kuhlmann was my advisor in typesetting; and Christian Behring always knew these little computer tricks.

I also profitted from conferences and workshops I was able to attend while I was writing this thesis, particularly from the COLING conference in Taipeh and the workshop on Linguistic Corpora and Logic Based Grammar Formalisms in Utrecht.

Finally, I want to thank my supervisors. I always received input when I needed it, but never in a restricting way.

Zusammenfassung in deutscher Sprache

Diese Diplomarbeit entwickelt, implementiert und evaluiert einen Ansatz zum unüberwachten Lernen von Wortstellungsregeln aus syntaktisch annotierten Korpora.

Die Entwicklung maschinenlesbarer Grammatiken für computerlinguistische Anwendungen, wie Übersetzungssysteme, Dialogsysteme, Grammatik-Korrektursysteme, usw., ist zeitaufwendig und anspruchsvoll. Ansätze zum Lernen von Grammatiken aus Korpora existieren, sind aber linguistisch oft wenig tiefgreifend. Auch vermischen diese Ansätze oftmals mehrere linguistische Ebenen, etwa Wortstellung und Subkategorisierung. Das entwickelte System zeichnet sich durch seine klare Modularisierung nach linguistischen Gesichtspunkten aus. Die gelernten Regeln sind theorieunabhängig und damit in verschiedene Grammatikformalismen übersetzbar. Die Regeln sind außerdem linguistisch feinkörnig. Das System hat den Anspruch auf Sprachen anwendbar zu sein, die sich typologisch stark in ihrer Wortstellung unterscheiden.

Die Eingabe des Systems ist ein syntaktisch annotiertes Korpus. Ein *climbing*-Modul identifiziert diskontinuierliche Knoten im Dependenzbaum, und verwendet einen aus der Topologischen Dependenzgrammatik (TDG) entlehnten Mechanismus des Kletterns, um das Phänomen zu behandeln. Das Lernsystem schließlich teilt Syntaxbäume in lokale Unterbäume, und verwendet ein *Präzedenzpaar*, ein einzelnes Vorkommen zweier Dependenten unter einem gemeinsamen Kopf, als Primitiv. Ein Graph-Algorithmus wird verwendet, um eine Worstellungsregel zu errechnen, die einer topologischen Beschreibung entspricht. *Automatic Feature Selection* ist ein Mechanisums, der iterativ einen Entscheidungsbaum-Lern-Algorithmus verwendet, um diejenigen linguistischen Merkmale auszuwählen, die relevant sind, um Wortstellung vorherzusagen.

Das System wurde vollständig implementiert, und auf dem Negra-Korpus deutscher Zeitungstexte evaluiert. Eine linguistische Beurteilung zeigt die Vergleichbarkeit mit Topologischen-Felder-Analysen aus der Literatur. Auch feinkörnige Regeln über Mittelfeld-Stellung werden gelernt. Die Regeln werden auch verwendet, um ein topologisches Korpus zu erzeugen. Dabei wird 97.6% Knoten-Recall erreicht. Ein Vergleich mit einem extern manuell erstellten Korpus zeigt 74.5% Übereinstimmung (unlabelled bracket recall).

Contents

Chapter 1

Introduction

Formal grammars of natural languages play a central role in many applications of computational linguistics; for example in dialogue and expert systems, in machine translation, text summarisation or grammar checking. However, grammars of natural languages that can be universally applied to different text types, domains and applications, with high precision and coverage, are not visible in the near future. Therefore, the development of new grammar fragments for specific purposes, the adaption of grammars and lexicons to new domains, and the development of grammars for new natural languages play an important role in research and practice. There is a large number of competing grammar formalisms, which differ in their formal expressiveness, the descriptive devices they use, and the linguistic assumptions they make. This means that the grammatical knowledge implicit in a grammar of a specific formalism is confined to this formalism and its processing devices. Inter-framework conversion of grammars often results in larger target grammar size, worse legibility of the target grammar, and sometimes a change of coverage. Generally, manual grammar development and maintenance is a highly skilled and time consuming task.

This insight resulted in the development of machine learning techniques that can acquire grammatical knowledge, given example sentences of the language to learn. Training data for these algorithms is available today for many languages in the form of corpora and treebanks of different sizes and domains. In the best case, adaption of a learning system to a new domain or language means retraining on different data. While some grammatical decisions can be successfully predicted with the aid of learning systems, for example part-of-speech disambiguation, algorithms that acquire a complete and linguistically valuable grammar from scratch are not state of the art. Although the linguistic depth of grammar induction systems is increasing, most contemporary approaches are still linguistically shallow, low in the level of linguistic insight or abstraction, and, just as with manual grammars, confined to specific formal assumptions. Many acquired grammars are "black boxes" in the sense that their acquired behaviour is implicit in weights or transition probabilities hard to interpret for humans.

This thesis develops a new approach to the acquisition of syntactic knowledge which takes these thoughts into consideration. The scope of the learning system is restricted to the acquisition of *word order* rules. Word order is a theoretically interesting and practically relevant subproblem of grammar learning. A linguistically clearcut phenomenon is identified and modelled, instead of intertwining linguistically separate phenomena into a single formal means. The learned rules are *symbolic* and human-readable. These characteristics result in easy integration of the learning system with other components of a grammar, learned or manually written. The assumptions built into the learning system are *not confined to a specific grammar formalism*, which has the advantage that the acquired knowledge can be made use of by different frameworks. The approach is *unsupervised*, which means that there is no need to feed the structures which are to be learned into the system.

The input of the system is a syntactically annotated treebank. During training, discontinuous configurations in the trees are identified in order to account for long-distance dependencies. Then, for each class of linguistic head found in the data, a *topological field description* is learned, which models the grammaticalised sequence of all dependent items that can occur under the head. The learning primitive is a pair of two elements under a common governor. With the aid of a *graph algorithm*, a partial order on these elements is *robustly* calculated. Since word order may depend on features from different linguistic levels, a component for *automatic feature selection* is added, which automatically learns a function from dependent word forms to the elements of the field descriptions. *Decision tree learning* (Quinlan (1998)) is used as a strategy for this task.

The system is evaluated on the Negra corpus of German newspaper texts (Skut et al. (1998)). Evaluation shows that the algorithm is able to discover an instantiation of the generalised topological fields assumed, which resembles traditional accounts of word order in many respects. It also shows that the learned rules can be successfully applied to convert the input treebank into a topological treebank. They can also be exported to a concrete grammar formalism, and used for parsing and generation. The learned rules model constraints implicit in the word order rules of a language as well as the variability that occurs in word order. Automatic feature selection discovers fine-grained linguistic rules.

The structure of the thesis is as follows. Chapter 2 gives an introduction to machine learning, with a special focus on the state of the art in grammar induction, as opposed to manual grammar development. Chapter 3 provides the linguistic basic of the thesis, analysing word order cross-linguistically as well as word-order phenomena in German. It also introduces TDG (Duchier and Debusmann (2001)), a dependency grammar formalism, from which some notions will be borrowed.

Chapter 4 develops the word order learning architecture, introducing one component after the other. Section 4.4 shows how discontinuous configurations are treated, Section 4.5 defines precedence pairs as a primitive for learning, and Section 4.6 provides an algorithm that builds graphs from these primitives, which are interpretable as field descriptions. Section 4.7 deals with making the approach robust, and introduces automatic feature selection. Section 4.8 shows how to export and make use of the learned rules. Chapter 5 describes an implementation of the learning architecture in Java, using C4.5 for decision tree learning, and shell scripts for evaluation.

In Chapter 6, the system is evaluated on the Negra corpus. Section 6.2 evaluates internal parameters of the system, showing that little data suffices to arrive at good results, but also revealing convergence problems for some parameters settings. Section 6.3 evaluates the application of the learned rules to producing a corpus of topological structures from the input treebank. 97.6% recall on nodes during rule application is achieved. A comparison with a manually created, different topological corpus yields 74.5% unlabelled recall on brackets. Section 6.4 presents a preliminary experiment on applying the learned rules to TDG generation. Section 6.5 judges the learned rules on linguistic grounds, finding similarities to topological field analysis in the literature.

The appendices include a sample of learned word order rules, an overview of the corpus annotation scheme, and mathematical definitions.

Chapter 2

Machine Learning

This chapter provides an introduction into machine learning, with a focus on grammar induction. In Section 2.1, I define and classify machine learning systems, and characterise necessary steps in developing and evaluating a ML system. In Section 2.2, I discuss approaches to the learning of syntactic structures from corpora, and compare these approaches to manual grammar development. I restrict the focus to systems that learn word order rules in Section 2.3. Section 2.4 provides an introduction to decision tree learning, a standard machine learning technique I will be using, which has been applied to discover linguistically fine-grained rules in data.

2.1 Classification of Machine Learning

A machine learning system (Russel and Norvig (1995), Mitchell (1997), Hutchinson (1994)) is a computer program that automatically improves with experience. It can be rendered as an "agent" in the sense of artificial intelligence, who perceives its environment in the form of *examples*, develops hypotheses from these examples consistent with the *background* (or prior) *knowledge* it has, and arrives at a set of *classifications*. This classification enables the agent to make predictions about unseen future examples. Russel and Norvig (1995) expresses this argument as a formula $Background \wedge Hypothesis \wedge Examples \models Classifications$.

Machine learning systems can be classified into *supervised* and *unsupervised*. A supervised learning scenario is one where the classification is given with the examples during learning: the *training data* includes positive or negative feedback. In an unsupervised scenario, in contrast, the classification is created by the system without any feedback on how "good" or "bad" its actions are.

Imagine an agent who wants to learn to play the piano. The agent has sensors (eyes, ears) which perceive the piano playing of a piano teacher, and effectors (hands), which can achieve results. In a supervised scenario, the agent also has an external criterion of success, provided by the teacher. An unsupervised scenario, in contrast, is one where the teacher does not talk to the student.

This thesis follows an unsupervised learning approach for a simple reason. There is no clear external criterion of what a "good" word order rule is. A possible candidate for such an external criterion might be the performance of a parser or generator that employs the learned rules on a test set. However, such a strategy would rely on a concrete parsing formalism, and therefore on a concrete theory of word order built into the grammar formalism. It is an aim of this thesis, though, to discover word order regularities in the data without making very specific theoretical assumptions about word order.

This does not mean that there are no assumptions at all. The task of unsupervised learning without any prior assumptions, pure inductive learning, is "generally very hard", and not pursued any more nowadays (Russel and Norvig (1995, p. 557, 626)). As is stressed by Daelemans and Hoste (2002), theoretical research has shown that no inductive algorithm is in fact universally better than any other. Thus, assessing the prior assumptions of a machine learner is essential. A related issue is the *bias* of a learning system. This term refers to how one hypothesis is preferred over the other during search through the *hypothesis space*. A principle used in many systems is known as Occam's razor. It states that a simple, or short explanation for a phenomenon is to be preferred over a more complex explanation. Russel and Norvig (1995, 559) credit Mitchel (1980) for emphasising the importance of bias in inductive learning.

Machine learning systems are evaluated by using the learned rules to classify unseen *test data*. In supervised learning, this is easily done by omitting the classifications from the training data, and comparing the predicted classification to the original classification (the Gold Standard). Unsupervised learning can only be indirectly evaluated, or compared to intuitions about the data, due to the lack of an external measure of quality.

Several factors can cause errors of a learning system. A learner *overfits* the data if it accounts for the examples in the training data correctly, but fails to generalise this behaviour to unseen data. It *underfits* if it fails to account for a trend that is in fact observable. Errors may also be due to wrong prior assumptions. The piano playing agent overfits if it can play only the very pieces it was taught. It underfits if it uses its forefinger for playing only: it will fail miserably on fast pieces. This example also illustrates the importance of a well-suited set of prior assumptions: an agent that starts with some knowledge about fingering in piano playing can learn faster, and achieve better results. For the learning system to be developed in this thesis, this means that a well-considered choice of theory independent linguistic assumptions must be found.

One of the most important design decisions when building a machine learning architecture is the *representation* of the domain of application. This includes the formalisation of the the input examples as well as the output classifications. Sometimes, but not always, the same language, e.g. predicate logics, is used for these tasks.

The input examples should include all properties of the data which are possibly relevant for making predictions. The piano player for example is unlikely to achieve an efficient fingering if it cannot see the teacher's keyboard.

The output, as well as the internal architecture may represent data on a *symbolic* or a *subsymbolic* level. Subsymbolic machine learning techniques are "black boxes", because their internal structure is represented in a non-human readable way. An example are neural net approaches, which employ a *hidden* layer of nodes, not interpretable in terms of the input and output symbols used. A symbolic learning scenario, in contrast, produces human-readable rules; symbolic systems are "glass boxes".

In addition, the output classification acquired by the system may be *statistical* or *non-statistical* in nature. In a non-statistical setting, an example either belongs to a concept, or it does not, while in a statistical setting a probability distribution over classes is learned. I follow a symbolic, non-statistical approach in this thesis. This has the advantage that decisions taken by the system are always easily traceable, and are open to linguistic assessment.

Since there is a rich variety of machine learning algorithms today, the choice of an appropriate algorithm is another important step when developing a learning system. Hutchinson (1994) provides a helpful diagram for this task. A comprehensive overview of symbolic learning is given in Briscoe and Caelli (1996). Some approaches are general paradigms, while others are tailored to a specific learning task. I refer to a number of techniques which are relevant to this thesis. The term *clustering* is very general, and refers to unsupervised classification. *Memory based*, instance based, or lazy learning employs the idea that the set of examples is not abstracted until an unseen instance from the test data is seen. *Decision tree learning* uses trees to represent decisions during classification, and is outlined in more detail in Section 2.4.

The formal study of what classes of problems are theoretically learnable, learnability theory, is relatively new, and will be only mentioned in passing in the following chapter, when directly relevant to grammar learning. Natarajan (1991) is an introduction to theoretical machine learning. The field received much attention in the 1990s (Hutchinson (1994, 387)). Valiant (1984) provided a general framework for learning, PAC ("probably approximately correct") learning. Today, there seems to be a clash between theoretical results on one side, and practical applications on the other. Hutchinson (1994) gives this advise to the designer of a machine learning system (p. 388)

> I urge you, be not dismayed. If you think you can devise a practical learner for a specific situation, then you are probably right.

2.2 Grammar Induction

Human learning, and especially human learning of natural language, has always served as a real-world archetype of machine learning, and the idea of modelling human language learning on computers is not a new one. It is, however, not the goal of this thesis to make a contribution to human learning of natural language, but rather develop a machine learning system with practical use and relevance to linguistic theory. It is still interesting to compare the architecture to human learning. Gold (1967) investigated the question whether children can possibly learn their mother language merely from the "poor" input they receive: positive examples of grammatical sentences. Their learning seems almost unsupervised, as they are seldom corrected. Gold's controversial hypothesis was that there needs to be some prior knowledge about the structure of language in order to enable children to learn a language. Chomsky (1981) integrated this insight into his Universal Grammar, and stipulated innate universals of grammar in addition to learnable "parameters". Niyogi and Berwick (1996) theoretically investigate learnability in the limit in the sense of Gold in a very simple 3-parameter model. They conclude that local maxima are problematic for search, and state that "how the sample complexity scales with the number of parameters is an important question that needs to be addressed" (p. 189). The comparison with human language acquisition suggests, however, that – at least given some prior knowledge – learning rules about language should be possible.

Not only is automatic learning of grammar rules possible, it is also desirable. The goal of finding the "true" grammars of languages is unsolved, and natural language applications need computational grammars specifically tailored to the application, domain, natural language, and grammar formalism. There is a large number of formally different grammar formalism. Inter-framework conversions have been proposed, but are not unproblematic. For example, grammar size may increase considerable with conversion (Yoshinaga and Miyao (2002)).

Grammar development is a time consuming task. The development of a large-scale English XTAG grammar (Group (2001)), e.g., has been going on for almost a decade now, with 8 grammar developers and 6 system developers working at this task. In many grammar frameworks, the task of modularizing grammar development, and distributing the work among grammar writers is problematic.

Evaluation figures of grammars depend heavily on the test corpus, or test suite used. Group (2001) report to have been able to parse only 20% of the sentences of a corpus of weather reports, and a manual adaption of the grammar was necessary to increase this figure to 89.6%, which is a state of the art figure (p. 305). Another evaluation of the same grammar shows 84.2% dependency accuracy on NP chunks from the Penn Treebank Marcus et al. (1995), restricted to sentences of length 15. An example of a hand-crafted high-coverage grammars of German is Müller (1999).

Kinyon and Prolo (2002) classify grammar development strategies into four categories, with the aid of two features: hand-crafted vs. automatically generated, and high level of syntactic abstraction vs. low level of syntactic abstraction. They state:

> Although linguistically motivated, developing and maintaining a totally handcrafted grammar is a challenging (perhaps unrealistic?) task ... Small and even medium-size grammars are not useful for practical applications because of their limited coverage, but larger grammars give way to maintenance issues.

Due to the problems of manual grammar development, systems that can learn syntactic structure from POS-tagged input texts have been developed. An early example is Berwick (1985), who assumes, following Gold, a context-free grammar as a grammar-framework, and learns production rules with the aid of an incremental shift-reduce parser, which starts with an empty grammar, and adds production rules to the grammar as soon as parsing fails. The algorithm proposed by Wolf (Hutchinson (1994, 168)) follows the same strategy.

Both algorithms are instances of what Klein and Manning (no year) call "structure learning", i.e. discovering syntactic production rules from a sequence of part-of-speech tags. They employ a matrix of word span indices, and learn constituents (distituents, rather, as they call it) with a version of the EM algorithm, a clustering algorithm. The search space is restricted to "binary, tree-equivalent bracketings". The algorithm is run on the POS tags of the corpus, as well as on induced POS tags. They report an f-value of 71% (which is an average of recall and precision) on non-trivial brackets.

All approaches compared so far work on POS-tagged text as input. Today, however, large syntactic databases (treebanks) are available for many languages, which include syntax trees in addition to POS-tags. Current treebanks often annotate syntactic structures in the form of labelled phrases, following the Penn Treebank for English (Marcus et al. (1995)), or dependency relations, following the Prague Dependency Treebank for Czech (Hajič (1998)). The German Negra and Tiger treebanks (Skut et al. (1998), Brants et al. (2002)) combine notions from dependency grammar and phrase structure grammar. All treebanks also feature disambiguated part-of-speech information, and sometimes additional morphological information. Often, syntactic functions are also annotated[1]. I continue with a comparison of approaches to grammar induction from treebanks.

A papers by Charniak (1996) lead to a number of approaches, known as "treebank grammars", which simply read off probabilistic context free grammar rules (PCFG) from a given treebank, and employ these rules for parsing. Charniak reports 80% accuracy (i.e. number of non-crossing brackets) of such a grammar, after some manual corrections. More recent numbers are around 90%. The problem of such a treebank grammar is, however, that its restrictive power is almost zero, i.e. it will assign an analysis to a random POS sequence, and produce a very large number of ranked parses for a given POS sequence. A solution of the overgeneration problem lies in lexicalisation of these grammars. With lexicalization, however, undergeneration becomes a problem, and unrealistically large training corpora would be necessary. Collins (1999b,a) presents a lexicalized, statistical parser applicable to English and Czech.

Xia (1999), Xia et al. (2001) present an approach to extracting lexicalized TAG grammars for typologically different languages from treebanks. After extracting elementary trees (which are not distinguishable from rules in the TAG formalism) from the input treebank, rule based post-processing steps are applied to distinguish arguments from adjuncts, and identify coordinations. There is no generalisation on the word order level, though.

There has been significant improvement in grammar learning in the last years, but the opposition identified by Kinyon and Prolo (2002) has not yet been resolved: Manual grammars are linguistically insightful and human readable, but time consuming and hard to create and maintain. Learned grammars lack linguistic abstractions, which makes them hard to interprete for humans, and less valuable for natural language applications.

There has been some progress towards adding linguistic abstractions to learned grammars. In the TAG approach, adjuncts and arguments are identified, employing the linguistic abstraction of a subcategorization frame. The same measure is taken by Sarkar and Zeman (2000). This is not the case for Klein and Manning (no year), but they add linguistically valuable information by learning from induced POS tags. Lexicalization of treebank grammars also makes the grammars more fine-grained. While these are all steps towards more abstract learned grammars, the goal has not yet been reached.

Furthermore, none of the approaches compared apply linguistic abstractions to word order phenomena. All approaches either rely on non-crossing phrase structures, and consequently face problems with discontinuous realisations (see Section 3.2), or do not involve permutation rules to account for word order variation (as in the TAG approach). In context-free production rules, the order of elements on the surface is not conceptually separated from the obligatory appearance of subcategorized elements. This immediately translates into coverage

[1]For an example of a treebank sentence, the reader is referred to Figure 4.2, page 34.

and overgeneration problems when applied to languages with free word order. Berwick, e.g., explicitly treats topicalized constructions like "Candy, Sally likes" (p. 184) as noise in the data, with the desired behaviour of his system being not to be distracted from learning the more general rule.

A system that learns word order rules from a treebank in a linguistically fine-grained and abstract way is therefore desirable. Such a system is a step into the direction of Kinyon and Prolo (2002)'s "learned" grammars with a "high level of syntactic abstraction" (type D grammars).

2.3 Word Order Learning

As opposed to the many approaches toward the learning of syntactic structures or parameters, there are only few approaches specifically devoted to the learning of word order from corpora. This section reviews three approaches that the author is aware of.

Villavicencio (2000) presents a system that models human acquisition of the predominant word order of arguments with respect to their heads on a 1000 sentence corpus of English child-parent conversation. She assumes a universal grammar in the sense of Chomsky (1981), and binary syntactic structures, along with a lexical type hierarchy, which includes types as "noun", "transitive verb", "ditransitive verb", "transitive control verb", along with a classification of the arguments of these classes of heads (subject, object, direct object). Her algorithm starts with the initial assumption that all arguments occur on the same side of the head, and iteratively refines this hypothesis for specific subclasses in the hierarchy, where subclasses inherit their parent's feature by default. Like in some of the approaches discussed in the previous chapter, a parser is part of the system, which adapts the grammar on failure. The input of the parser is a sentence, annotated with a logical form (i.e. a deep-syntactic structure). At any time, the current direction value of a node in the hierarchy is the highest probability value in the data, and a threshold determines in which case a hypothesis should be adapted.

The aim of Villavincencio's system is to find rules about the predominant word order of classes of arguments with respect to their head. This view is in principle not able to make predictions about the order of arguments of the same head among each other. Baldridge (2002, p. 199) identifies that when introducing new arguments of a head, according to the type hierarchy, she relies on a property of the English language, which does not extend to other languages, especially those with a freer word order. In fact, the system has been run on English only. Another problem is that Villavincencio seeks to find predominant word order, a view which conceptually prohibits finding rules about variation in word order: the position of an argument is always predicted to be either to the left, or to the right, and it is only through the additional assumption of rules of commutativity that word order variation is accounted for. As a last point of criticism, there does not seems to be an account of adjuncts, as opposed to arguments, although adjuncts occur widely in free data.

Following Charniak (1996), Becker and Frank (2002) trained a probabilistic parser on a treebank of topological structures, which encode word order regularities of German, a language with an intermediate amount of word order freedom. They used the Negra treebank of German as an input corpus. Tree-transformation rules were manually developed with the aid of a tree-transformation language, which creates the topological corpus from the treebank. This corpus had to be manually corrected in order to yield a gold standard treebank for evaluating the topological parser, reaching a labelled precision of 93.0% and a labelled recall of 93.7% of the conversion rules.

The goal of Becker and Frank was the supervised training and evaluation of the topological parser on the manually created gold standard. This means that a word order treebank needs to be created with the aid of manually written rules and manual post-correction for every language that is to be parsed, if this approach is to be extended to other languages. The performance of the parser is good for this supervised scenario (93.4% precision, 92.9% recall is the best performing model), but all the relevant word order regularities the system is expected to learn, along with a complete linguistic theory of word order is assumed. This contrasts sharply with the goal of this thesis of unsupervised learning.

The last approach is Gamon et al. (2002). They use decision tree learning, a supervised machine learning technique which will be outlined in the following section, to learn rules about word order in German. Their scope however is extremely narrow. The learning system merely predicts whether a given clausal complement (a relative or complementizer clause) should be locally realised, or non-locally displaced to the right on the syntactic surface. Thus, the output of the system is a yes/no-decision. The input is a richly annotated treebank. They identify a wide range of linguistic features relevant to predicting the decision, among them category and syntactic relation of both the clausal complement node and its father and grandfather, phrase length, etc.

2.4 Decision Tree Learning

Decision Tree Learning (Briscoe and Caelli (1996, 31ff)), and more specifically top-down decision tree induction (Quinlan (1998)) is a supervised machine learning technique. In this thesis, Quinlan's C4.5 system is used. Typically, the output parameter has few discrete values, while input parameters can have many values, discrete or numeral. A standard example of decision tree learning is weather forecast, where the set of possible outcomes can be assumed to be restricted (e.g., sunny, cloudy, rain, snow), and input parameters are possibly complex (e.g. temperature, wind strength, air pressure, yesterday's weather, ...).

Decision Tree Learning has also been applied to linguistic topics, e.g. to automatic assignment of dependency roles (thematic roles, roughly) from syntactic information (Zabokrtsky (2001)), and even to word order phenomena, as shown in the previous section.

C4.5 employs a *greedy* strategy: it analyses the input data, and chooses a single parameter whose values predict the outcome best. It then sub-classifies the data according to these values, and reiterates. The strategy is greedy, because there is no backtracking involved, and a decision is taken at every iteration step which appears promising locally, but need not be globally optimal.

The rules generated by a decision tree learner can be represented in tree format, as in Figure 2.1 (adapted from Zabokrtsky (2001, p. 57)), where node labels are feature to be picked, edge labels are the values of the dominating

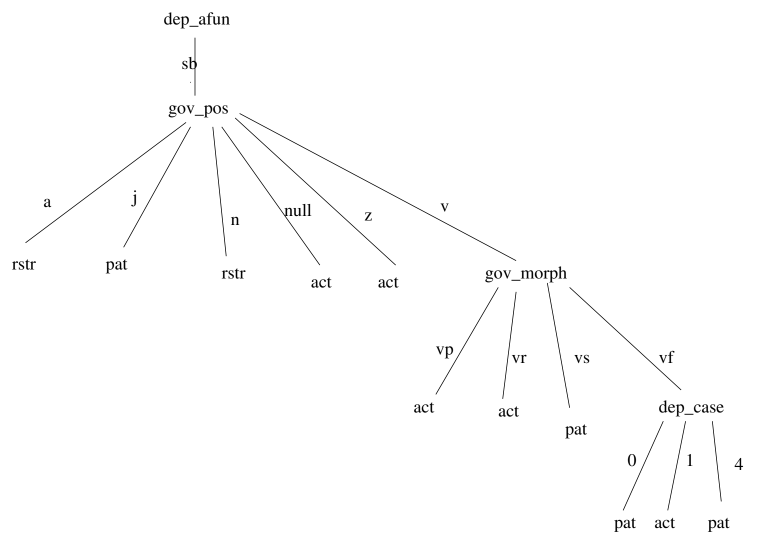


Figure 2.1: An example decision tree, from Zabokrtstky. Here, output values are thematic roles (actor, patient, restricted), and parameters are syntactic features of governor and dependent nodes.

node's feature, and leaves are the predicted output value.

Decision Trees can also be converted into rules by traversing the tree and generating a rule for each path from the root to a leaf (Quinlan (1998, p. 45ff)). Rules have a set of conditions on the input parameters, and a prediction for the output value, and can be augmented by a probabilistic measure of reliability. One of the rules implicit in Figure 2.1, is e.g. *dep_ afun=sb* $\wedge$ *gov_pos=a* $\Rightarrow$ *output=rstr.*

One important characteristic of converting trees to rules is that, given a set of input parameters and values, there need not necessarily be a unique rule in the set of generated rules whose preconditions match (Quinlan (1998, p. 47)). It is therefore necessary to posit an *order* on generated rules, in order to determine which rule to pick in case of multiple match. This order can be established on probabilistic grounds.

Decision Tree Learning will be used for Feature Selection (Section 4.7) in this thesis as a submodule of the larger learning architecture. It is suitable because the technique is fast, produces symbolic output, and the input data is complex.

Chapter 3

Word Order

This chapter lays the linguistic foundation of the thesis. In Section 3.1, I argue that an investigation into word order is crucial if a grammar induction system is to be universally applicable to a range of typologically different languages. I provide a data-driven analysis of word order phenomena in German in Section 3.2, employing standard descriptive terminology. Finally, I review formal approaches to these word order phenomena in Section 3.3.

3.1 Typology of Word Order

Since Greenberg (1966), languages have been typologically compared with respect to their word order. These studies revealed that among the many mathematically possible orders, only few are in fact attested by the languages of the world. Typological analysis is relevant to this thesis, because the choice of linguistic theory should be general enough to capture the phenomena that occur across languages. Only then will it be possible to successfully apply the word order learning system to different languages.

The primary aim of topological studies has always been to discover language-universal statements of the form "if a language has prepositions rather than postpositions, and places the adjective after the noun, it will (always, or likely) place a relative clause after the noun". Such a statement is called an implicational universal. Implicational universals may be statistical or non-statistical in nature.

Two assumptions are normally made in the typological literature (Hawkins (1983, p. 11ff)). First, a notion of *basic word order* is assumed: In English for instance, the basic word order at the sentence level is Subject-Verb-Object. Second, it is assumed that subjects, objects, relative clauses, etc. are identifiable on semantic grounds across languages. Both assumptions are controversial.

I will not go into the details of establishing categories cross-linguistically. This issue is side-stepped in this thesis by relying on the annotation of the corpus. It is important to note, though, that a corpus is normally created on the basis of a tagset which describes the specific target language rather than cross-linguistically valid categories. Running the system on a corpus with typologically motivated categories would certainly yield different, and typologically interesting results.[1]

I now turn to the notion of basic word order. According to Hawkins (1983), "the biggest problem for a notion of basic word order is to be found in the ordering of the arguments of the verb at the sentence level". He continues with a definition (p. 13).

> I am going to use the term 'doubling' to describe the situation in which one and the same modifier category ... can occur both before and after its head in a given language

In the face of doubling, he provides (slightly simplifying) two criteria with which to establish the basic doublet. The first one is simple, and could be easily discovered with the aid of a machine learning system: where two order variants exist, the more *frequent* one is the basic one. The second criterion relies on the notion of linguistic *markedness*. A construction is marked if it serves a special purpose in the language system. Kruijff (2001) makes this notion more precise by modelling interactions with information structure, particularly prosody and contextual boundness, case-marking, etc.

The view taken in this thesis is that markedness can – in theory – be reduced to frequency, if the corpus is annotated with all relevant features from different linguistic levels. An approach is therefore desirable that is independent of the concrete corpus annotation scheme. Due to the problem of doubling, I will not assume that a "basic" word order can always be found. Rather, I opt for an approach where all relevant linguistic features are included into the rules the system learns.

Hawkins gives examples where his strategy fails: For English genitives, he does not commit to either GenN or NGen. For German sentence level order, he neither commits to SVO nor to OVS (p.14). In these cases, there are no predictions in Hawkin's theory, because none of the implicational universals apply. This leads to abandoning Greenberg's SV and VO as type indicators (p.16).

There is the implicit assumption in the definition of doubling that the head plays an important role. The term doubling is not used, e.g., to refer to cases where indirect and direct objects permute, because the head is not involved in this order variation. I will come back to this issue in Section 4.7.3.

While Hawkins, after mentioning the problem of basic word order, is not concerned with it any more, Steele (1978) was the first to investigate word order *variation* in a single language more closely. She concentrates on the order of subject, verb, and object, and establishes a classification of languages according to which kinds of rearrangements they allow, assuming that the types VOS, VSO, SOV, and SVO exist.

First, she observes that each logically possible variation is at least attested by one language of a type. She then formulates *constraints* on the variation that occur within a language, and classifies languages according to whether they violate these constraints. Her constraint A is that the verb position should not be changed. Her constraint B is that a rearrangement where O precedes

[1]For example, the input corpus has two different edge labels for pre-nominal and post-nominal genitives in German (edgelabels GL, GR). Due to this distinction, the order learning system *cannot* learn detailed rules about the placement of genitives in German, because the – seemingly – best explanation is the occurence of the GL or the GR tag.

S should be omitted. A "rigid" word order language is one whose occurring rearrangements do not violate these constraints. A "free" word order language is one that violates all constraints, and a "mixed" word order language is one that violates some. Steele still excludes "highly marked" constructions from her theory. Among them are non-local topicalisations like "That man I dislike". This means that she does not make any predictions about these rearrangements.

Kruijff (2001, p. 202) extends Steele's classification by applying it not only to matrix clauses, but also to dependent clauses. He also extends her language sample, and classifies German as a mixed word order SVO language which obeyes Steele's constraint A.

Korthals and Kruijff (2003) show that languages do not only differ in the amount of head-local rearrangements (which they call "scrambling"), but also in the amount of non head-local rearrangements ("discontinuity"). Head-local rearrangements correspond loosely to Hawkin's "doubling" and can be described by Steele's constraints on variation, while discontinuity is excluded from Steele's theory. Korthals and Kruijff (2003) report that 11.05% of the phrases of the German Negra treebank are discontinuous, opposed to 1.03% in English. They measure the amount of scrambling by dividing the number of ordered dependency trees by the number of unordered dependency mobiles, and achieve a "scrambling factor" of 1.02 for German, but only 0.33 for English S nodes. The next section will investigate local and non-local word order variation in German more closely.

3.2 Word Order in German

While the topological perspective is important in order to find a linguistic theory that is general enough to be applicable cross-linguistically, it is often not fine-grained enough to account for the word order phenomena actually found in a particular language, either because certain phenomena are explicitly excluded from the theory, or because of a clash between what is considered a theoretically interesting exception and what is actually a frequently occurring exception. This section therefore takes a closer look at German word order phenomena in a data-driven way. First, I investigate constraints on word order in German, which cannot be violated, and compare them to variation in word order and softer constraints. Then, I investigate discontinuity in German. Finally, I review descriptive approaches that are traditionally applied to account for the phenomena listed, and are widely accepted.

Example 1 below is the first sentence of the Tiger corpus (Brants et al. (2002)). Counting the proper name as a single unit, there are $5! = 120$ mathematically possible permutations of its word forms. Only about 6 of them can be clearly judged grammatical without doubts, some more are at the border of acceptability, or have a different truth-conditional meaning. Below are some example permutations of the sentence.

(1) Ross Perot wäre vielleicht ein prächtiger Diktator

(2) *Ross Perot ein wäre vielleicht prächtiger Diktator

(3) *Ross Perot wäre vielleicht prächtiger Diktator ein.

(4) *Vielleicht Ross Perot wäre ein prächtiger Diktator.

(5) Wäre Ross Perot vielleicht ein prächtiger Diktator?

(6) ... Ross Perot vielleicht ein prächtiger Diktator wäre.

(7) Vielleicht wäre Ross Perot ein prächtiger Diktator.

Examples (2) and (3) show that there are many permutations of the sentence which obviously violate a clear and hard constraint of German syntax. A determiner can neither be dislocated from the noun it specifies, nor can it occur after it. Examples (4) and (5) demonstrate the violation of another hard constraint of German syntax, which refers to the position of the verb. In sentence (4), the verb takes the third of four positions, which results in ungrammaticality. If the verb is in first position, though, the sentence is again grammatical, but can only be interpreted as a question, rather than a declarative sentence. If the verb is at the end, as in example (6), the sentence is acceptable as a subordinate clause only. Example (7), finally, demonstrates that, in spite of these constraints, German does allow for some word order variability, which is not directly explicable by sentence type or syntactic factors. Examples (1) and (6) do not differ in meaning, but they do differ in the way they behave in discourse, without going into a theoretical analysis at this point.

There is some more variability in German syntax, as the following set of examples, permutations of Tiger sentence 17, illustrates. In these examples, a judgement of grammaticality is less easily achieved than in the previous set of examples, and the classification into acceptable and unacceptable reflects the intuitions of the author only.

(8) ... daß sich ein Dogmatiker in Washington schwer tun würde

(9) ... daß ein Dogmatiker sich in Washington schwer tun würde

(10) ... daß sich ein Dogmatiker in Washington würde schwer tun

(11) ? ... daß sich in Washington ein Dogmatiker schwer tun würde

(12) *? ... daß in Washington sich ein Dogmatiker schwer tun würde

(13) * ... daß in Washington ein Dogmatiker sich schwer tun würde

(14) * ... daß ein Dogmatiker in Washington sich schwer tun würde

There seems to be no syntactic, semantic or pragmatic difference between sentences (8) and (9) whatsoever, nor a difference in acceptability. Sentence (10) seems to differ from the previous sentences in register. Sentence 4 differs in information structure, but appears to be considerably worse than sentences (8) or (9) to the author. Sentences (11) to (13) are probably unacceptable, but (13) is arguably acceptable with a slightly different semantics, where "Washington" is a dependent of "Dogmatiker".

The previous sentences showed that there are hard constraints as well as variability in German data, causing differences on various linguistic levels, and different grades of acceptability. In the following I will concentrate on a subclass of word order variation in German, *non-local,* or *discontinuous* word order phenomena. While the notion of non-locality differs among syntactic frameworks, a non-local realisation of a dependent under a governor can be defined

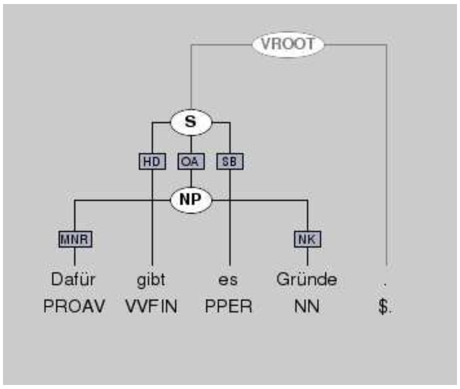


Figure 3.1: Tiger Sentence 86, with a non-local realisation of "dafür"

as one where foreign material occurs between the two items. The annotation scheme of the corpus, which will be described in Section 4.3, allows to identify these cases easily, as it represents non-local realisations with crossing edges, as shown in Figure 3.1.

100 sentences from the Tiger corpus were searched for non-local phenomena. Some examples of sentences exhibiting discontinuity are now given, and will be classified in the following. In each of the examples, the two or more bracketed word forms constitute a discontinuous phrase.

(15) Und ein anderer Manager vermutet, daß [sich] ein Dogmatiker wie Perot [in Washington schwer tun] würde. (17)

(16) [Nun] werden sie [umworben]. (64)

(17) [Da] sind [sich] alle [einig]. (47)

(18) [Dafür] gibt es [Gründe]. (86)

(19) [Daß Perot ein Unternehmen erfolgreich führen kann, davon] sind selbst seine Kritiker [überzeugt]. (6)

(20) [Es] ist wirklich schwer zu sagen, [welche Position er einnimmt]. (36)

(21) Allerdings glaubt fast die Hälfte der Chief-Executives, daß Perot durchaus [Chancen] habe, [die Wahl im November zu gewinnen], wenn er kandidiert. (11)

(22) Die Getreideproduktion wird [voraussichtlich 10 Mio Tonnen geringer] ausfallen [als geplant], während gleichzeitig die Bevölkerung um 18 Millionen Menschen steigt. (96)

(23) [Geschäftemachen] ist seine Welt [und nicht die Politik]. (45)

(24) [die Frage ist nur], meint ein Finanzexperte [ob er ins Weiße Haus einziehen kann], ohne uns vorger zu sagen, was er eigentlich machen will. (48)

Examples (15) and (16) are the most frequently occurring class. Here, the corpus annotates a discontinuous VP.[2] By convention, the subject is annotated as a dependent of the main auxiliary verb, and non-subject arguments as dependents of the embedded non-finite verb. Dependents from the embedded predicator however can be realised discontinuously in the domain of the auxiliary verb. The auxiliary verb may be a modal or auxiliary to build complex time forms, or constitute the passive-construction as in example (16). Examples (17) and (18) (presented before) illustrate that also material dependent on embedded adjectives or nouns can be non-locally fronted to the left-hand side. Example (19) will be analysed as an instance of what Höhle (1986) calls K-field and what is also known as the Wackernagel position. Here, the fronted modifier of "überzeugt" occurs twice and compactly, once as a full NP and once pronominalised. Just as in example (20), the corpus annotates these cases as instances of "place-holder-phrases", with a placeholder element (es, davon), and a phrase it substitutes. Example (21) is a representative of a large class of right-extraposed clausal material. It illustrates right-extraposition of a complement clause of a nominal. Another important and equally behaving class are relative clauses. Example (22) illustrates a discontinuous comparative phrase, which is also frequently occurring, and example (23) is an instance of asymmetric coordination, where the second conjoint is extraposed to the right hand side. Example (24) enters the area of discourse structure. Here, the corpus annotation is somewhat inconsistent. Sometimes these cases are analysed as parentheses, sometimes as placeholder phrases, and sometimes as "discourse level constituents".

The constraints on, as well as the variation in continuous and discontinuous realisations of German have traditionally been analysed with the aid of *topological field models* (Drach (1963)). A more recent outline is Höhle (1986). The important descriptive grammars of German, e.g. Eisenberg (1999), employ this model as well.

A topological field analysis starts at the sentence level, with the finite verb of the main clause as a central element. Furthermore, topological field analyses normally start with simple sentences, excluding complex cases of coordination and ellipsis. The first distinction a topological analysis makes is to classify main clauses into *verb initial (V1), verb second (V2) and verb final (VF) clauses* (Eisenberg (1999))[3]. The main functions of each of the sentence types are declarative sentences for verb second clauses, and imperatives and yes/no-questions for verb initial clauses. Verb final clauses occur mainly as subclauses. Höhle (1986), however, gives a range of invented, but quite realistic examples, which demonstrate that this association of sentence type and syntactic or pragmatic function is merely loose: there are exceptions in all cases, and in either direction.

Table 3.1 provides a topological field analysis of sentences from an initial portion of the Tiger corpus, some of which were presented earlier. The basic differentiation of V1, V2 and VF clauses is reflected in the tables. It is remarkable, though, that V1 clauses occur infrequently in the corpus data, due to the rather restricted use of this clause type to imperatives, yes/no-questions, and

[2]Although this may be a controversial linguistic decision, it is an aim of this thesis not to change the input corpus annotation.

[3]Eisenberg (1999) speaks of *Verberst-, Verbzweit-* and *Verbletztsatz.* The Duden grammar Drodowski and Eisenberg (1995) speaks of *Stirnsatz, Kernsatz, Spannsatz* respectively. Höhle (1986) speaks of F1-Satz, F2-Satz, E-Satz respectively. I use verb initial, verb second and verb final in this text.

verb initial clause (V1)

coord	wackernagel	finite	mittelfeld	verbal complex	nachfeld	Tiger #
Oder		brauchen	wir eine öffentliche Auseinandersetzung mit ihnen?			117
Aber		schauen	Sie, ...			124

verb second clause (V2)

	coord	wackernagel	vorfeld	finit	mittelfeld	verbal complex	nachfeld	Tiger #
		Daß Perot ein Unternehmen erfolgreich führen kann	davon	sind	selbst seine Kritiker	überzeugt		6
			Es	ist	wirklich schwer	zu sagen	welche Position er einnimmt	36
... viele Unternehmen, die meinen,			Perot	sei	einer von ihnen, ...			19
			Was	bewirkt	ihrer Ansicht nach ein solches Verhalten?			103

verb final clause (VF)

coord	"c"	mittelfeld	verbal complex/finite	nachfeld	Tiger #
	daß	Perot ein Unternehmen erfolgreich	leiten kann, ...		6
	... der	heute in fünf Konzernen im Aufsichtsrat	sitzt		18
	... daß	Perot durchaus Chancen	habe,	die Wahl im Novermber zu gewinnen, ...	11

Table 3.1: Topological field analysis

ART	ADJ	SBST	NGr	PrGr	S
ein	neues	Buch	dieses Autors	mit vielen Bildern,	das uns erstaunt

Table 3.2: Noun phrase topology (from Eisenberg)

certain optative constructions in spoken language. The tables are taken from Höhle (1986), but some of his abbreviated field labels were replaced by more widely used terms.

I will refer to an analysis according to the topological field approach as a *(topological) field description.* I conceive of a field description as an assignment of linguistic elements (phrases) to *fields*, and assume the sequence of fields of a field description as totally ordered. A field can constrain the number of elements which can be realised in it. I will refer to this property of fields as *field cardinality*. There are also restrictions on the linguistic form and function of elements which can occur in a field. In the descriptive literature, these restrictions refer to a rather shallow syntactic analysis of the elements. In the following, the fields that occur in Table 3.1 will be described in terms of the elements they can hold and their cardinality (unless otherwise stated, a field can hold zero or more elements).

Most fields are common to all clause types: *coord* is assumed to hold a single optional coordinator. *finite* is a field which holds the unique finite full or auxiliary verb of the sentence. In V1 and V2 sentences, the fields *finite* and *verbal complex* constitute what is often called the s*entence bracket:* The verbal complex contains a sequence of non-finite verbal material, used to assemble complex modality or tenses. In VF sentences, the finite verb and the verbal complex occur compactly towards the end of the field description. The elements that occur between *finite* and *verbal complex*, or before both in VF clauses, are called the *mittelfeld.* This field can hold an unrestricted number of arguments. All field descriptions also have a *nachfeld*, to the very end of the field description, which can hold extraposed, i.e. discontinuously realised material.

Other fields do not occur with all clause types. The *vorfeld* occurs in V2 clauses only and can contain arguments of the verb as well as topicalized discontinuous material. *c* is a field label used by Höhle (1986) in VF clauses to distinguish coordinators and subordinators. According to him, nominal material can also occur in this field. The field *Wackernagel* was mentioned before. It participates in a special topicalization construction, in which the topicalized phrase appears twice, as a full phrase, and pronominalised.

There is less research on the topology of the noun phrase, possibly because it is considered simpler. Eisenberg devotes a section to it, and postulates the fields in Table 3.2 (p. 400). He continues with investigating rules on adjective order, referring to semantic classes of adjectives.

An early, but very comprehensive account on German word order which is compatible with the topological field model is Bech (no year). More recent research will be classified according to the fields of the topological field model, which are under concern.

Reape (1994) gives a theoretical account of the interrelation between the verbal complex and mittelfeld order. He covers many phenomena which are known to be hard to model in phrase-structure accounts, as *scrambling, cross-*

serial dependencies and *VP intraposition* (p. 316, 331, 353).

Hinrichs and Nakazawa (1993) investigate the linearisation of elements in the verbal complex. They show that factors such as the type of the modal, the question whether a predicator is a raising verb, etc. influence the possible permutations of elements in the verbal complex. An example of such a permutation was given in 3 above.

Uszkoreit et al. investigate extraposition of clausal material to the nachfeld from the perspective of performance, concentrating on relative clauses. They show that length and extraposition distance are important factors when deciding whether to realise a relative clause in the mittelfeld, or to extrapose it to the nachfeld. Gamon et al. (2002) build on their work.

Kurz (2000) presents an empirical corpus investigation on word order in the mittelfeld. Some of her findings may be generalisable to the vorfeld. She reviews a range of accounts which have been applied to this topic. Examples (8) to (14) above suggested that there may be fine-grained rules governing preferences on mittelfeld order. Obviously, placing the reflexive accusative pronoun towards the end leads to a severe decrease in acceptability. Explanations from different linguistic levels have been provided in the literature:

Engel (1970) refers to linguistic form and syntactic function in order to predict mittelfeld order, and proposes this sequence of elements (adapted from Hoberg (1981, p. 42)).

NPron AReflPron APron DPron Ndef Nndef Ddef ADef Dndef Andef G

Here, N stands for nominative, A for accusative, D for dative, and G for genitive. Pron stands for pronominalised, and def and ndef refers to definite and non-definite NPs. Hoberg (1981) tests Engel's theory, actually finding the sequence above attested by the corpus. She identifies a problem concerning the order of datives and accusatives, though, and suggests animaticity as an alternative explanation.

Besides syntactic function, thematic roles have been suggested for predicting mittelfeld order. Scheepers (2000) shows in a psycholinguistic acceptability test and an eye tracking experiment that thematic roles do have an influence on acceptability and reading times, but only a mediating one, and that syntactic function is in fact the superior criterion.

Kurz showed that also the semantic class of the verb has influence on unmarked order in the mittelfeld. While there may be interactions with thematic role accounts, this is an indication that at least some word order rules may be best modelled by lexicalization.

In particular work from the Prague School of Linguistics (Sgall et al. (1986)) has used topic-focus structure, or contextual boundness as an explanation of word order. Kruijff (2001) also stresses the importance of topic-focus articulation to word order. A reflection of topic-focus-articulation is in fact implicit in Engels criterion of definiteness.

Finally, phrase length was suggested, and concentrated on by Hawkins (1984), who predicts that shorter elements precede longer ones, and that long elements tend to be extraposed over short distances.

While the topological field model is generally accepted as a descriptive device of German word order, it has several problems. Up to now, no formal framework has been presented which explicates the constraints on which elements can occur in which field, neither has a formal characterisation of field cardinality been given. Furthermore, the topological field model has hardly been applied to more complex sentences, e.g. to cases of coordination and ellipsis.

A number of investigations on more fine-grained rules governing word order in each of the fields of the topological model have been reviewed. While many alternative explanations from different linguistic levels have been suggested, none of the explanations have proven to be uniquely superior.

3.3 Approaches to Word Order (TDG)

This section reviews some formal grammar approaches to word order, which are in principle able to account for the data of the previous section. The list of frameworks could be easily extended.

In phrase structure (PS) approaches, the linguistic notions of word order and valency (or subcategorisation) and adjunction are combined into the notion of a phrase. This makes a separate analysis of these phenomena difficult. Furthermore, a treatment of discontinuous phrases is impossible due to the non-tangling condition of PS analyses. These problems lead to corrections early. The Generalised phrase structure grammar (GPSG) of Uszkoreit (1987) postulates two different levels of description, an ID (immediate dominance) level, and an LP (linear precedence) level, separating the two notions clearly. Rules about linear precedence are formulated in a constraint-based fashion. A constraint violation does not immediately result in ungrammaticality, but rather the number of violated constraints results in a higher or lower degree of markedness or grammaticality. Uszkoreit suggests the following five word order preferences, and shows how they can be integrated into his grammar framework.

> +NOM<+DAT, +NOM<+AKK,+ DAT<+AKK, -FOKUS<+FOKUS,
> +PRONOMEN<-PRONOMEN

A similar adaption has been added to the closely related HPSG framework. With the inclusion of the DEPS, additional to the SUBCAT list, a better separation between word order issues and valency issues has been reached. Reape (1994) deals with word order by operations of domain union and shuffling. A related approach is Kathol (2000). While unmodified categorial grammar (CG) was proven to be equivalent to CFG, shuffling operations have also been defined for categorial grammar, which account for word order freedom. Kruijff (2001) also employs a version of categorial grammar. Also in LFG, an account of word order have been presented Broeker (1998), which is compatible with topological field models, and can account for discontinuous realisations.

Dependency grammar Tesnière (1965), Melčuk (1988), Sgall et al. (1986) has never been concerned with word order. In this thesis, however, a recently proposed version of dependency grammar will be assumed, which remedies this problem in an elegant way. Debusmann (2001), Duchier and Debusmann (2001) present a multi-dimensional dependency formalism, called Topological Dependency Grammar (TDG). TDG offers an account of word order, which assumes a generalised version of the topological fields theory. Since concepts from TDG will be taken over for the purpose of this thesis, a short review of this formalism will be provided. TDG uses two tree structures to describe a sentence, an ID tree of syntactic functor-argument structure, and an LP tree describing word

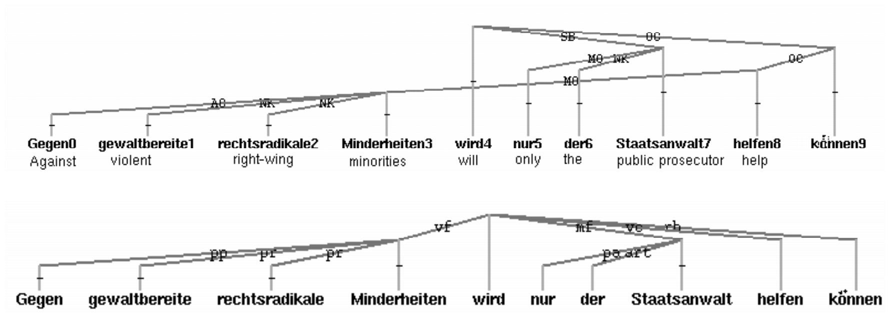


Figure 3.2: (above) TDG ID tree for Tiger sentence 133. (below) Corresponding LP tree.

order. Both define a one-to-one mapping between word forms of the input sentence and nodes in the trees, which gives the trees the appearance of dependency trees.[4]

TDG's ID trees are *unordered,* which means that only the *parent*-relation is defined on nodes, as usual for trees (Debusmann (2001, p. 24)), but no left-to-right order on nodes. ID trees will be depicted in a left-to-right fashion anyway, but it is important to note that this is just a way to depict these trees. ID tree edges are labelled with *syntactic roles.*

The nodes of LP trees are *ordered*, and the order on the nodes corresponds to the order of the words on the surface (Debusmann (2001, p. 54ff)). In order for an LP tree to be *well-formed*, it needs to be *projective*, i.e. for every node in the tree, *its yield must cover a contiguous substring* of the words of the sentence. The yield of a node is defined as the set of nodes under it, including itself. When depicting ID and LP trees, edges violating projectivity appear as crossing edges.[5] Edges of LP trees are labelled with *topological field labels.* There is a further well-formedness constraint on LP trees: For every node in the tree, the labels of its outgoing edges must conform to a globally defined *total order on field labels.*

Figure 3.2 shows and ID tree, and a corresponding LP tree for a sentence converted from the Tiger corpus.[6] Up to now, TDG has not come up with a linguistically motivated set of syntactic functions or field labels, but leaves the choice of edge labels to the grammar writer. In the figure, the Negra syntactic functions were taken as ID edge labels, and a set of field labels for the LP tree was loosely adapted from Debusmann (2001) for the purpose of illustration. By convention, ID analyses of German have assumed two dependents for finite auxiliary or modal verbs: the subject, and the non-finite full verb form, with further arguments and adjuncts attached as dependents of the non-finite verb.

[4]In contrast to the reference quoted, I will not use the term "dependency tree" for LP trees, but for ID trees only, and distinguish "dependency trees" from "word order trees" or "topological trees". Especially in the implementational chapters, I will also use the terms "ID-tree" and "LP-tree", without implying that the kind of tree structures I use internally are identical to TDG's ID and LP trees in all respects. For definitions of some of the mathematical terms used in the following, confer to Appendix B.

[5]A formal definition of a crossing edge will be given in Section 4.4.

[6]Section 4.3 gives details on converting corpus sentences to ID trees.

```
defuses { id lp }
deforder {
  po pp art pr nnh vf vafinh mf vc rb
}
defword t_nn {
  edgeID { sb mo }
  edgeLP { vf mf }
  nodeLP { nnh }
  valencyID { ac mo* }
  valencyLP { po? pp? art? pr* }
  blocks { ac mo }
}
defword t_vafin {
  edgeID { }
  edgeLP { }
  nodeLP { vvfinh }
  valencyID { sb mo* }
  valencyLP { vf mf* vc }
  blocks { }
}
defword 'Minderheiten' t_nn { }
defword 'wird' t_vafin { }
```

Figure 3.3: TDG declarative grammar formalism fragment

This analysis results in a non-projective ID edge from *helfen* to *Minderheiten* in the example sentence.

ID and LP trees are related by a set of ID/LP principles (Debusmann (2001, p. 63)). The most important principle for our purposes is the *climbing principle*, which states that if node w is the direct governor of v on the ID level, and $w_1, ..., w_n$ are its indirect governors, the direct governor of v on the LP level must be one of $w, w_1, ..., w_n$.

In Figure 3.2, the edge between *Minderheiten* and *helfen* cannot be a valid LP edge, since it violates projectivity. Consequently, there will be an edge between *Minderheiten* and *wird*, but not between *Minderheiten* and *helfen* in the corresponding LP tree, conforming to the principles of projectivity and climbing.

I complete the review of TDG with TDG's *lexicalized principles.* An ID/LP analysis is well-formed only if each lexical entry *offers* the edge labels of its children, and *accepts* the edge label its governor offers. This needs to be true of the ID and LP level. In the example, the word *Minderheiten* can only receive the LP-label *vf* (Vorfeld) because its governor's lexicon entry *wird* offers such a label, and the lexical entry of *Minderheiten* accepts it. On the ID level, *Minderheiten* can only be a child of *helfen*, because *helfen* offers a *mo* (modifier) label.

Figure 3.3 sums up the discussion of TDG with a grammar fragment in the TDG declarative grammar formalism tailored to explain selected aspects of the example sentence in Figure 3.2. The *deforder* statement defines the total order on field labels. *vafinh* and *nnh* are *internal* field labels, which specify the position of the head among its dependents, while the others are *external* field labels, which specify the position of sisters among each other. *t_nn* and *t_vafin* are *lexical types*, from which the lexical entries (Minderheiten, wird, etc.) inherit. The type for nouns specifies that nouns can occur as subject or modifiers of their governor, and be realised in vorfeld or middlefield position. It also specifies valency on the ID and LP level, with the aid of optionality flags. The *blocks* feature of nouns prohibits *ac* and *mo* dependents to "climb through". *valency* and *edge* features implement offering and accepting of edges.

TDG-like tree structures were chosen as a basis for the learning architecture to be developed for several reasons. The formalism clearly separates between a level of valency and adjunction on the one hand, and word order on the other hand. This leads to an expressive, but yet simple view on learning word order rules: This task can be rendered as learning rules that map from ID trees to LP trees. TDG's climbing principle offers a possibility to deal with discontinuity, which is not available in this simplicity in other formalisms. Furthermore, TDG is not committed to a specific set of topological field labels, or syntactic functions, but is flexible enough to be applied to corpus data, as it is annotated. Since TDG has not yet been applied to corpus data, it is a theoretically interesting question whether the formalism is expressive enough to elegantly express the phenomena found in the data. While TDG is a lexicalized formalism, a mechanism of lexical inheritance provides the possibility to express rules at an arbitrary level of generality.

Clearly, TDG cannot handle statistical preferences, which seem to be part of the nature of some word order phenomena described earlier. With the exception of Uszkoreit (1987), none of the particular approaches discussed above has provided a model of statistical preferences or graded grammaticality. While such approaches are being developed, they are out of scope for the purpose of this thesis.

Chapter 4

A Word Order Learning System

This chapter presents the main contribution of the thesis. Here, I develop an approach to word order learning that fits in well to a wider grammar learning scenario. I state the goals in Section 4.1. In Section 4.2, I sub-classify the problem into corpus conversion, climbing, primitive extraction and field induction, robustness and feature selection, and export, and develop an architecture that follows this classification. In the following sections, I develop solutions for each of the subproblems in this order. The result is a machine learning architecture whose implementation will be described in the following chapter.

4.1 Goals of this approach

The goal of this thesis is to develop, implement and evaluate a new machine learning architecture to learning of word order rules from treebanks. The approach can be characterised as

1. *unsupervised:* Although the input is a richly annotated syntactic tree, which encodes word order, the rules that are to be learned differ from the input structures: The learning algorithm can be framed as clustering nodes of the syntax trees into field indices, which are not encoded in the input data. This contrasts to Becker and Frank (2002), who assume a hand-made training corpus annotated with the structures to be learned.

2. *symbolic:* The learned rules are linguistically interpretable descriptions, rather than probability distributions over a given class of linguistic tags.

3. *robust:* The approach should be able to deal with noise in the input data, which statistical tendencies, and with different types of training data with regard to annotation scheme and size.

Going into more detail, desired characteristics of the learning system are the following.

1. *clearly modularised:* The approach fits into a larger grammar learning scenario of which word-order learning is a sub-module. An entire learning

architecture would comprise of, beside learning of word order, learning of subcategorization frames for lexicon entries, learning of subcategorization alternation and adjunction rules, and learning of agreement. Only through modularisation is it possibly to explicitly exclude the aforementioned phenomena from the focus of attention, and design a system devoted exclusively to word order. However, the architecture should not only be clearly modularised externally, but also internally, i.e. it should have clearly identifiable subcomponents. Immediate advantages of this clear modularisation are good human-readability of the produced rules, easy maintenance, and good integratability with other work that focuses on other aspects of syntactic learning, e.g. subcategorization frame extraction Sarkar and Zeman (2000). This contrasts e.g. with PCFG approaches, which do not distinguish between word order, subcategorization and adjunction rules Charniak (1996), and is a step towards "linguistically abstract" grammars in the sense of Kinyon and Prolo (2002).

2. *formalism independent*: Due to the problems of inter-framework conversion (Section 2.2), the produced rules should be general enough to be exportable to an arbitrary grammar formalism. As an example of this strategy, an experiment will be presented in Section 6.4, where the produced rules were exported to the TDG formalism, and used for parsing. Formalism-independence is a characteristic that distinguishes this approach from Xia et al. (2001), just to name one.

3. *few linguistic assumptions:* While the term topological field descriptions will be used in the following, only a generalised, and very abstract notion of topological fields will be assumed (see e.g. Kruijff (2001)). The result is a learning system that can actually discover the regularities implicit in the data, and can even be used to evaluate the predictions of descriptive linguistic theories. A particular phenomenon discussed in Section 3.2, which should be modelled, is discontinuity. The consequence is applicability to a wide range of typologically different languages. The learned rules will be compared to linguistic theories in Section 6.5. This contrasts e.g. with Villavicencio (2000), who assume a Principles and Parameters theory, along with X-bar assumptions and a given hierarchy of phrase classes, and learn parameter values only.

4. *linguistically fine-grained*: The number of linguistic features annotated in treebanks is steadily increasing (Abeille et al. (2003)), and word order regularities can potentially depend on many linguistic features, as became evident in Chapter 3. The architecture should therefore be flexible enough to capture regularities dependent on any linguistic feature the corpus is annotated for. This also means that the architecture should be independent of the specific annotation scheme of the corpus used.

4.2 Overview of the learning system

This section develops a machine learning architecture following the desiderata from the previous section, and gives an overview of this architecture, before each

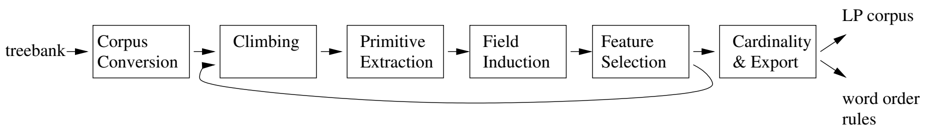


Figure 4.1: System Architecture

of the components is described in more detail in the following sections. Figure 4.1 is an overview of the learning architecture, indicating input and output.

For reasons outlined in Section 3.3, dependency structures were chosen as an internal representation of syntactic structure. In order to be independent of the concrete corpus used for training, there is a Corpus Conversion module which converts the corpus to dependency structures. It will be described together with a description of the input corpus format in Section 4.3.

The choice of input representation was characterised as essential for a machine learning architecture in Section 2.1. The system component that perceives of the input data as a set of primitives to learn from is called Primitive Extraction and will be outlined in Section 4.5. While an entire syntactic tree is certainly too large a primitive to learn from, an example set consisting of single words is obviously too little information to learn rules about word order. It will be argued that *a pair of two words occurring under a common head* in a syntactic tree, where each of the words, and the head is represented by a linguistic feature structure, is a suitable primitive. Such a triple of feature structures will be called a *precedence pair.*

Discontinuity was argued to be a potentially widely occurring phenomenon of the input data in Section 3.1. In Section 4.4, a Climbing module will be described that deals with discontinuous configurations. It will be argued that discontinuous realisations can be identified in the input trees, and resolved by making discontinuously realised nodes *climb* to their parent nodes. As can be seen in Figure 4.1, the climbing module is prior to the primitive extraction module, with the primitive extraction module working on the modified trees *(climb trees).*

The Field Induction component, outlined in Section 4.6, calculates word order rules from the precedence pairs extracted during primitive extraction. I am following a *generalised topological fields theory* which meets the desideratum of few linguistic assumptions of the previous section. During field induction, a *graph* is constructed from the precedence pairs in the data, whose nodes are descriptions of dependent elements, and whose edges are the *indirectly-precedes*-relation. Such a graph is learned for each class of head item found in the corpus. The section will illustrate that it is possible to construct such a set of graphs from the data, and that these graphs can be interpreted as generalised topological field descriptions.

Feature Selection (Section 2.4) aims at making the approach robust. Two devices are identified for this task: a parametrisation of the *edge selection function,* which determines which precedence pairs to judge as reliable to be included in the graph, and the *feature selection function* mapping from words to linguistic feature structures when constructing precedence pairs. It depends heavily on feature selection whether the goal of learning linguistically fine-grained rules can be achieved: If feature selection is too coarse grained, only very general word order regularities will be discovered. If feature selection is too fine grained, the data will be overfit. Two strategies of feature selection will be proposed, *manual feature selection*, where the feature selection function is given, and *automatic feature selection,* where *decision tree learning* is used to adapt the feature selection function iteratively, and converge towards a near-optimal function.

FIELD CARDNIALITY AND EXPORT (Section 4.8) deals with (a) augmenting the learned word order rules with information on field cardinality and (b) exporting the learned rules to different formats. This module produces a corpus of *topological trees* conforming to the learned rules, using a test corpus, which is disjoint from the training corpus. It exports the learned field descriptions to a human readable form, and to a concrete grammar formalism. As an example, export to the TDG declarative formalism is demonstrated in Section 4.8. The output of the EXPORT module will be the basis for evaluation of the system in Chapter 6.

4.3 Corpus Conversion

With the goal in mind of being able to handle discontinuity, it was decided to work internally on dependency trees solely, which may have crossing edges. This section describes the interface to the input corpus, which cannot be expected to be available in this form. In the implementation, care was taken to make the corpus conversion easily exchangeable.

Two corpora of German, the Negra corpus and the Tiger corpus, were available in Negra annotation format (Skut et al. (1998), Brants et al. (2002))[1]. Other corpora were also available in this format. This annotation scheme has the advantage of allowing for crossing edges, and features syntactic function annotation. It therefore resembles TDG's ID structures closely. Figure 4.2 shows an examplary sentence in Negra annotation format.

Word forms are labelled with POS-tags according to the Stuttgart-Tübingen tagset (Schiller et al.). There is also a small set of phrase labels, and a larger set of functional labels, indicated at the edges. Appendix A contains tables of all tags in the Negra/Tiger corpora.

In order not to feed too much linguistic knowledge into the learning architecture, it was decided to leave all structural decisions made in the input corpus unchanged, as well as the corpus tagset. A mapping from Negra mobiles to dependency can then be defined as an algorithm that recursively lifts the terminal head of each non-terminal node in the Negra mobile, and adds non-head nodes as dependents. This algorithm is outlined in Algorithm 1.[2]

I define the *projection line* in step 2 as all nodes in the Negra syntax tree, which are on a path downwards in the tree, following *head labels* only. By definition, a node is a member of its own projection line. The *lexical head* of a node is the last element of the projection line of a node, and is always a terminal node. Since the linguistic core of the algorithm is the determination of what constitutes a *head label*, I discuss this question, before I illustrate the algorithm by means of example.

[1]The Tiger annotation scheme is an extension of the Negra scheme
[2]Punctuation is removed from the trees during conversion.

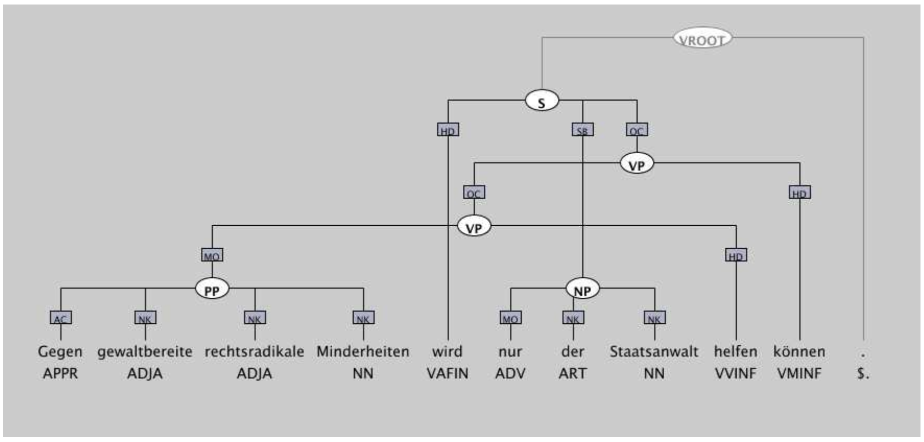


Figure 4.2: Corpus sentence in Negra annotation scheme (Tiger sentence 133)

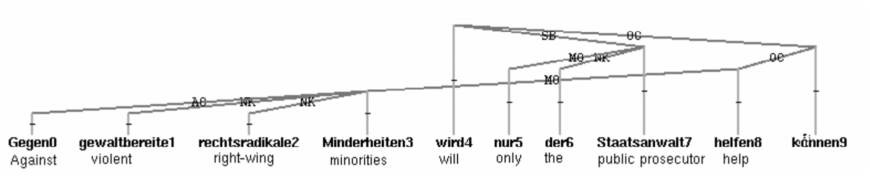


Figure 4.3: Dependency tree corresponding to Figure 4.2

Algorithm 1 DependencyTree convert(NegraTree n)

```
1. construct a dependency node $n'$ from the *lexical head* of $n$
2. for all nodes $m$ on the *projection line* of $n$
    (a) for all nodes $k$ which are not on the *projection line* of $m$
        i. $k' = convert(k)$
        ii. add $k'$ as a direct dependent of $n'$
3. return $n'$
```

In the case of S and VP nodes, a designated functional label HD indicates the head. Other types of phrases do not have an HD label, but a similar designated element. These are discourse level constituents, and placeholder phrases. Noun phrases, however, are flatly annotated in Negra, with all central elements bearing the label NK (noun kernel). The right-most NK element may be taken as a head in this type of constructions. The same is true of multi-component proper names, multi-component numerical values and adverbial phrases ("und zwar, immer wieder, ..."). This strategy may lead to arbitrary decisions in some cases. However, the very reason why these phrases were chosen to be flatly annotated in Negra is that the decision of what is the head is controversial in these cases. If no head is found according to the outlined strategy, the left-most element is returned as the head by default.

In coordinated structures, the coordination (edge label CD, POS tag KON) is taken as the head, if the conjoints are phrasal (CJ). If there is no coordinating element, or only terminal coordinated elements, the default strategy of picking the left-most dependent phrase applies. The result is a picture where the right conjoint is a dependent of the left conjoint, similar to the analysis of Melčuk (1988). This happens mostly in simple coordinated noun phrases. Also in case of ellipsis, the default strategy applies and selects the left-most element as the head.

The conversion from Negra tree structures to dependency structures can be illustrated with the aid of Figures 4.2 and 4.3. The lexical head of the top level S node is "wird", since it is labelled with the HD edge label. This node is converted into a dependency node. There is only one node on the projection line of the S node, which is the S node itself. There are two subnodes which are not on the projection line, the NP and the upper VP. On both of them, the algorithm is recursively called in step 2(a)i of Algorithm 1. Thus "Staatsanwalt" and "können" are inserted as direct dependents of "wird" in the dependency tree (step 2(a)ii), and further recursive calls are made.

Since dependency trees of the kind of Figure 4.3 are the input of the learning architecture, it is worth considering the information such a tree encodes. A tree consists of

1. *nodes*: each node is associated with a word form, and linguistic features of this word form, specifically with its part-of-speech tag. The edge label of the incoming (upward-pointing) edge of each node can also be considered a feature of the node.

2. *edges:* an edge is a relation between two nodes, a governor and a dependent

3. *order:* the tree also encodes an order on words (nodes) on the surface

4.4 Climbing

This section makes a central independence assumption on the nodes of dependency trees, on the basis of which the primitives for the learning system will be formulated in the following section. The assumption leads to a further dependency tree conversion algorithm, which is executed prior to the extraction of primitives.

It was stated in Section 2.1 that an essential step in designing a machine learning system is the representation of the input data. Considering entire trees as input data, though, is inadequate for symbolic learning[3]. Therefore, an independence assumption on nodes in the trees with regard to word order is needed, which allows to decompose trees into smaller subtrees. These subtrees will be regarded as primitive, and independent of each other, and will form the basis of the definition of a primitive example for the learning architecture in the next section. Consider the following definitions.

Word Order Domain Let W be the set of node tokens in the corpus. The *word order domain* (or *domain*) of a node, $D(w)$, is a *sequence*[4] of nodes, which at least includes the node itself. $D(w) \subseteq W$; $w \in W$; $w \in D(w)$. Call w the *head of the domain* $D(w)$.

Independence Assumption If word forms a and b occur in a common domain, $\exists h \in W : a \in D(h)$, $b \in D(h)$, their order depends on features of a, b and the head of their common domain, h only. If a and b do not occur in a common domain, there is no rule about their respective order.

The formalisation of word order domains builds on the insights of Kathol (2000) and Duchier and Debusmann (2001), although the linguistic concept is not new. A trivial definition of a word order domain could be based on ID trees. Assume the word order domain of a node w as the set containing w's dependents in the ID tree, and w itself. In the example of Figure 4.3, this would mean that the order domain of "Minderheiten" is the sequence <"gegen", "gewaltbereite", "rechtsradikale", "Minderheiten">. By the independence assumption from above, this amounts to saying that there are grammaticalised order rules on the order of "gegen" and "gewaltbereite", or "gegen" and "Minderheiten", but that a rule on the respective order of "gegen" and "der" is linguistically unmotivated.

With defining a word order domain on the basis of ID trees, it is not possible to account for discontinuous phenomena. In the example, no rule could be formulated that predicts that the subtree "Minderheiten" and the subtree "Staatsanwalt" can commute. The rest of this section will follow TDG in defining word order domains not on ID trees, but on converted tree structures, which I will call *climb trees*. Climb trees are mathematically equivalent to TDG's LP trees except for their edge labels. The reader is referred to Debusmann (2001) for a more detailed survey.

Climb Tree The climb tree $T_C =< N, V_C, r >$ of an ID tree $T_{ID} =< N, V_{ID}, r >$ ($r \in N$, $V \subseteq N \times N$) has the same nodes, and the same root as its ID tree. For every node n, the parent of n in the climb tree is one of n's *transitive parents* in the ID tree. Every edge in a climb tree must be *projective.*

Crossing Edge An edge from a governor m to a dependent n is a *crossing, or non-projective edge* if there is at least one word form w, appearing on the surface between m an n, which is not an indirect dependent of m.

[3]Assuming entire trees as examples is in fact a possible assumption, and would lead to a memory based learning scenario (see Section 2.1).

[4]A sequence can be defined as an ordered set. A sequence will be notated in angle brackets if the order on the set matters, or in braces if only the underlying set is of importance.

Algorithm 2 Convert an ID tree T_{ID} to a climb tree T_C

```
1. for every node $n$ in $T_{ID}$ in top-down order
    (a) let $C$ be the list of transitive parents of $n$, from $parent_{ID}(n)$ to the root
    (b) iterate $c \in C$
        i. if the edge between $c$ and $n$ is projective, establish $parent_C(n) = c$, and break
```

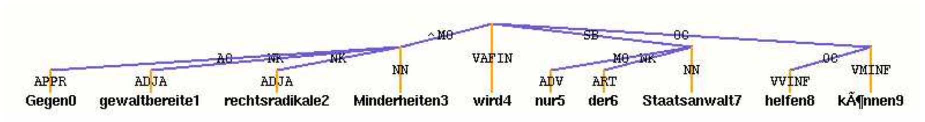


Figure 4.4: Climb Tree for the example sentence

Algorithm 2 is the easiest algorithm which can be designed to calculate a climb tree from an ID tree. The algorithm considers all transitive parents of a node in the ID tree, and establishes the nearest of these transitive parents as the parent of the node in the output climb tree, for which the edge is non-crossing. Correctness and termination of this algorithm can be proven by showing that a node can at least climb to the domain of the root, and that it necessarily stops climbing there, because an edge between the root and an arbitrary node is projective by definition.

If word order domains are defined on the basis of climb trees instead of dependency trees, a representation of the data has been found that can handle discontinuity. In Figure 4.3, the word order domain of "Staatsanwalt" is unchanged, but the word order domain of "wird" is now <"Minderheiten", "wird", "Staatsanwalt", "können">, because the edge between "Minderheiten" and "wird" has to climb over a distance of 2 nodes in order to be non-crossing. Figure 4.4 shows a climb tree computed from the earlier example sentence.

In the following sections, primitives for learning will be defined on the basis of climb trees. Nodes that have climbed during conversion from ID trees to climb trees will be called *non-local elements*, and will be distinguished from *local elements.* During induction of field descriptions, non-local and local elements will be treated the same.

Nothing was said about the edge labels of climb trees yet. The edges of input dependency trees reflect a relation between two nodes. When changing edges on the way from dependency trees to climb trees, the edge label should be adapted. In depicting climb trees, I will mark the climbed edge with a ^ sign to indicate that the edge label reflects the relation to the original governor. It is more accurate, however, to include the exact nature of the climbed edge into the representation of the tree. This is accomplished by the following definition of a climbing path. I will not go into further details of how to notate a climbing path. The concept resembles the notion of a path in LFG Bresnan (2001), and notation is available there.

Climbing Path and Climbing Distance The *climbing path* of a node is the sequence of nodes c iterated during execution of Algorithm 2. The *climbing distance* is the number of nodes c iterated.

I conclude with a judgement of the linguistic validity of the independence assumption made. Certainly, the assumption follows linguistic practice, see e.g. Collins (1999a). There are, however, some cases where it is not clear whether the assumption is correct. One case in point are the conditions that trigger verb-secondness vs. verb-final-placement in German subordinate clauses. Here, the occurrence of a subordinating conjunction triggers the position of the verb.[5]

Algorithm 2 only assumes edges to climb in the face of discontinuity. In other words, it is biased towards assuming non-climbed edges, where they are possible. Consider, however, the following, modified example sentence.

1. Nur der Staatsanwalt wird gegen gewaltbereite rechtsradikale Minderheiten helfen können.

Example 1 is a continuous realisation of the previous example. Consequently, the climb tree according to Algorithm 2 will not differ from the ID tree. It may be desirable in some situations not to choose the nearest possible parent in step 1b(i). TDG, in any case, assumes that climbing also takes place in sentences like 1 above, speaking of "forced climbing". The development of such an algorithm, however, is left for future research.

[5]A technique that is employed in many syntactic frameworks to account for these cases (e.g. GPSG and HPSG) is to include special features, inherited from below (foot features) into the descriptions of nodes.

4.5 Primitive Extraction

This section deals with the question of how to represent the input climb trees as a set of primitive examples, from which word order rules can be learned.

Section 4.3 described the input data as trees, consisting of nodes with associated *features*, edges as *relations* between nodes, and an *order* on nodes. The previous section made an independence assumption on nodes, which allowed a reduction in the number of relations in the trees and the consideration of local subtrees only. This leaves two open questions:

1. Which linguistic features should be included into the example descriptions?

2. How should the notion of *order* be formalized?

The first question will be postponed until Section 4.7. It will be mathematically captured, though, in this section by the definition of a *feature selection function.* This question is relevant, because the set of possibly word order relevant linguistic features is large, and possibly drawn from different linguistic levels (see Section 3.2). For this reason, I decide to represent word forms as linguistic feature structures (attribute-value-matrices, AVMs, see Appendix B). An AVM is a function from features to values, e.g. POS-tag, syntactic relation, base form, etc. A possible feature structure for the word "gegen" is [rel:x, pos:y, base:gegen]. The approach parametrizes the features to pick, and even offers the possibility to learn a set of order relevant features.

There are two possible answers to the second question from a mathematical point of view. Order is formalized mathematically as a binary relation on a set. In the previous section, it was assumed that a rule about the order of two word forms can only be formulated if they are in the same domain. This leaves room for either expressing order in terms of *transitive precedence*, or *immediate precedence* on the syntactic surface. In the example sentence from above, "gegen" immediately precedes "gewaltbereite", and transitively precedes any other word in the sentence. It neither precedes nor succeeds itself. I opt for transitive precedence, making another linguistic assumption which was motivated in Sections 2.2 and 3.2.

Assumption Word order can be described independently of valency.

If this assumption is correct, it will be advantageous to pick transitive precedence as a formalization of word order, because dependents of a certain type might or might not occur in the data, due to valency and adjunction phenomena. Immediate precedence will fail in the face of this characteristic of the data, while transitive precedence will behave robustly. The reader is asked to verify this claim.

I now formalize the ideas from above, starting with the definition of a *feature selection function*, and continuing with the definition of a primitive for learning.

Head Feature Selection Function Let W be the set of node tokens of the corpus. Let E be a set of *(dependent) elements,* and H be a set of *head elements* (or *head classes*); all three sets are disjoint. Let $\tilde{f} : W \to H$ be a function from word form nodes to the set of head classes, the *head feature selection function.*

Dependent Feature Selection Function Let $f : W \to E$ be a function from word form nodes to the set of elements, the *dependent feature selection function.*

With the definition of the feature selection functions, the problem of which linguistic features to pick as word order relevant has been mathematically expressed. Following the discussion above, I assume W to be a set of feature structures, and write $feature(w)$, $w \in W$ to refer to a feature of a feature structure. In this section, E and H are assumed to be arbitrary, descriptive symbols, although they will eventually be sets of feature structures too. $\tilde{f}$ is assumed as given throughout this thesis, although there will be some remarks on learning it. As long as also f is assumed as known a priori, I will speak of *manual feature selection.* If f is to be learned, I will speak of *automatic feature selection.*

Below is a possible head feature selection function:

$$\tilde{f} = pos(w)$$

The dependent feature selection function should at least be able to distinguish between a non-head and a head element. A dependent feature selection function with this characteristic is the following.

$$f_h(w) = \begin{cases} \mathsf{HD} & \text{if } h = w \\ rel(w) & \text{otherwise} \end{cases}$$

In the running example, the node "Staatsanwalt" is reduced to NN by the head feature selection function. As regards the dependent feature selection function, it depends on the "point of view" from where the node is reduced: $f_{\text{"Staatsanwalt"}}(\text{"Staatsanwalt"}) = \mathsf{HD}$, but $f_{\text{"wird"}}(\text{"Staatsanwalt"}) = \mathsf{SB}$. A more sophisticated dependent feature selection function tailored to the annotation format of the input corpus is the following. No further comments will be made at this point on an appropriate choice of a function though.

$$f_h(w) = \begin{cases} \mathsf{HD} & \text{if } h = w \\ pos(w) & \text{if } \tilde{f}(h) = \mathsf{NN} \\ rel(w) & \text{otherwise} \end{cases}$$

I now define the primitive example of the learning architecture, making use of the feature selection function, the definition of order and the assumption about how much local tree context should be taken into account. Such a primitive example will be called a *precedence pair*, and defined as an occurrence of two *dependent elements* under a common *head element*, either indirectly preceding or succeeding each other.

Precedence Pair Instance A *precedence pair instance* is a triple $\langle w_1, w_2, w_h \rangle$ of word forms from the domain of w_h, the *head* ($w_1, w_2, w_h \in D(w_h)$), such that $w_1 \prec w_2$. $\prec$ is the *transitively-precedes-relation.*

Precedence Pair A *precedence pair* is a triple $\langle e_1, e_2, h \rangle$, $e_1, e_2 \in E$; $h \in H$ such that there is a precedence pair instance $\langle w_1, w_2, w_h \rangle$ with $f_h(w_1) = e_1$; $f_h(w_2) = e_2$; $f(w_h) = h$. Let the set of precedence pairs be called P.

wird	$\langle MO, HD, VAFIN\rangle$, $\langle MO, SB, VAFIN\rangle$, $\langle MO, OC, VAFIN\rangle$,
	$\langle HD, SB, VAFIN\rangle$, $\langle HD, OC, VAFIN\rangle$, $\langle SB, OC, VAFIN\rangle$,
Minderheiten	$\langle AC, NK, NN\rangle$,$\langle AC, NK, NN\rangle$,$\langle AC, HD, NN\rangle$,
	$\langle NK, NK, NN\rangle$,$\langle NK, HD, NN\rangle$,$\langle NK, HD, NN\rangle$,
Staatsanwalt	$\langle MO, NK, NN\rangle$,$\langle MO, HD, NN\rangle$,$\langle NK, HD, NN\rangle$,
können	$\langle OC, HD, VMINF\rangle$

Figure 4.5: Precedence Pair Instances constructed top-down from the example sentence, with the first dependent feature selection function

The third member of the tuple will sometimes be skipped, if it is clear from the context. Examples of precedence pair instances in Figure 4.4 are ⟨"Staatsanwalt", "können", "wird"⟩ and ⟨"wird", "Staatsanwalt", "wird"⟩. The corresponding precedence pairs, constructed with the underlying feature selection functions from above, are $\langle \mathsf{SB}, \mathsf{OC}, \mathsf{VAFIN}\rangle$ and $\langle \mathsf{HD}, \mathsf{SB}, \mathsf{VAFIN}\rangle$. Note that the second of the precedence pair instances contains the same node twice, and that the corresponding precedence pair describes the position of the head within its own domain. If the head is clear from the context, we may abbreviate the precedence pairs to $\langle \mathsf{SB}, \mathsf{OC}\rangle$ and $\langle \mathsf{HD}, \mathsf{SB}\rangle$.

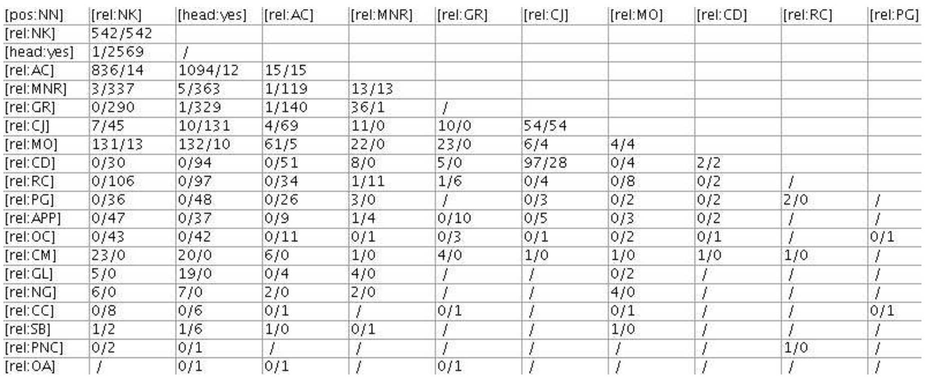
Algorithm 3 presents an algorithm how to construct the set P for a given input sentence. Figure 4.5 lists all precedence pairs constructed according to this algorithm from the example sentence.

Algorithm 3 constructing precedence pairs P from an input sentence S

```
1. INPUT $P$: set of precedence pairs
2. for all nodes $n$ in the tree $S$
    (a) let $D$ be the domain of $n$
    (b) for all $m \times m' \in D$ where $m$ indirectly precedes $m'$
        i. add a precedence pair $\langle f(m), f(m'), n\rangle$ to $P$
```

I conclude this section with an example of precedence pairs calculated from real data. Figure 4.6 shows the precedence pairs of the head class [pos:NN], with a feature selection function that relies on syntactic relation only, calculated from the initial 1000 sentences of the Tiger corpus. The precedence pairs are arranged in a *precedence table,* a convenient way to represent a large number of precedence pairs. The head element is indicated in the left upper corner of the table, and column and row captions indicate dependent elements. The value before the slash indicates how often the precedence pair <row,column> was seen in the data, and the value after the slash indicates how often <column,row> was seen. To capture this notion mathematically, I define the *frequency* of a precedence pair at this point.

Frequency of a Precedence Pair the *frequency*, or *count*, of a precedence pair $c(p)$; $p \in P$ is the number of precedence pair instances of a precedence pair.



[pos:NN]	[rel:NK]	[head:yes]	[rel:AC]	[rel:MNR]	[rel:GR]	[rel:CJ]	[rel:MO]	[rel:CD]	[rel:RC]	[rel:PG]
[rel:NK]	542/542									
[head:yes]	1/2569	/								
[rel:AC]	836/14	1094/12	15/15							
[rel:MNR]	3/337	5/363	1/119	13/13						
[rel:GR]	0/290	1/329	1/140	36/1	/					
[rel:CJ]	7/45	10/131	4/69	11/0	10/0	54/54				
[rel:MO]	131/13	132/10	61/5	22/0	23/0	6/4	4/4			
[rel:CD]	0/30	0/94	0/51	8/0	5/0	97/28	0/4	2/2		
[rel:RC]	0/106	0/97	0/34	1/11	1/6	0/4	0/8	0/2	/	
[rel:PG]	0/36	0/48	0/26	3/0	/	0/3	0/2	0/2	2/0	/
[rel:APP]	0/47	0/37	0/9	1/4	0/10	0/5	0/3	0/2	/	/
[rel:OC]	0/43	0/42	0/11	0/1	0/3	0/1	0/2	0/1	/	0/1
[rel:CM]	23/0	20/0	6/0	1/0	4/0	1/0	1/0	1/0	1/0	/
[rel:GL]	5/0	19/0	0/4	4/0	/	/	0/2	/	/	/
[rel:NG]	6/0	7/0	2/0	2/0	/	/	4/0	/	/	/
[rel:CC]	0/8	0/6	0/1	/	0/1	/	0/1	/	/	0/1
[rel:SB]	1/2	1/6	1/0	0/1	/	/	1/0	/	/	/
[rel:PNC]	0/2	0/1	/	/	/	/	/	/	1/0	/
[rel:OA]	/	0/1	0/1	/	0/1	/	/	/	/	/

Figure 4.6: Precedence Table for Normal Nouns

4.6 Field Induction

This section presents an approach to constructing a set of *field descriptions* from the the set of precedence pairs observed in the corpus. A field description should be interpretable as a rule that predicts the order of elements, when given an unordered domain of a node.

In the previous section, I made the assumption that the order of two word forms with respect to each other depends only on their features, and on features of their common head. If this assumption is correct, it should be possible to construct a field description for each class of head elements, i.e. for each $h \in H$. The data upon which to construct the field description is a subset of the set of precedence pairs, viz. exactly those precedence pairs that occurred under instances of the head class h, formally $P_h = \{\langle e_1, e_2, h' \rangle \in P \mid h = h'\}$. P_h will be called the *precedence pairs of h*. The set of elements in the precedence pairs of h, $E_h = \{e \in E \mid \exists \langle e_1, e_2, h \rangle \in P_h,\ e = e_1 \vee e = e_2\}$, will be called the *elements of h*.

I assume now that a *generalized topological fields model* is an appropriate form to describe the grammaticalised order of the elements of h. A topological field model does not have a means, though, to express preferences in the data, as opposed to hard constraints. The model should fulfill two criteria:

1. it should be robust in the face of annotation errors, or highly infrequent exceptions

2. while symbolic in nature, it should be augmentable such that it is able to express tendencies in the data

As pointed out in Section 3.3, page 24, a topological field description is a sequence of fields, each of which can contain a defined set of elements. Equivalently, a field description can be defined as follows:

Field Description The *field description of head h* is an assignment of elements to a set of consecutive, non-negative integers, the *field indices*. F_i refers to the set of elements assigned to field index i, and is called a *field*.

The question is the whether it is possible to learn such a field description from precedence pairs. Such a learning algorithm can be viewed as an instance of clustering (see Section 2.1), which clusters elements into field indices.

Assume a field description with three fields as linguistically desired:

$$F_1 = \{a, b, c\};\ F_2 = \{a, d, e\};\ F_3 = \{f, g, c\}$$

The formalization has the disadvantage of allowing for multiple occurrences of the same element in different fields. For example, c occurs in the first and the third field, a in the first and second. In the field descriptions of Section 3.3, this was a characteristic often assumed, e.g. subjects could occur in the vorfeld or mittelfeld. This characteristic, however, poses a problem towards learning field descriptions from precedence pairs: Assume the field description above was to be learned from precedence pairs. The algorithm might have seen the pairs $\langle a, e\rangle$ and $\langle e, a\rangle$. It could conclude from this fact that there is no order constraint for a and e, and include them into the same field. However, it might also have seen the pairs $\langle a, c\rangle$ and $\langle c, a\rangle$. How should it know that a and c are not to be included in the same field? Furthermore, it might have seen pairs $\langle a, d\rangle$ and $\langle d, a\rangle$, just as $\langle b, d\rangle$ and $\langle d, b\rangle$. How should it know that a can occur in field 1, while b cannot?

In the formalization that follows, I assume that there cannot be multiple occurrences of the same element in different fields. This assumption makes a field description of head h interpretable as a *partial order on the elements of h*. See Davey and Priestley (1994) for the connection between a partial order and assignment to integers.[6] I will show how it is still possible to come closer to representations like the one above towards the end of this section.

Field Element Order The *field element order of head h* is a partial order on the elements of h, $O_h = \langle E_h, \lhd\rangle$. $\lhd$ is an irreflexive, anti-symmetric, transitive relation on E_h.

It remains to be shown that the field element order can be calculated from a set of precedence pairs. A graph algorithm (see Appendix B) will be developed to accomplish this task. Assume a graph whose nodes correspond to the elements of h as follows.

Order Graph The order graph of h is a directed graph $G_h = \langle N, T\rangle$, $T \subseteq \langle N \times N\rangle$ with an isomorphism i between N and E_h.

Edge Selection Function Assume a function $s_h : E_h \times E_h \rightarrow \{0, 1\}$, the *edge selection function*. Let there be an edge $\langle n, m\rangle$ in G_h iff $s_h(i(n), i(m)) = 1$.

Obviously, the shape of the order graph depends heavily on the edge selection function, whose purpose it is to predict which edges to include into the order graph, on the basis of the available precedence pairs. The most straightforward edge selection function is given below, and will be referred to as *simple edge selection*.

[6]An assignment of elements to integers is not *equivalent* to a partial order, but *induces* such a partial order on the elements. It will become clear below that Algorithm 3 computes the partial order, while Algorithm 5 computes an assignment from elements to integers which conforms to this partial order. Since there is more than one assignment that conforms to the partial order, the algorithm includes a further assumption which will be commented on below.

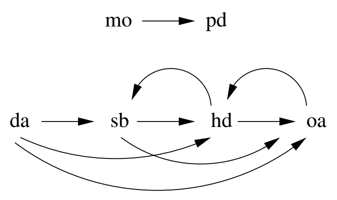


Figure 4.7: Example of an order graph

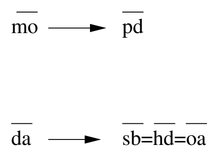


Figure 4.8: Order graph with equivalent nodes collapsed to single nodes

$$s_h(e_1, e_2) := \; 1 \text{ iff } \exists \langle e_1, e_2, h\rangle \in P_h$$

Simple edge selection is sufficient for the rest of this section. It will be refined in Section 4.7. Simple edge selection predicts an edge in the order graph whenever there is evidence in the precedence pairs for one element preceding the other. Figure 4.7 shows a simple example of an order graph that might be constructed with simple edge selection.

An order graph may contain cycles. Linguistically, a cycle in the order graph reflects the fact that there is no clear-cut constraint to the respective order of the elements on the cycle. While an acyclic graph defines a partial order on its nodes, a cycle that contains cycles does not. Nevertheless, the order graph can serve as a basis for the computation of the desired partial order on elements D_h. This idea is formalized in the following definition and two theorems, adapted from Mehlhorn (1999, Chapter 4, p. 25ff) (also see Davey and Priestley (1994)).

Theorem A directed acyclic graph induces a partial order on its nodes.

Strongly Connected Components A strongly connected component of a directed graph $G = \langle N, T\rangle$ is a maximal subgraph $G' = \langle N', T'\rangle$ of G in which every node is reachable from any other node ($n \to^* m \to^* n$ for $n, m \in N'$). Any two nodes that are in the same strongly connected components are called *equivalent.*

Theorem The strongly connected components of a directed graph can be computed in linear time.

Figure 4.8 shows a graph that differs from the graph in Figure 4.7 in this respect: all equivalent nodes in the previous graph are collapsed to a single node. All edges between non-equivalent nodes are maintained. Mehlhorn (1999) presents a linear-time algorithm for this task, However it is fairly complex.

Mehlhorn (1999) assumes a graph as given, and explicitly outputs a graph like the one in 4.8. I developed a simpler algorithm, presented in Algorithm 4, which (a) does not explicitly construct a graph like the one in Figure 4.8, since this is not necessary for our purposes, and (b) combines the construction of the graph from precedence pairs and pre-calculations into a single step.

A node is constructed for each element found in the set of precedence pairs (line 4). The graph is stored in *adjacency list representation* as a map from type *Element* to type *Node.* Data structure *Node* defines a set of elements, a field index, and lists of incoming and outgoing edges (line 2). Furthermore, a list *initial* (line 3) is maintained during construction which holds the set of nodes with indegree 0 (i.e. the set of nodes which are not preceded by any other node, line 4a to 4c).

I claim that after termination of algorithm 4, *nodes* holds a directed acyclic graph, and *initial* holds a set of nodes with indegree zero. A proof of these claims proceeds by showing the invariant of the algorithm that after each complete execution of the loop that begins in line 4, the graph is acyclic. To see this, note that an empty graph is acyclic. After each insertion of an edge, the graph is checked whether the new edge results in a cycle in line 4(d), and if so the cycle is removed on the fly in lines 4(d)i to 4(d)iii. This prooves termination of the algorithm, and acyclicity of the resulting graph.

It remains to be shown that lines 4(d)i to 4(d)iii in fact remove the resulting cycle. In step 4(d), the graph is systematically expored with the aid of *Node.mark*, and all paths from *nodes{e2}* to *nodes{e1}* are calculated Mehlhorn (1984, p.17). Since there is an edge from *nodes{e1}* to *nodes{e2}* all nodes in *equivalent* are on a strongly connected component. Lines 4(d)iii A-D adapt the data structures such that all nodes in *equivalent* are merged into a single node. A proof is skipped.

In line 6, Algorithm 5 is invoked on the graph defined by *nodes.* This algorithm computes a mapping from nodes to integers that conforms to the partial order defined by the graph in *nodes.* This is achieved by recursively ensuring that wherever there is an edge from node n to node m, the field index of m is at least $n + 1$ (line 6). Termination is guaranteed by the use of the *mark* flag. Correctness follows from the discussion. After termination of Algorithm 5, the integer *field* of data type *Node* holds the desired assignment to integers.

Since the computation of these integers is started from the set of nodes in *initial*, the assignment to integers is *left-aligned.* With noting this bias of the algorithm, I refer to footnote 6. An alternative bias, which is slightly more complex would be starting the search at the position of the head in the field description.[7]

For illustration, Figure 4.9 presents a field description induced from the data presented earlier in Figure 4.6. On this data, 8 fields for nouns were found. The picture shows the order graph, with its nodes aligned according to their field index in the horizontal dimension. Thus, columns of nodes can be directly read as fields. The picture also indicates the learned field description in text format.

4.7 Robustness and Feature Selection

There is a single property of the order graph which has a crucial influence on the eventual shape of the induced field descriptions: the cycles it contains. Imagine,

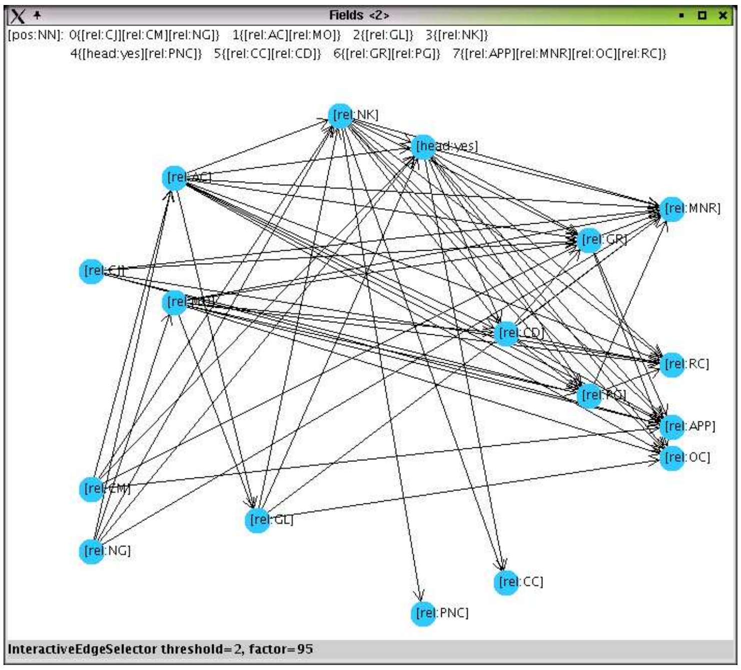
[7]In practice, the alignment bias only plays a role in the face of disjoint chains of different length in the partial order, which is relatively seldom.

Algorithm 4 induce(P)

```
1. INPUT P: a SET of type PrecedencePair={e1:Element; e2:Element}
2. let nodes be a MAP from type Element to type Node={elements: SET of
   type Element; field:int; mark:boolean; outedges,inedges:LIST of type
   Node};
3. let initial be a LIST of type Node;
4. for each (e1, e2) in P for which the edge selection function is true:
   (a) if e1 not in nodes, map it to a new Node({e1},0,false,nil) and add
       nodes{e1} to initial;
   (b) if e2 not in nodes, map it to a new Node({e2},0,false,nil);
   (c) remove nodes{e2} from initial;
   (d) if nodes{e1} is reachable from nodes{e2}
         i. let equivalent be the set of nodes on paths from nodes{e2} to
            nodes{e1}
        ii. create a new node merge
       iii. for all nodes n in equivalent
              A. remove n from nodes and from initial
              B. add all edges, which leave n towards a node that is not in
                 equivalent, to merge.outedges
              C. add all edges, which lead towards n from a node that is not
                 in equivalent, to merge.inedges
              D. insert merge into nodes under all keys which are in
                 n.elements
5. for each n in initial: solve(n,0)
6. OUTPUT: nodes as a map from type Element to an integer Node.field
   for all elements in P
```

Algorithm 5 solve(n,i)

```
1. INPUT n:Node, i:integer;
2. if n.mark, return;
3. set n.mark to true;
4. if n.field<i, set n.field to i;
5. for all nodes m directly reachable from n:
   (a) solve(m,i+1)
```



Fields <2>

[pos:NN]: 0{[rel:CJ][rel:CM][rel:NG]} 1{[rel:AC][rel:MO]} 2{[rel:GL]} 3{[rel:NK]}
4{[head:yes][rel:PNC]} 5{[rel:CC][rel:CD]} 6{[rel:GR][rel:PG]} 7{[rel:APP][rel:MNR][rel:OC][rel:RC]}

InteractiveEdgeSelector threshold=2, factor=95

Figure 4.9: Order Graph and Field Description of Normal Nouns

e.g. a stage in the execution of Algorithm 4 would build a field description $F_1 = \{a\}$; $F_2 = \{b\}$; $F_3 = \{c\}$; $F_4 = \{d\}$, and is about to add a precedence pair $\langle d, a\rangle$. With the insertion of a single edge, the whole description would be conflated to a single node. In the following, I will classify the data that can lead to cycles in the graph, and propose strategies to deal with each of the classes.

There are two formal devices defined in earlier sections how the construction of the order graph can be influenced:

- the edge selection function
- the dependent feature selection function

The aim in employing these devices is to *robustly* construct field descriptions. This means that low-frequency phenomena or errors in the data should not lead to drastically different results. It also means that the system should be able to automatically adapt the feature selection function as it is required to express the characteristics of the data.

As an illustration for the classification of cycles that follows, consider Figure 4.10, a precedence table calculated from 1000 sentences of Tiger, showing the precedence pairs of finite auxiliary verbs. A cycle in the order graph may result from

1. *free variance* of the two elements. In this case, the cycle is justified, the two elements should be included into a common field, and no adaption of the model is necessary.

2. *noise*, i.e. annotation errors in the data, or highly infrequent exceptions. An example is the precedence pair <[head:yes], [rel:SVP]> in Figure 4.10: Separable verb prefixes occurred after the head 63 times, but only twice before the head.[8] A strategy to deal with these type of cycle is to ignore the infrequent precedence pair during edge selection, and not include it in the order graph. This results in a field representation which will reject the exceptional training sentence as ungrammatical. Section 4.7.1 presents a refined definition of the *edge selection function*, which uses thresholds to exclude low-frequency pairs.

3. *a wrong feature selection function.* While a rule governing the occurrence of the two elements is not obvious under this feature selection function, it might be the case that there is a rule governing a subclass or superclass of any of the two elements. Thus, an adaption of the feature selection function that classifies elements orthogonal to the previous classification might reveal the rule. An example is the precedence pair <rel:SB,rel:DA>. While the respective order of datives and subjects is unclear under a feature selection function that considers syntactic relation, a function that considers POS-tag, in particular pronominalisation, might yield more clearcut results. Section 4.7.2 presents a *decision tree learning approach* towards automatic feature selection that iteratively adapts the feature selection function used.

4. *elements that can either occur before or after the head.* This will be considered a special case on the basis of a linguistic argument. Consider the pairs <[rel:SB],[head:yes]> and <[rel:OA],[head:yes]> as examples. The data indicates that subjects, just as with accusatives, can either precede or follow the head, which will result in heads, subjects and accusatives being conflated to the same field. Confer Figure 4.11 for an example of a field description calculated from the data in Figure 4.10. While this is correct behaviour within the theory as it is defined until now, the linguistic analyses discussed in Section 3.3 prefer a representation where the head has a field on its own, and the dependent elements can either precede or succeed it. However, this was rendered impossible by the definition of a field description in Section 4.6. Section 4.7.3 presents a *splitting rule* which modifies the feature selection function by including special head-position features in order to distinguish pre-head items from post-head items.

Before Robust Edge Selection, Automatic Feature Selection and Splitting will be outlined in detail, further attention is devoted to the connection between precedence pair counts and cycles in the graph. In the general case, when discovering a cycle in the graph, the decision must be made to either maintain the cycle and reduce all nodes to a single field, or exclude one of the edges on the cycles from the graph, or adapt the feature selection function of one of

[8]In fact these two instances are both instances of the verb "hinzukommen", so that an exceptional rule that refers to the base-form of the verb might be conceivable. In the view of the author, this would still overfit the data, in this specific case. The two corpus sentences are "Hinzu kommen einige flächige Glasfassaden und – wo Metall verwendet wird – Kupferblech." (Tiger 309) and "Hinzu kommt, daß in der letzten Zeit eine Reihe staatlicher Subventionen für die Grundnahrungsmittel abgeschafft worden sind, dank derer sich die arme Bevölkerung über Wasser halten konnte." (Tiger 377)

the elements on the graph. This would involve a complex calculation based on the frequencies of all the precedence pairs involved in the cycle. It is possible, though, to approximate the general case by a simpler heuristics. To this end, recall that the definition of a precedence pair is based on the *indirectly-precedes-relation.* From this fact follows the following

Heuristic Assumption 1 A cycle of length larger than 2 in an order graph is likely to contain a sub-cycle of length 2.

Consider Figure 4.12 for illustration. The first row shows a cycle of length 1, and the order domain and order pair from which it might have been constructed. The second row shows cycles of length 2, with their respective data, and the third row shows cycles of length 3. The left column shows cycles that contain sub-cycles, while the cycles on the right hand side do not. Details aside, figure (vi) shows that a very specific case of complementary distribution of the elements involved is necessary for a cycle of length larger than 2 to emerge, which does not contain a sub-cycle.

Heuristic Assumption 2 A metric based on a comparison of $c(\langle e_1, e_2\rangle)$ and $c(\langle e_2, e_1\rangle)$ is a good approximation of the likelihood whether this precedence pair will be involved in a cycle.

With these pre-considerations, I can now define a probabilistic measure of quality of a precedence pair.

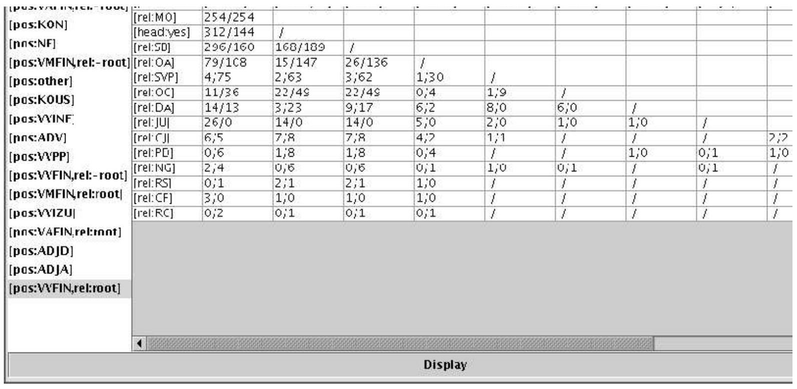
Inverse Precedence Pair Call the precedence pair $p^{-1} = \langle e_2, e_1\rangle$ the *inverse precedence pair* of a precedence pair $p = \langle e_1, e_2\rangle$

Precedence Likelihood The precedence likelihood of a precedence pair p is defined as $P(p) = \frac{c(p)}{c(p)+c(p^{-1})}$

Conflict Ratio The *conflict ratio* of two elements is a function $q : E \times E \rightarrow [0, 1]$, and can serve as a basis for edge selection and automatic feature selection. Define $q(e_1, e_2) = 1 - (2 \cdot |p(\langle e_1, e_2\rangle - 0.5|)$

The conflict ratio function assigns the highest value, 1, if the count for the precedence pair $\langle e_1, e_2\rangle$ is the same as the count of the inverse precedence pair. It assigns the lowest value, 0, if one of the counts is zero. A precedence pair with a relatively high conflict ratio will be called a *conflicting precedence pair.* Using the examples from above, the precedence pair <[head:yes],[rel:SVP]> has a low conflict ratio, but <[rel:SB],[rel:DA]> and <[rel:SB],[head:yes]> have high conflict ratios.

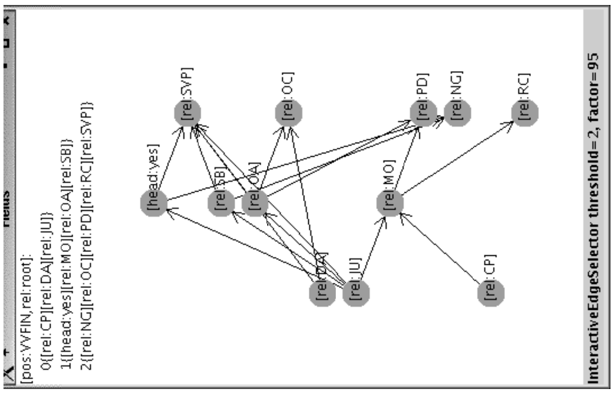
Note that precedence pairs are also constructed for pairs of elements $\langle e, e\rangle$. An abstract example of such a precedence pair is given in Figure 4.12(i), a concrete example is the pair <[rel:MO],[rel:MO]> from Figure 4.10. The conflict ratio of such a precedence pair is 1 by definition, since it was seen in both orders in the data. A precedence pair of the form $\langle e, e\rangle$ will be called a *self-conflicting precedence pair*. Self conflicting precedence pairs can occur in the data only due to *multiple occurrence* of the underlying element within the same domain. Multiple occurrence can either be *compact* as the occurrence of *a* in Figure 4.12(i), or *distributed* as in Figure 4.12(ii).



[pos:KON]
[pos:NF]
[pos:VMFIN,rel:-root]
[pos:other]
[pos:KOUS]
[pos:VVINF]
[pos:ADV]
[pos:VVPP]
[pos:VVFIN,rel:-root]
[pos:VMFIN,rel:root]
[pos:VVIZU]
[pos:VAFIN,rel:root]
[pos:ADJD]
[pos:ADJA]
[pos:VVFIN,rel:root]

[rel:MO]	254/254								
[head:yes]	312/144	/							
[rel:SB]	296/160	168/189	/						
[rel:OA]	79/108	15/147	26/136	/					
[rel:SVP]	4/75	2/63	3/62	1/30	/				
[rel:OC]	11/36	22/49	22/49	0/4	1/9	/			
[rel:DA]	14/13	3/23	9/17	6/2	8/0	6/0	/		
[rel:JU]	26/0	14/0	14/0	5/0	2/0	1/0	1/0	/	
[rel:CJ]	6/5	7/8	7/8	4/2	1/1	/	/	/	2/2
[rel:PD]	0/6	1/8	1/8	0/4	/	/	1/0	0/1	1/0
[rel:NG]	2/4	0/6	0/6	0/1	1/0	0/1	/	0/1	/
[rel:RS]	0/1	2/1	2/1	1/0	/	/	/	/	/
[rel:CF]	3/0	1/0	1/0	1/0	/	/	/	/	/
[rel:RC]	0/2	0/1	0/1	0/1	/	/	/	/	/

Figure 4.10: Precedence Table of finite verbs



Fields

[pos:VVFIN,rel:root]:
0{[rel:CP][rel:DA][rel:JU]}
1{[head:yes][rel:MO][rel:OA][rel:SB]}
2{[rel:NG][rel:OC][rel:PD][rel:RC][rel:SVP]}

Figure 4.11: Order Graph and Field Description of finite verbs

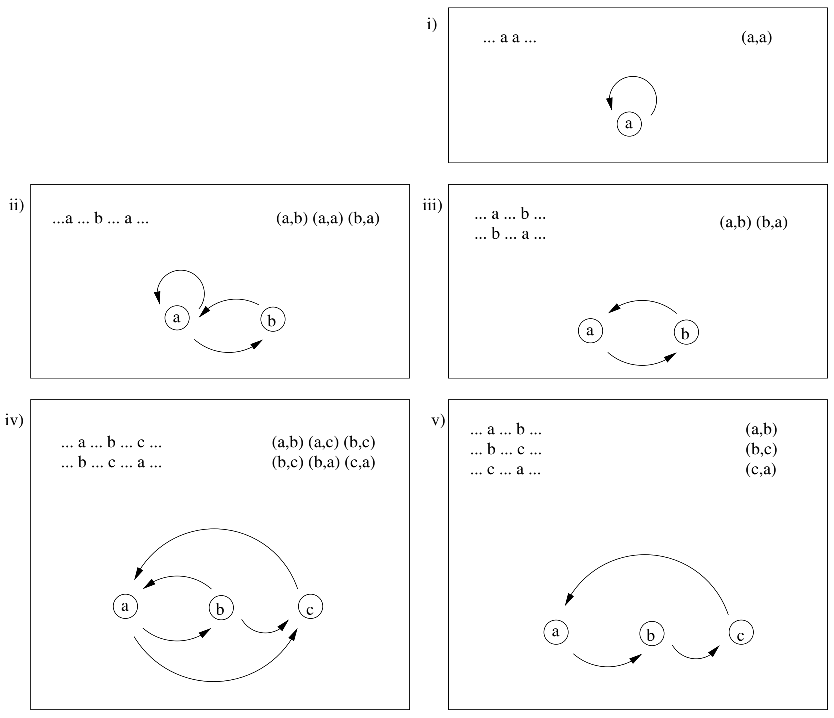


Figure 4.12: Types of cycles

4.7.1 Robust Edge Selection

The way is now paved for elaborating on the definition of the edge selection function defined in Section 4.6, following the strategy outlined in point 2 of Section 4.7 above. Recall that Algorithm 4 iterates the precedence pairs for which the edge selection function returns true in line 5, and builds the order graph from these precedence pairs. In Section 4.6, a simple edge selection function was defined as

$$s_h(e_1, e_2) := \ 1 \text{ iff } \exists \langle e_1, e_2, h \rangle \in P_h$$

which *selects* every pair, i.e. returns true for every pair. This includes conflicting (and self-conflicting) precedence pairs.

The edge selection function will now be changed to not include conflicting precedence pairs into the order graph. First, note that self-conflicting precedence pairs, which result in cycles of length 1, do not add any information to the order graph, so they can be safely ignored. Precedence pairs with a high conflict ratio lead to a cycle in the graph of length 2 by definition, therefore both elements will be conflated to a single node. Thus conflicting precedence pairs should be ignored too. Precedence pairs with conflict ratio 0 or close to 0, in contrast, should be selected robustly: Here, the more probable sequence should be chosen, ignoring the data into the opposite direction. This avoids a cycle of length 2. By the inverse of Heuristic Assumption 1, it will also avoid a cycle of greater length.

Besides the exclusion of conflicting precedence pairs, the edge selection function should also exclude precedence pairs that are not justified on much training data. A threshold will be introduced that ignores low-frequency precedence pairs.

The discussion is summed up in the following robust edge selection function:

$$s_h(e_1, e_2) := \ 1 \text{ iff } c(\langle e_1, e_2 \rangle) + c(\langle e_2, e_1 \rangle) > T_c \text{ and } p(\langle e_1, e_2 \rangle) > T_p$$

The function relies on threshold values T_c, the count threshold, and T_p, the ratio threshold. The term *conflicting precedence pair* can now be refined as referring to a precedence pair whose conflict ratio is above the ratio threshold, and which is therefore ignored during edge selection.

The edge selection function is a means to robustly exclude low-frequency precedence pairs from being included into the order graph. More sophisticated edge selection is conceivable, which employs true statistical testing rather than a threshold technique.

4.7.2 Automatic Feature Selection

This section comes back to point 3 of Section 4.7 above, which argued that a conflicting precedence pair may be due to an inadequate feature selection function. Recall from section 4.5 that the head feature selection function is a function from words to head elements, $\tilde{f} : W \rightarrow E$ and the dependent feature selection function is from words to dependent elements, $f : W \rightarrow E$. Imagine a conflicting precedence pair $\langle e_1, e_2, h \rangle$. This precedence pair was constructed from a set of precedence pair instances, according to the definitions in Section

4.5, viz. the precedence pair instances with $f(w_1) = e_1$, $f(w_2) = e_2$, $f_H(w_h) = h$. The inverse images of e_1, e_2, and h, $f_h^{-1}(e_1)$, $f_h^{-1}(e_2)$ and $\tilde{f}^{-1}(h)$ are the sets of first element nodes, second element nodes, and head element nodes.

The instances of the conflicting precedence pair can now be analysed in order to change the feature selection function. The following cases can be distinguished:

1. a subclass of $f^{-1}(e_1)$ is non-conflicting with $f^{-1}(e_2)$, or vice versa
2. a subclass of $f^{-1}(e_1)$ is non-conflicting with a subclass of $f^{-1}(e_2)$
3. a subclass of $f^{-1}(e_1)$ is non-conflicting with a superclass of $f^{-1}(e_2)$, or vice versa
4. $f^{-1}(e_1)$ and $f^{-1}(e_2)$ are non-conflicting under a subclass of $f^{-1}(h)$

In the general case, there is no restriction on the cardinality adaptations of the sets E and H, and the search space for finding the best head and dependent feature selection functions is the set of functions from W to E, and from W to H. Clearly, the search space needs to be restricted to be computationally feasible. A first restriction I will make is to disallow adaptations of the feature selection function of type 3, where a superclass of an already established element $e \in E$ is considered. This will lead to an algorithm that *monotonically and greedily* increases the cardinality of E. Secondly, I will exclude automatic adaptation of the head feature selection function (type 4), although the approach will turn out to be, in principle, augmentable into this direction. Points 1 and 2 above will be uniformly treated.

Section 4.5 chose non-recursive, linguistic feature structures as a representation for word forms, i.e. as members of set W. Nothing was said about the nature of sets E and H for manual feature selection. For automatic feature selection, though, I will now assume that also E and H are sets of non-recursive feature structures, and present a general model of a feature selection function.

Feature structures can be arranged in a lattice according to the subsumption relation. Assume the feature structures depicted in Figure 4.13 (See Appendix B for a definition of subsumption). The *least upper bound* of a feature structure is the unique most specific feature structure that subsumes it, if it exists. In the example, the least upper bound of [a:y,b:z] is [b:z] (in symbols, [b:z] $\sqsubseteq$ [a:y,b:z]), while [a:x,b:y] does not have a least upper bound. However if a subsumption hierarchy is to be employed in the definition of the feature selection function for automatic feature selection, a unique assignment of words to feature structures is needed. Therefore, I define the operation of *first least upper bound* on feature structures.

First Least Upper Bound Let A be a set of feature structures. Let x be a feature structure. Let R be a total order on A. *The first least upper bound of x in A according to R* is the smallest $a \in A$ according to R such that $a \sqsubseteq x$, $\neg\exists\, a' \in A : a \sqsubseteq a' \wedge a' \sqsubseteq x$.

If we assume the order R to be reflected by the left-to-right arrangements of the feature structures in Figure 4.13, the first least upper bound of [a:x,b:y] is [a:x].

The general form of a feature selection function for automatic feature selection can now be framed as follows:

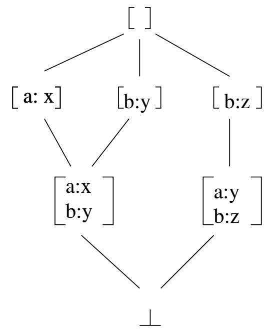


Figure 4.13: Subsumption hierarchy of feature structures

Feature Selection Function for Automatic Feature Selection $f : W \to E$ is a feature selection function. W and E are sets of non-recursive feature structures. R is a total order on E. Then, for automatic feature selection, let $f(w)$ be the first least upper bound of w in E according to R.

Another assumption needs to be made more precise. Feature selection functions for manual feature selection were sometimes parameterised with a subscript in Section 4.5. It was assumed in Section 4.4 that the order of two elements e_1, e_2 depends on features of e_1, e_2, and their common climb tree head. For automatic feature adaption, I explicitly stipulate that there is an own dependent feature selection function for each head element $h \in H$:

Dependent Feature Selection Function Depends On Head Class The dependent feature selection function for automatic feature adaption, f, is parametrised with the head class. $f : H \to (W \times W \to E)$. $f_{h,w} : W \to E$ ($w \in W$, $h = \tilde{f}(w)$) indicates the dependent feature selection function for a head node w, which is of head class h.

Assuming a different dependent feature selection function for each class of head extends the assumption of Section 4.4, attributing a special role to the head. There is a close correspondence to the type hierarchy of Villavicencio (2000). As an example, consider the head classes of matrix clause verbs and verbs occurring in subordinate clauses. The dependent feature selection function of verbs in subordinate clauses may contain a class for relative pronouns in its set of elements, E, while this element is not necessary to describe the order of elements under verbs at matrix clause level.

The considerations above lead to an algorithm that iteratively judges the conflict ratio of all precedence pairs of a given head class, picks one, and considers all instances of these precedence pairs. It then determines additional features that prove relevant to the order of the precedence pair, and monotonically extends the feature selection function by adding new classes to the range of the dependent feature selection function. The algorithm is sketched in Algorithm 6, and will be commented on in the rest of this section.

Step 1 defines the starting point for search. The starting point of a search algorithm may have significant influence on the outcome. A possible candidate for step 1 is the feature selection function that distinguishes head and dependent

Algorithm 6 Automatic Feature Selection Function Adaption

```
1. Start with a trivial feature selection function
2. Pick a suitable precedence pair from the precedence table
3. Find feature-value pairs that predict the order of the elements of the
   chosen precedence pair in the data, and add these features to the range
   of the feature selection function
4. While not converged, go to step 2
```

items only. There is a choice, though, of how to treat climbed elements. They can either be included into a third class of "climbed dependents", or included among the dependent items. In the following, the feature structure [head:yes] is used to refer to the head element, and the feature structure [] is used to refer to any other element.

As a heuristic in step 2, I propose to pick the precedence pair with highest conflict ratio. This means that the most serious sources of conflicts in the data are tackled first. This heuristic may also have influence on the eventual feature selection function.

As indicated before, the method of choice in step 3 is decision tree learning. This means the system will inherit all the characteristics of this technique that were outlined in Section 2.4. What is unusual about the algorithm, however, is that decision tree learning is embedded into a larger iterative algorithm, which chooses a subsection of the data the decision tree learner receives as input in step 2.

The task of the decision tree learner is to add appropriate feature structures to the lattice of feature structures E, as subclasses of e_1 and e_2. These features should be those which predict the surface order of e_1 and e_2 best. This means, the output parameter is to be the surface order of e_1 and e_2 (either e_1 precedes e_2 or vice versa). What is needed in order to represent the data in this format is some external criterion on which to distinguish e_1 and e_2. For this purpose, recall that a precedence pair is defined as a tuple $\langle e_1, e_2, h\rangle$, $e_1, e_2 \in E$; $h \in H$. An equivalent view on a precedence pair is the following.

Precedence Pair (Equivalent Formalisation) A precedence pair $\langle e_1, e_2, h\rangle$ can equivalently be defined as a quadruple $\langle e_{small}, e_{large}, h, r\rangle$, $r \in \{\lhd, \rhd\}$. Let e_a be the smaller of e_1 and e_2, and e_z the larger, according to an arbitrary *order on elements, L*, e.g. the lexicographic order on feature structures. Let $r = \lhd$ if $e_1 = e_a$ and $r = \rhd$ otherwise.

For example, the precedence pair ⟨[rel:OC], [rel:SB], [pos:VVFIN]⟩ can equivalently be written as ⟨[rel:OC], [rel:SB], [pos:VVFIN], $\lhd$⟩, and ⟨[rel:SB, [rel:OC], [pos:VVFIN]⟩ is equivalent to ⟨[rel:OC], [rel:SB], [pos:VVFIN], $\rhd$⟩.

Figure 4.14 shows example data calculated on a small test corpus. The data lists the precedence pair instances of a conflicting precedence pair ⟨[], [head:yes], [pos:VVFIN,rel:-root]⟩. This is the first precedence pair chosen during adaption of feature selection function of finite verbs. The features *posh* and *relh* are the POS-tag and syntactic relation of the head, *posa, rela, posz, relz* are the same features for e_a and e_z respectively. Figure 4.15 shows all rules actually

```
posh,relh,posa,rela,posz,relz,out
---------------------------------
VVFIN, CJ, VVIZU, OC, VVFIN, CJ, >.
VVFIN, CJ, VVINF, SB, VVFIN, CJ, <.
VVFIN, CJ, PPER, DA, VVFIN, CJ, >.
VVFIN, CJ, PTKNEG, NG, VVFIN, CJ, >.
VVFIN, CJ, PPER, SB, VVFIN, CJ, <.
VVFIN, CJ, NN, OA, VVFIN, CJ, >.
VVFIN, CJ, ADV, MO, VVFIN, CJ, <.
VVFIN, CJ, VAFIN, MO, VVFIN, CJ, <.
VVFIN, CJ, PPER, SB, VVFIN, CJ, >.
VVFIN, CJ, PRF, OA, VVFIN, CJ, >.
VVFIN, CJ, NN, MO, VVFIN, CJ, >.
VVFIN, CJ, NN, MO, VVFIN, CJ, >.
VVFIN, SB, KOUS, CP, VVFIN, SB, <.
VVFIN, SB, CARD, MO, VVFIN, SB, <.
VVFIN, SB, NN, SB, VVFIN, SB, <.
VVFIN, SB, VVINF, OC, VVFIN, SB, >.
VVFIN, RC, PRELS, SB, VVFIN, RC, <.
VVFIN, RC, NN, OA, VVFIN, RC, <.
...
```

Figure 4.14: Input data representation for decision tree learner. Precedence Pair instances of precedence pair <[],[head:yes],[pos:VVFIN,rel:-root]>

discovered by the decision tree learner, in the output format of the C4.5 decision tree learner (Quinlan (1998)). The output variables $\lhd, \rhd$ occur as `<`, `>` in the figure. The second rule found (Rule 09), e.g., predicts that if the POS of the smaller element is a relative pronoun, it is likely to occur to the left of the head. Rule 11 predicts that if the relation of the smaller element is "negation", it is likely to occur to the right of the head. Rule 04 found two relevant features. It predicts that dative pronouns will occur to the right hand side of the head, with a low level of reliability. Note that the decision tree learner also finds rules that refer to features of the head (*relh, posh*). I will comment on such rules at the end of the section.

One point remains unanswered: The lexicographic order is not uniquely defined in case of a *self-conflicting precedence pair* (see page 4.7), since neither $e \lhd e$ nor $e \rhd e$. The solution to this problem is simple. For a self-conflicting precedence pair, each of its precedence pair instances is fed to the decision tree learner twice, once in either direction. As a consequence, rules are discovered twice.

Another detail shall not remain unmentioned: It was noted in Section 2.4 that rule antecedents need not be disjoint. This means that if classes established by rule antecedents are integrated into the feature structure lattice, feature structures need not be disjoint, possibly resulting in a situation as in Figure 4.13 above, where there is no unique upper bound. The order implicit in the implementation conforms to the point of time when an element was added to the range of the feature selection function.

Before I present the final algorithm, with all of the above discussion integrated, I propose a simple convergence strategy for step 4, which signals convergence as soon as no new classes where found. Chapter 5 will go a little deeper

```
Final rules from tree 0:
Rule 20:  relh = RC -> class < [95.3%]
Rule 09:  posa = PRELS -> class < [93.9%]
Rule 03:  rela = CP -> class < [93.0%]
Rule 14:  rela = SB -> class < [83.8%]
Rule 16:  relh = MO -> class < [82.7%]
Rule 19:  relh = OC -> class < [77.1%]
Rule 05:  posa = ADJD -> class < [75.6%]
Rule 06:  posa = ADV -> class < [73.8%]
Rule 10:  posa = VVFIN -> class > [84.1%]
Rule 11:  rela = NG -> class > [70.7%]
Rule 15:  rela = SVP -> class > [70.7%]
Rule 13:  rela = RC -> class > [63.0%]
Rule 12:  relh = CJ, rela = OA -> class > [57.4%]
Rule 07:  relh = CJ, rela = MO -> class > [55.6%]
Rule 04:  posa = PPER, rela = DA -> class > [50.0%]
Rule 01:  relh = APP -> class > [45.3%]
Rule 18:  posa = VAFIN -> class > [45.3%]
Default class:  <
```

Figure 4.15: Learned decision tree rules for the data in Figure 4.14. A rule consists of a rule number, an antecedent, a prediction, and a reliability estimate.

[head:yes]	735	[rel:MO]	76
[pos:NN,rel:MO,split:pre-hd]	258	[split:pre-hd]	74
[rel:SB,split:pre-hd]	245	[rel:SVP]	68
[pos:ADV,rel:MO,split:pre-hd]	202	[pos:ADV,rel:MO,split:post-hd]	67
[pos:PRELS]	195	[pos:ADJD,rel:MO,split:pre-hd]	64
[rel:CP]	185	[rel:SB,split:post-hd]	60
[pos:NN,rel:OA,split:pre-hd]	153	[rel:OC,split:post-hd]	52
[pos:PPER,rel:SB,split:pre-hd]	141	[pos:ADJD,rel:MO,split:post-hd]	33
[pos:NN,rel:MO,split:post-hd]	131	[pos:PRF,rel:OA,split:post-hd]	32
[pos:NN,rel:OA,split:post-hd]	95	[rel:OC,split:pre-hd]	31
[pos:PRF,rel:OA,split:pre-hd]	85	[pos:ADJA,rel:MO,split:pre-hd]	30

Table 4.1: Learned classes

into this issue.

Algorithm 7 sums up the discussion above in a more detailed instantiation of Algorithm 6 from above. Step 3(c) provides details of how to integrate the new feature values found into the lattice of feature structures, relying on the operation of feature structure unification (see Appendix B for a definition). As an example, consider the rules from Figure 4.15. Here, the precedence pair chosen in step 2 is <[],[head:yes]>. One of the rules discovered by the decision tree learner refers to "posa=PRELS". The smaller of the elements [] and [head:yes] according to the lexicographic order is [], thus the rule refers to []. In consequence, [] $\sqcup$ [pos:PRELS] = [pos:PRELS] is added to the domain of the feature selection function.

Table 4.1 lists the most frequent elements discovered by a complete run of Algorithm 7 for finite verbs. The *split* features will be commented on in the next section.

Algorithm 7 Automatic Feature Selection Function Adaption

```
1. Let f be a dependent feature selection function with
   E={[head:yes],[head:no,dist:0],[dist:>0]}
2. let <e1,e2,h> be the next precedence pair of h with highest conflict ratio
3. Find feature value-pairs that significantly predict the order of the chosen
   precedence pair in the data
   (a) let {<w1,w2,h>} be the instances of <e1,e2,h> in the corpus
   (b) run a decision tree learner with all features of w1,w2,h as input
       variables, and output variable R={<,>}
   (c) for all decision tree rules found which are above a significance
       threshold
       i. for all features [f1:v1,f2:v2,...] in the rule that refer to w1, add
          e1⊔ [f1:v1,f2:v2,...] to E
       ii. for all features [f1:v1,f2:v2,...] in the rule that refer to w2, add
          e2⊔ [f1:v1,f2:v2,...] to E
4. While new classes were found, go to step 2
```

Automatic feature adaption can be summed up as follows. I assume that there is a set of dependent feature selection functions (similar to Villavicencio (2000)), one for each class of heads. An iterative algorithm adapts each feature selection function, starting from an initial trivial feature selection function. In each iteration, one conflicting precedence pair is heuristically chosen for each precedence table at a time. The precedence pair instances of the precedence pair are retrieved, and fed into a decision tree learner, which returns rules that suggest possible subclasses that show less conflict. These rules are used to monotonically extend the range of the feature selection function.

I assume the head feature selection function as static throughout this thesis. Note, however that the decision tree learner produces rules that refer to features of the head also.[9] Algorithm 7 ignores such rules in steps 3(c)i and 3(c)ii. It is possible to include these rules into the algorithm, and use them to adapt the head feature selection function. Experiments need to show whether automatic head class adaption is well-behaved.

4.7.3 Splitting

Point 4 of Section 4.7 above stated that the position of the head within a field description is a linguistically central characteristic. It was illustrated in Figure 4.11 that without an adaption of the model, the system does not attribute this special role to the head element. This section demonstrates that another adaption to the feature selection function leads to the desired behaviour. It is argued that this adaption reflects the linguistic knowledge of the prominent

[9]These rules come in two disguises: Either, there is an explicit reference to a feature ending in “h”, or there is an implicit reference to a feature ending in “a” or “z”, but one of the elements of the precedence pair is the head element.

role of the head in a topological description, and that it is valid to put this knowledge into the system.

It was a characteristic of the definition of a field description as defined in Section 4.6 that elements cannot occur in more than one field. This restriction is now loosened, and will make field descriptions possible where one and the same item occurs in a field before, and after the head. Such an element is called a *split* element:

Split Element a *split element* is an element that can occur in two fields in a field description. A *stable element* is an element for which this is not the case. The two instances of a split element are distinguished by a feature *split* in the element's feature structure, whose value is either *pre-x* or *post-x*, where x is a stable element in the same field description.

To justify this measure on linguistic grounds, compare the concept of a split element to Hawkin's concept of *doubling* Hawkins (1983, p. 11ff). Hawkins speaks of doubling when the same element can occur before or after the head. In the formalisation of splitting, I avoid direct reference to the head element, and speak of the stable element instead. Splitting is general enough to be also applied to two non-head elements. However, it is unclear which of the two should be the stable element in this case. I therefore assume the head always to be the stable element. Algorithmically, care should be taken to only split those elements for which this is necessary, and the default should be not to split elements. Two splitting strategies were actually implemented, and shall be briefly discussed here.

Splitting On Failure Split an element iff automatic feature adaption fails on a chosen precedence pair, and one of the elements is the head element

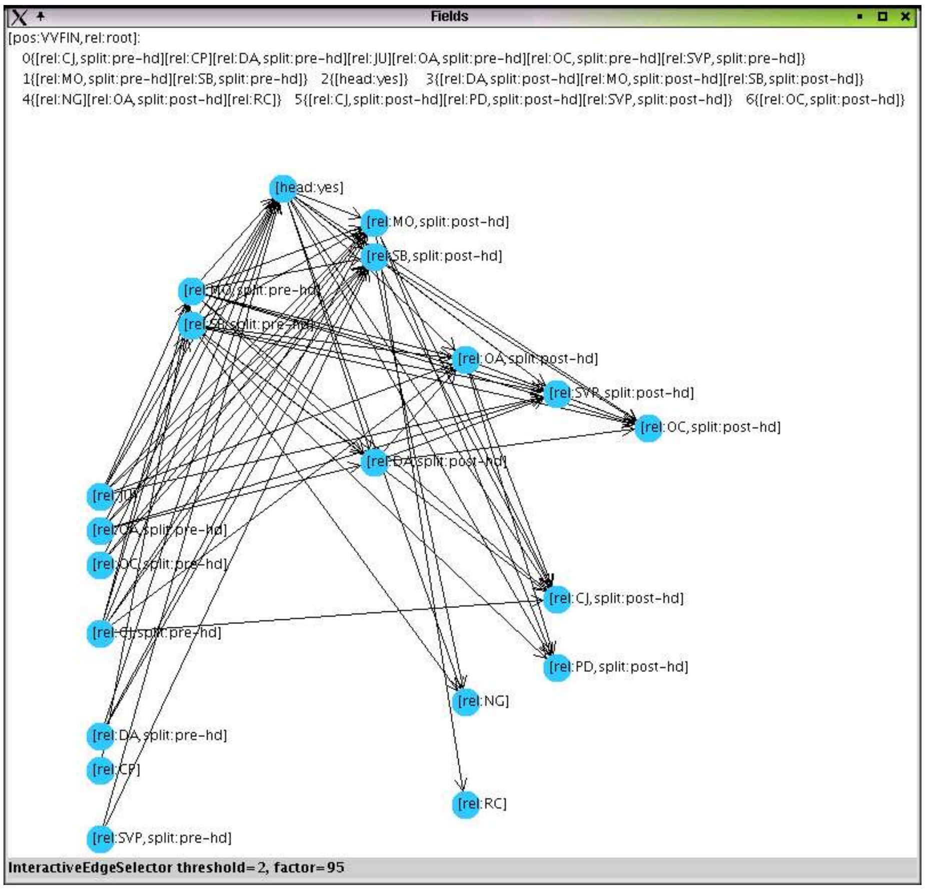
Head Splitting Split every element that conflicts with the head

Splitting On Failure uses splitting as a fall back strategy if the decision tree learner is not able to find a linguistic feature that predicts the order of the conflicting pair. Thus, this strategy is inserted into Algorithm 7 as step 3(d). The disadvantage are long running times. Head Splitting is run after convergence of Algorithm 7, at the advantage of shorter running times, but at the disadvantage of worse integration of Splitting and Decision Tree Learning.

Technically, splitting means that a split-feature must be included into the feature structures of word forms, which is ignored until splitting is enabled on this class. The details of the implementation, and the possible parameters are explained in Sections 5.2 and 5.3.

Figure 4.16 shows a field description calculated from the same data as Figure 4.11, but with splitting enabled. Note that the description contains more nodes, because elements that were only involved in conflicting precedence pairs were not used in the previous field description.

The mathematical consequence of splitting for the definition of a field description is that the constraint that each element can occur in one field only is effectively loosened, although mathematically the two instances of a split element are different objects.



```
[pos:VVFIN,rel:root]:
  0{[rel:CJ,split:pre-hd][rel:CP][rel:DA,split:pre-hd][rel:JU][rel:OA,split:pre-hd][rel:OC,split:pre-hd][rel:SVP,split:pre-hd]}
  1{[rel:MO,split:pre-hd][rel:SB,split:pre-hd]}   2{[head:yes]}   3{[rel:DA,split:post-hd][rel:MO,split:post-hd][rel:SB,split:post-hd]}
  4{[rel:NG][rel:OA,split:post-hd][rel:RC]}   5{[rel:CJ,split:post-hd][rel:PD,split:post-hd][rel:SVP,split:post-hd]}   6{[rel:OC,split:post-hd]}
```

```
InteractiveEdgeSelector threshold=2, factor=95
```

Figure 4.16: Field Description calculated from the same data as Figure 4.11, with splitting

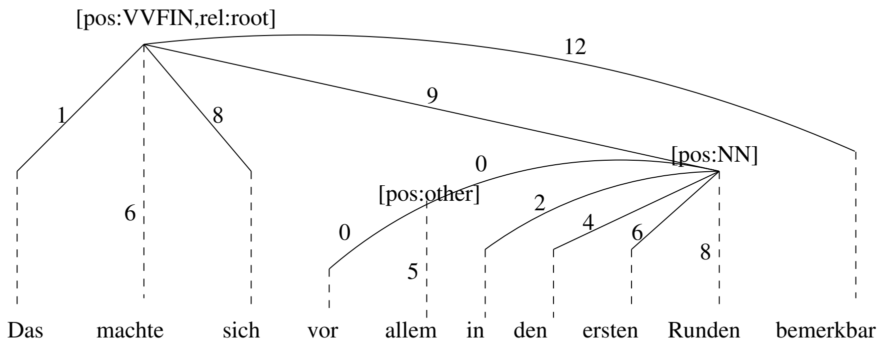


Figure 4.17: Exported Topological Tree

4.8 Field Cardinality and LP Tree Export

This section describes how the learned field descriptions can be used to convert a test treebank of ID structures into an output treebank of learned LP structures. The creation of an LP corpus from a given ID corpus is one of many possible applications of the learned rules. Besides the learned field descriptions themselves, the produced LP corpus plays an important role during evaluation of the system. The section also shows how this conversion process may be used to collect information about field cardinality of the learned fields, and add this information to the learned field descriptions. The calculation of field cardinality constraints was not implemented due to time restrictions.

LP Corpus Export

One way to make use of and evaluate the learned word order rules is to convert a test treebank to a treebank of topological tree structures with aid of the learned word order rules. The input of this module are trees of the same kind as the input of the learning architecture. A corpus of ID structures disjoint from the corpus from which the field descriptions were learned is necessary if export is used for evaluation. The output are TDG LP trees formally. If the learning system is successful, they are also linguistically valuable. Figure 4.17 shows an example of an exported tree.[10] Node labels specify the head class of the node. The downwardspointing edges specify the field index the head element itself is situated in. Edge labels specify the field index at which the dependent node is situated within the field description of its governor. In the example, "machte" is realized in the 6th field of the field description for finite root node verbs, and has four dependents, which are realized in fields 1, 8, 9, and 12 of this field description. "Runden", realized in the field [pos:VVFIN,rel:root]9, in turn offers the field description [pos:NN], and is realized in the 8th field of this field description. It offers fields 0, 2, 4, and 6 for its dependents. For reasons of legability, some node labels were excluded from the figure.

Algorithm 8 presents how an ID tree can be converted to an LP tree with aid of a set of learned field descriptions. The task can be rendered as deciding

[10]The picture was manually created, since the screenshot was not legible. The external tree displayer used often produces illegible output for more complex trees.

```
% "gibt" in "Bei einer Tombola gibt es etwas zu gewinnen "

% [pos:VVFIN,rel:root]: 0{[rel:SB,split:pre-hd][rel:MO,split:pre-hd]}

        1{[head:yes]} 2{[rel:SB,split :post-hd][rel:OA]}

        3{[rel:MO,split:post-hd][rel:SVP]} 4{[rel:CJ]}

1 [dist:1,rel:MO][pos:VVFIN,rel:root]0       false  3  2  10037

2 [head:yes][pos:VVFIN,rel:root]1            true   3  3  10037

3 [rel:SB,split:post-hd][pos:VVFIN,rel:root]2 true  3  4  10037

4 [rel:OC][pos:VVFIN,rel:root]2              false  3  7  10037
```

Figure 4.18: Example output for an unsuccessful rule application, in model MAN-E4, trained on 50 sentences

for each node (a) whether and how far it should climb, and (b) which field label it should be assigned. I will speak of a *realisation* of a word form in a field of a field description. Question (a) is answered by applying the ID tree to climb tree conversion strategy from Section 4.4 to the input ID tree. The solution to question (b) is implemented in line 2(b) of Algorithm 8. Here, the actual sequence of words on the surface, each reduced by the appropriate dependent feature selection function, $f(n)$, is matched against the learned field description, $F(n)$. More particularly, a current index in the field description is iteratively set to the next field that can realise the next word form.

Note that step 2(b) may fail, since the test corpus can contain word order phenomena which were not seen in the training data, or failed to be seen above thresholds. In this case, the node is marked as unsuccessful, and the algorithm is continued at the current field index.

Algorithm 8 LPTree export(IDTree t)

```
1. let c be a climb tree, calculated from t
2. for all nodes n in c, in top-down order
   (a) let F(n) be the field description of n
   (b) for all children m of n, in left-to-right order
       i. let f be the next field in F(n) where f(m) can be realised
       ii. label the edge between n and m with f in the output LP tree
3. return the output LP tree
```

Figure 4.18 shows an example application of the algorithm on the test sentence “Bei einer Tombola gibt es etwas zu gewinnen.” (Negra 10037). “Bei einer Tombola” is classified as a non-local modifier [dist:1, rel:MO]. The learned field description for the head “gibt”, though, does not offer a field for this element. In consequence, word number 2, under word number 3, in sentence 10037 is marked as unsuccessful in line 4 of the example output.

field	1 elements	2 elements	3 elements	4 elements	5 elements
[pos:NN]4	374	110	14	1	0
[pos:VAFIN,rel:root]0	5	0	0	0	0
[pos:VAFIN,rel:root]4	27	0	0	0	0
[pos:VVFIN,rel:root]4	41	24	4	2	1

Table 4.2: Field Cardinalities of selected fields

Field Cardinality

Constraints on the number of elements which can occur in a single topological field are part of the topological fields analyses presented in Section 3.2. For example, the verbal *vorfeld* is generally assumed to offer space for one constituent only. Rules of this kind are not generated by the system as outlined until now, but it is easily possible to infer field cardinalities from the exported LP corpus, whose creation was mentioned in the previous paragraph. This is achieved by counting the number of realisations in each field of each field description in the exported data. Table 4.2 shows some example data for fields that occur in Figure 4.17. The data shows very clearly that the cardinality of the pre-modification field of nouns is unrestricted, while the verbal vorfeld of finite auxiliaries is restricted to 1. Surprisingly, the data also indicates a restriction of the middlefield of finite auxiliaries to 1 ([pos:VAFIN,rel:root]4), but not for finite full verbs ([pos:VVFIN,rel:root]4).

Following Höhle (1986) and Duchier and Debusmann (2001), I assume that a field can either have a fixed cardinality of 1, or an unrestricted cardinality. The calculation of field cardinality constraints from data of the kind in Table 4.2 was excluded from the implementation of the system due to time restrictions.

4.9 Output Data

Before the implementation of the system will be described and finally evaluated in the next chapters, this section reflects upon the form of output data produced, providing a basis for possible applications and evaluation.

Three kinds of output data are interesting for different kinds of evaluation and application.

- the topological corpus
- the field descriptions
- the precedence tables

The topological corpus contains trees like the one in Figure 4.17. Since these trees are projective by definition, they can be converted to widely used bracket structures, as e.g. employed by the Penn Treebank. The topological tree structures of the system do not have non-terminal or pre-terminal nodes. Rather, they encode occurrence in the same field by identical consecutive field labels. The conversion to a bracket structure therefore involves conflating consecutive nodes with the same field label to a single field, and attaching them under a common non-terminal field label node. Figure 4.19 shows the same sentence in bracket representation. Nodes where the application of the learned rules failed

```
("[pos:VVFIN,rel:root]6" ("[pos:VVFIN,rel:root]1" ("POS" ("Das 0")))
("[pos:VVFIN,rel:root]6" ("POS" ("machte 1"))) ("[pos:VVFIN,rel:root]8" ("POS"
("sich 2"))) ("[pos:VVFIN,rel:root]9" ("[pos:NN]0" ("[pos:other]0" ("POS"
("vor 3"))) ("[pos:other]5" ("POS" ("allem 4")))) ("[pos:NN]2" ("POS" ("in 5")))
("[pos:NN]4" ("POS" ("den 6"))) ("[pos:NN]6" ("POS" ("ersten 7"))) ("[pos:NN]7"
("POS" ("Runden 8")))) ("[pos:VVFIN,rel:root]12" ("POS" ("bemerkbar 9"))))
```

Figure 4.19: Exported LP tree in bracket representation

are identifiable through the logging data provided by Algorithm 8, and displayed in Figure 4.18.

The field descriptions consist of a sequence of fields for each head, each field specifying a cardinality, and the elements in these fields. Also the implicit order on elements for feature selection needs to be specified. An XML format was designed to encode this data. Appendix C shows field index and elements only, for each head.

The field descriptions can also be converted to a concrete grammar formalism. Another XML format was specified as an interface to the TDG grammar formalism. An example is given in Figure 4.20. The tag *offerrules* defines for each class of head items which field labels this class offers, and encodes the cardinality of the field (unrestricted, exactly one, one or zero). The tag *acceptrules* specifies for each class of dependent elements which field labels it accepts. There is a designated class of elements, [head:yes], which specifies what TDG calls the "internal fields". For example, "[pos:NN]5" is the internal head field for nouns. The dependent class [rel:DA] specifies that it can be realised, among other fields, in the vorfeld of auxiliary verbs.

Finally, I comment on the information encoded by precedence tables. For each field description, its corresponding precedence table contains frequency counts of the underlying precedence pairs. In the implementation, precedence tables can be browsed, and corpus data can be displayed for each precedence pair. This makes decisions taken by the system transparent. The frequency information inherent in precedence table may also serve as a basis for probabilistic language models, but this issue is out of scope of this thesis.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE tdggrammar SYSTEM "tdggrammar.dtd">
<tdg>
  <grammar>
    <offerrules>
      <headclass headlabel="[pos:NN]">
        <offers fieldlabel="[pos:NN]0" card="*"/>
        ...
        <offers fieldlabel="[pos:NN]9" card="*"/>
      </headclass>
      <headclass headlabel="[pos:VAFIN,rel:-root]">
        <offers fieldlabel="[pos:VAFIN,rel:-root]0" card="*"/>
        ...
        <offers fieldlabel="[pos:VAFIN,rel:-root]9" card="*"/>
      </headclass>
      ...
    </offerrules>
    <acceptrules>
      <depclass id="[head:yes]">
        <accepts fieldlabel="[pos:NN]5"/>
        <accepts fieldlabel="[pos:VAFIN,rel:-root]5"/>
        ...
      </depclass>
      <depclass id="[rel:DA]">
        <accepts fieldlabel="[pos:VAFIN,rel:-root]2"/>
        <accepts fieldlabel="[pos:VVINF]1"/>
        <accepts fieldlabel="[pos:VVPP]1"/>
        ...
      </depclass>
      <depclass id="[rel:GL]">
        <accepts fieldlabel="[pos:NN]2"/>
        <accepts fieldlabel="[pos:NE]2"/>
      </depclass>
      ...
    </acceptrules>
  </grammar>
...
</tdg>
```

Figure 4.20: Subsection of the exported TDG grammar in XML format

Chapter 5

Implementation

This chapter describes the implementation of the word order learning architecture outlined in Chapter 4. Section 5.1 describes design principles and main features of the software, and explains how to access and use the software. Section 5.2 outlines the module structure of the implementation in more detail, and mentions the most important implementational decisions. Section 5.3 explains the effects of the system parameters.

5.1 Overview

Main learning architecture The architecture described in Chapter 4 is fully implemented in Java 1.4 Java (2003). The java code is platform-independent. An external decision tree learner, the C4.5 system of Quinlan (1998) is invoked from within the system. An external tool for displaying dependency trees (in interactive mode) by Denys Duchier is also used. The system can also be run without either of the external modules. The top level executable java class is called learn.WordOrderExtractor.

Options are passed to the WordOrderExtractor by command line options, or through an options file. Several option files are provided, each specifying one parametrisation of the system (a *model*). A simple shell script, run, takes the name of an options file as parameter, creates a subdirectory for output, and invokes the entire system, using the specified option file. A pathname to input training and test corpora is provided as one of the options. The system features an optional interactive GUI, implemented in Java Swing, which allows to browse the data (option *display*). It also features an option *tracetrain*, which creates snapshots of the system during training. An option *reconstruct* reads in output files, instead of re-calculating them. Output files are either in plain text, or in XML format.

Evaluation shell scripts The package also includes several scripts for evaluation, restricted to Unix/Linux platforms, which interpret the output files generated by a single model, or compare the output files of several models. These scripts are implemented in bash, sed, and Perl. gnuplot is used for plotting graphical data to Postscript. The Saxon XML/XSL parser converts XML data to graphical plots, HTML, or latex. EVALB calculates Parseval measures.

system option	default path	description
rootdir	-	top directory
outputdir	LEARN/data/	parent directory for any system output
outputprefix	LEARN/data/current/	subdirectory for system output
corpusdir	corpora/	location of input corpus
corpus	corpus.export	name of training corpus, in corpusdir
testcorpus	-	name of test corpus, in corpusdir
learnerBin	R8/bin/	path to decision tree learner C4.5
displayer	LEARN/oz/TreeDisplayer	path to Oz dependency tree displayer

Table 5.1: Default directory structure, and corresponding system options

Further documentation The entire Java implementation is documented using javadoc. Javadoc documentation is integrated with the source code, and a special documentation compiler converts interface specifications and documentation to HTML.

Installation The software, including all external modules, is available through a CVS repository. The top-level module name is DIPLOM. After checking out from the CVS, invoke DIPLOM/MakeAll, or follow these steps

1. include DIPLOM/LEARN/bin into the Java CLASSPATH variable
2. change to DIPLOM/LEARN/java, and type make. This compiles all java code to LEARN/bin, and creates the HTML documentation using javadoc at LEARN/doc, with index.html being the top level file.
3. compile the decision tree learner following the instructions in DIPLOM/R8/ReadMe, and move the binaries to DIPLOM/R8/bin. If desired also compile the tree displayer at DIPLOM/LEARN/oz by invoking ozc -x TreeDisplayer.oz

Directory structure The learning architecture is included in package DIPLOM/LEARN. LEARN/java contains the java sources, LEARN/bin is meant for compiled Java, and LEARN/doc for compiled Java documentation. LEARN/data is the parent directory of system output, and LEARN/data/evaluate contains the evaluation scripts, which write their output to LEARN/data/evaluate/results. The external decision tree learner is included at R8. EVALB and viewtree are provided at beckerAndFrank. corpora contains links to the input corpora, and scripts which process these corpora. All paths relevant to the main system can be changed with the aid of system options. Figure 5.1 summarises the most important directories, along with the name of the system options that can change the default settings.

Invocation The main system can be manually invoked with java learn.WordOrderExtractor –option1=value1 –option2=value2 ... An option takes the value from the command line, from the file WordOrderExtractor.options, or a default value in this order. The syntax of the option file is a sequence of lines of the form option2=value2, or lines starting in "%" for comments. A Java XML parser needs to be specified with the -D option, if option *reconstruct* is to be used.

file name	description
ClimbPatterns.txt	details about which elements climbed where during ClimbTree conversion
OrderPatterns.data.txt	For each node in the corpus, its representation as a set of precedence pairs
FieldPatterns.xml	Final XML representation of the learned word order rules
ExportLPPatterns.trees	Final bracket structure representation of the exported LP corpus
ExportLPPatterns.log.txt	Details about learned rule applications during LP corpus export
Adaptor.log.txt	Log on automatic feature adaption
WordOrderExtractor.err	Top-level-output, with echo of options, starting time, and execution trace

Table 5.2: Output files

Several option files, WordOrderExtractor.options.modelname are provided. The shell script run [modelname] copies the appropriate option file to WordOrderExtractor.options, creates a subdirectory called modelname for the output, and invokes java in the background, redirecting any output to a log file WordOrderExtractor.err in the output directory.

The evaluation shell scripts at DIPLOM/LEARN/data/evaluate are invoked with a model name, and sometimes with the number of an iteration ("iteration prefix"), if option *tracetrain* was enabled on the model. Output of these scripts is written to ...evaluate/results/modelname in HTML, Latex, Postscript or plain text format. EVAL-SingleModel generates details about one model, EVAL-CorpusSize collects information about the behaviour of a model during training, EVAL-CompareModels generates figures comparing different models.

Output Files When invoked, learn.WordOrderExtractor creates a number of output files in the outputdir/outputprefix directory. Table 5.2 lists the output files. Besides the output files, there are also log files (*.log) for the purpose of detailed system observations and debugging. If *tracetrain* is enabled, the output files are prefixed with the number of the corpus subsection, followed by a dot.

5.2 Technical Documentation

The Java implementation is about 12,000 lines of code, organised in 12 packages and a total of some 80 classes. Figure 5.1 represents the most important package dependencies as a UML-diagram (Fowler and Scott (2000)). In the following, I briefly describe the most important public classes from each of the packages in a bottom-up fashion, referring back to the theoretical concepts outlined in Chapter 4, which they implement. For each of the classes, I show the signature of its constructor, if it has a public constructor.

Package learn.basics This package provides two classes SystemInput and SystemOutput that serve for accessing the input corpus file, and any of the output files that is to be created. It also provides a single instance of class Options, which encapsulates all system options set for this run of the WordOrderExtractor.

Package learn.corpus This package provides access to the input corpus, in its original annotation format, and in the form of dependency trees which are internally used. Corpus(SystemInput source) implements the Iterator interface, returning instances of class tree.Sentence. A sentence provides two views on a corpus sentence: as it appears in the corpus, represented by class RootMobile(java.io.BufferedReader negraExportFormat), and in the form of a dependency

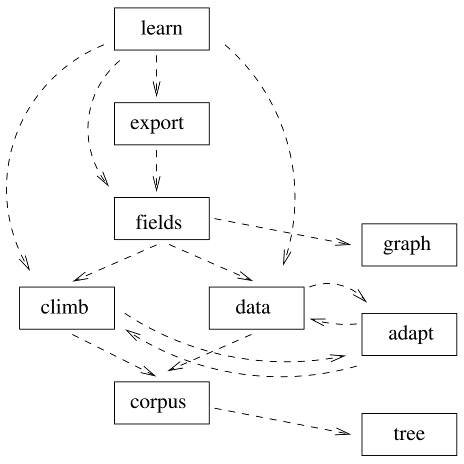


Figure 5.1: Package dependencies

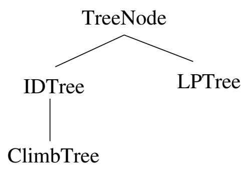


Figure 5.2: Class hierarchy of different types of trees

tree, as described in the tree package. A RootMobile corresponds to the single top node of the NEGRA annotation format, and can be constructed from a sub-portion of a file in NEGRA Export Format. A Mobile (the super-type of RootMobile) corresponds to an arbitrary node in a Negra mobile, and provides methods to distinguish terminal from non-terminal nodes, enumerate the children and parent of a node, and access edge labels, POS information and word forms associated with nodes.

Package learn.tree This package provides a hierarchy of classes that represent different types of syntactic trees. The base class is TreeNode, which provides basic tree operations (Fig. 5.2). Among them is test for dominance between two nodes, locating the root node, and accessing lexical information, which is encapsulated by class Word. Methods crossing() and crossing(TreeNode ancestor) test whether the edge from this node to another node (the actual governor, in the default case) is a *crossing edge* as defined in Section 4.4. Method childrenIterator() enumerates child nodes, method childrenAndHeadIterator() enumerates child nodes and the head itself. This is important for the construction of precedence pairs.

Instances of IDTree are returned by Sentence.getIDTree(). They represent an input sentence to the system, converted from Negra format to a syntactic dependency tree. The private method IDTree Sentence.convert(RootMobile source) implements the conversion strategy outlined in Section 4.3, while Sentence offers

$$\underset{\text{IDTree}}{\begin{bmatrix}\text{pos: VVINF}\\ \text{rel: OC}\\ \text{head: no}\\ \text{split: post–hd}\end{bmatrix}} \quad \underset{\text{ClimbTree}}{\begin{bmatrix}\text{pos: VVINF}\\ \text{rel: OC}\\ \text{head: no}\\ \text{split: post–hd}\\ \text{dist: 0}\end{bmatrix}} \quad \underset{\text{LPTree}}{\begin{bmatrix}\text{pos: VVINF}\\ \text{field: [pos:VVFIN, rel:root]4}\\ \text{intfield: [pos:VVINF]3}\end{bmatrix}}$$

Figure 5.3: Feature Structures, returned by the getFeatures() method of different subclasses of TreeNode

getIDTree() and getMobile() to the outside.

ClimbTree() adds a notion of climbing to IDTree. While each node in an IDTree has exactly one parent (or zero if it is the root), a ClimbTree node has always two parents, one local parent, which is inherited, and a potentially different non-local parent, where this node has climbed to. The edges to these non-local parents are ensured to be projective. ClimbTree instances are constructed by climb.ClimbingData.

An LPTree is the format used by the classes in the export module to represent a topological tree, calculated on the basis of the induced word order rules. An LPTree is guaranteed to be projective.

The abstract method FeatStruct TreeNode.getFeatures(boolean headFeature) provides a feature structure view on this node. IDTree's getFeatures method returns a feature structure containing all information found in the corpus. See Figure 5.3 for illustration. The feature structure includes two special features: "head" is the value of the boolean parameter. "split" is information that will be used only if splitting is enabled on the dependent class this feature structure falls into. ClimbTree's getFeatures method includes an additional feature "dist", which indicates the climbing distance from below of this node. LPTree's getFeatures method returns the field label of the field this node accepts, and the field label of the field the node itself is situated in.

Package learn.adapt This package implements a non-recursive feature structure, together with methods for feature selection, and automatic feature adaption. FeatStruct() implements a non-recursive feature structure as a hashtable from attributes to values, and offers flat feature structure unification, test for subsumption, and the calculation of the least upper bound in a collection of feature structures. Feature structures can be represented in human-readable format, or in a format appropriate for the decision tree learner. It is instances of FeatStruct that the methods in package tree return, when asked to describe a particular dependency node. Thus, a FeatStruct instance represents a "full" representation of a node in a dependency tree. This representation normally includes POS-tag, syntactic relation, and information on order with respect to the head, which is ignored unless splitting is enabled.

Interface FeatStructReducer implements the notion of a feature selection function from Chapter 4. It defines a method reduce(FeatStruct f, ReducedFeatStruct r) which creates a new ReducedFeatStruct r, when given a FeatStruct f. ReducedFeatStruct inherits from FeatStruct without significantly adding behaviour, and represents the notion of a "reduced" feature structure, as seen under the given

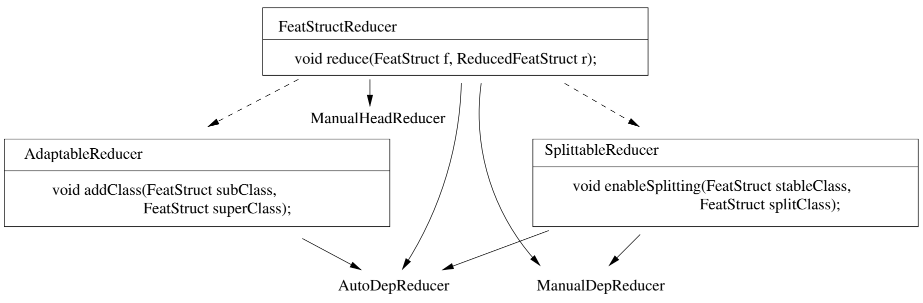

FeatStructReducer
void reduce(FeatStruct f, ReducedFeatStruct r);
ManualHeadReducer
AdaptableReducer
void addClass(FeatStruct subClass, FeatStruct superClass);
SplittableReducer
void enableSplitting(FeatStruct stableClass, FeatStruct splitClass);
AutoDepReducer
ManualDepReducer

Figure 5.4: Interface hierarchy of FeatStructReducers, and implementing classes

feature selection function. Figure 5.4 exemplifies the interface hierarchy and implementing classes of FeatStructReducer. ManualHeadReducer implements the head feature selection function from Chapter 4. ManualDepReducer implements manual dependent feature selection, with optional splitting, AutoDepReducer implements automatic dependent feature selection.

When method enableSplitting is invoked on an existing SplittableReducer, its behaviour changes to such an extend that the information whether the to-be-reduced feature structure occurs before or after another feature structure stableClass in the order instance, is preserved during reduction. stableClass needs to be [head:yes] in all implementations.

When method addClass is invoked on an existing AdaptableReducer, its behaviour changes such that a new feature structure is added to the range of the feature selection function. From now on, all feature structures subsumed by subClass are reduced to subClass. It was outlined in Chapter 4 that there need not be a unique subsuming feature structure, and that an order on feature structures is therefore necessary. This order is defined by the invocation order of consecutive addClass invocations, and the parameter superClass.

The complete learning system needs a single head feature selection function, each of which is associated with a dependent feature selection function. Class ReducerManager manages such a bundle of FeatStructReducer instances. It provides a static initial ReducerManager instance, built from fresh ManualHeadReducer or AutoDepReducer instances, depending on system options **adapt, split, uniqueHead, dectree**. A ReducerManager provides two methods ReducedFeatStruct reduceAsHead(TreeNode node) and reduceAsHead(TreeNode node, ReducedFeatStruct head), which call the node's getFeatures method , and reduce the returned feature structure with the appropriate FeatStructReducer. Figure 5.5 illustrates the behaviour of a ReducerManager.

There is a single instance of Adaptor(ReducerManager reducerManager), which is created from the initial ReducerManager instance. The task of the Adaptor is to iteratively adapt the ReducerManager by successive calls to its FeatStructReducer's addClass and enableSplitting methods. Adaptor defines a method boolean adapt(ReducedFeatStruct headClass, OrderData data), which will pick a precedence pair as outlined in Chapter 4, ask data for the order instances of headClass, and run the decision tree learner, resp. enable splitting. It will return true if converged, or false if further calls to adapt should be made, on the basis of the chosen convergence strategy (option **converge**).

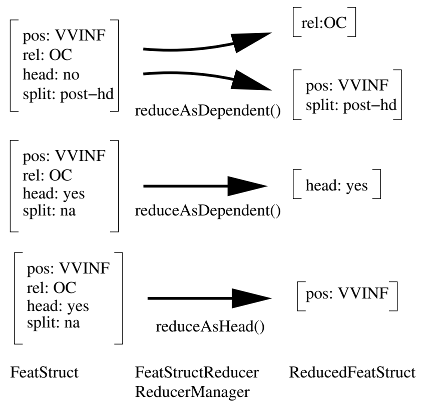


Figure 5.5: Feature Structures, reduced by different FeatStructReducers

The communication with the decision tree learner is implemented by class DecisionTreeLearner(String[] examples, String fileName, FeatStruct firstDep, FeatStruct secondDep, AdaptableReducer depReducer), which takes the chosen precedence pair, together with a string representation of this precedence pair's instances as arguments. Unsurprisingly, it also takes the AdaptableReducer to adapt, and a temporary file name for output. Internally, DecisionTreeLearner has private classes DecisionRules and DecisionRule for parsing C4.5 output, and adding the features referred to in these rules to depReducer. This task is implemented as discussed in Chapter 4, and relies on the test of subsumption provided by class FeatStruct.

Package learn.order This package implements precedence pairs and precedence tables. It defines a class OrderData(Corpus source, ClimbingData climbData, ReducerManager reducers), which offers to traverse the input corpus, convert the IDTrees it contains to ClimbTrees as predicted by climbData, and calculates a precedence table as defined in Chapter 4 for each head class assumed by the feature selection function reducers. OrderData offers a method PrecedenceTable getTable(ReducedFeatStruct head). A PrecedenceTable instance can enumerate its precedence pair types in different orders, and calculate conflict ratios. OrderData has also a method String[] getInstances(ReducedFeatStruct h, ReducedFeatStruct f1, ReducedFeatStruct f2), which retrieves the order instances of a precedence pair type from the corpus. Database retrieval is implemented via linear search through a temporary text file.

The package also provides classes OrderPair (a precedence pair) and OrderInstance. An OrderPair is able to return the smaller and larger of its elements according to the lexicographic order, and the first and second of its elements, if known.

Package learn.climb This class implements the conversion of IDTrees into ClimbTrees, and collects data on climbing at the same time. The interface is flexible enough to implement also more sophisticated climbing strategies than the one outlined in Section 4.4. The top-level class of the package is ClimbData, which collects data on all nodes in the entire corpus that had to climb. This data is collected in the form of ClimbList(ReducedFeatStruct head) instances, which describe all elements that climbed upward from this head class. This information may be used to predict where a node should climb in a specific case. The central method of ClimbingData is ClimbTree convert(IDTree source), which computes a ClimbTree from an IDTree. In the present implementation, the strategy outlined in Section 4.4 is implemented, which does not use the ClimbList instances that were collected.

Package learn.fields This package defines a class FieldData(OrderData data), which calculates a field representation for each precedence table in data, relying on the head and dependent feature selection functions implicit in the order data. For each head class, a field description, class Fields(PrecedenceTable table, EdgeSelector edgeSelector) is computed, which hides the notion of an order graph from Section 4.6. After computation, the Fields instances of the FieldData instance can be retrieved via Fields getFieldDescription(ReducedFeatStruct headAVM).

An EdgeSelector is an interface specifying function boolean conclusive(int prec, int succ) which selects or rejects a precedence pair depending on its precede and succeed count. Options **edgefactor** and **edgethresh** determine the behaviour of the implementing class.

For each precedence pair selected by the EdgeSelector, Fields creates two FieldNode(ReducedFeatStruct item, int id) instances and draws an FieldEdge-(FieldNode target) between them. FieldNode.setIndexLeast(int i) and Fields.solve() implement Algorithms 4 and 5 of Section 4.6. After calculation of the order graph, the field description can be retrieved by Iterator Fields.nodeIterator(), which returns a sequence of Field instances, each specifying a field index and a field label via Field.getIndex() and Field.getLabel().

Package learn.graphics This package implements the GUI of the system as a Java Swing user interface. It provides classes whose names end in "GUI", and mostly correspond to high-level classes of the architecture. FieldsGUI graphically displays an order graph, and allows to interactively change thresholds via InteractiveEdgeSelector. OrderDataGUI represents a set of precedence tables. The head class can be selected from a list, and the precedence table is displayed as a javax.swing.JTable. The user can switch to the order graph corresponding to the table. Clicking on a cell in the table results in the concrete corpus instances of the precedence pair being displayed in a new window, implemented by InstanceDataGUI. A further click on one of the instances opens a CorpusGUI, which shows the sentence in plain text. CorpusGUI also allows for browsing the entire input corpus, and viewing a tree representation of the current sentence. Displaying trees is implemented by tree.DisplayTree, and makes use of an external Oz module by Denys Duchier. CorpusGUI allows to display IDTrees, ClimbTrees and exported LPTrees.

The GUI is displayed after completed learning if the system option **display** is switched on.

Package learn.graph This is a helper package for graphics, which displays a graph on a javax.swing.JPanel.

Package learn.export The top-level class of this package is Exporter(FieldData, ClimbingData climbData, ReducerManager reducers), which implements the export strategies discussed in Section 4.8. An Exporter instance has full knowledge of a completed run of the learning system.

Method analyse(Corpus corpus, SystemOutput out) applies this knowledge to a test corpus (disjoint from the training corpus), and exports this input corpus to a file of of LPTrees. A bracketed representation of these trees is chosen as output format so that standard software can be applied to this data. This data is evaluated in Section 6.3. The conversion strategy is implemented as outlined in Algorithm 8, with class FieldAnalysis(TreeNode node, FieldData fieldData, ReducerManager reducers) implementing the matching of a sequence of dependents and a field description.

Method toTDG(Corpus corpus, SystemOutput out) converts the knowledge of the learning system to the XML interface format specified in Figure 4.20, and adds a TDG lexicon entry and a specification of the ID structures in the corpus, also in XML. This data is partially evaluated in Section 6.4.

Package learn This package provides a single executable class WordOrderExtractor, creating a single instance of WordOrderExtractor(SystemInput source) and running it with the global system options provided by learn.basics.Options.-OPTIONS. Options are read from the global option file, or parsed from the command line. Most of the high-level system options are queried in the implementation of WordOrderExtractor, especially file input/output options and options affecting the invocation or skipping of entire submodules.

The core method is run(), reproduced in Figure 9, which implements the control flow of Figure 4.1. System option **tracetrain** results in calling run() on increasing corpus sub-portions.

Algorithm 9 Code fragment from learn.WordOrderExtractor.java

```
public void run() {
  logger.info("Running system ...");
do {
  refreshData(); // clear reducers, climbData and orderData
  climb(); // call climbData.analyseCorpus(source)
  order(); // call orderData.analyseCorpus(source)
} while (!adapt()); // call adaptor.adapt(headClass, orderData) on all classes
of reducers
  fields(); // calculate field descriptions from orderData
  writeData(true,true); // write climbingData, orderData, fieldData to XML
  export(); // run Exporter on test corpus, and export XML
}
```

5.3 System Parameters

Submodule switches

adapt *true:* implied by dectree or split; causes iterated calculation of order data.

climb *true:* Precedence pairs are calculated from climb trees. During export, this means there are no fields offering positions for non-local elements. *false*: precedence pairs are calculated from ID trees, treating non-local realisations as if they were local.

dectree *true:* use decision tree learning for automatic feature adaption, starting with the classes defined by AutoDepReducer.java (or any other interface implementing AdaptableReducer).

split *true:* splitting, integrated with decision tree learning. If decision tree learning fails to find new classes on a chosen conflicting precedence pair, split this pair, and reiterate (very long running times).

uniqueHead *true:* splitting, to be executed after decision tree learning on all precedence pairs of every precedence table above edgefactor which conflict with the head (Head Splitting).

Thresholds

adaptthresh *(integer)* minimum total count of precedence pairs $(c(p)+c(p^{-1}))$ in order to invoke decision tree learning on them.

edgefactor *(0-100)* in order to include an edge into the order graph, maximum conflict ratio, as a percentage. In the implementation, the conflict ratio is defined as $\frac{min(c1,c2)}{max(c1,c2)}$, where $c1$ and $c2$ are the counts of the precedence pairs.

decthresh *(positive float ≤ 1)* minimum reliability of found decision tree rules (as provided by C4.5 output) in order to include them into the feature selection function.

edgethresh *(integer)* in order to include an edge into the order graph, minimum total count of the precedence pair $(c(p) + c(p^{-1}))$.

Controlling and Evaluating

display *true:* open interactive GUI after finishing calculations.

reconstruct *true: do* not create any output, and do not make any calculations, but read in an existing set of output files for displaying.

subsections *(integer)* 0 or 1 to disable. Otherwise the size of the first corpus subsection in sentences. See subseclg.

subseclg *(float)* 0 to disable: each corpus subsection has equal size, as defined by subsections. Otherwise, the base of the logarithm with which to grow corpus subsections: each corpus subsection is subseclg times larger than its predecessor.

tracetrain *true:* create snapshots of training after each corpus subsection, prefixing all output filenames with the iteration number, followed by a dot ("iteration prefix")

Chapter 6

Evaluation

In this chapter, I evaluate the implementation of the word order learning architecture on the German Negra corpus, using different parameter settings. Section 6.1 describes the parameter settings of the different models evaluated, and describes the different training and test sub-corpora used. Section 6.2 evaluates the system internally, by investigating its convergence behaviour. In Section 6.3, I indirectly and externally judge the learned word order rules by evaluating the topological corpus created on the basis of the word order rules. Section 6.4 presents a preliminary experiment which applies the rules to parsing and generation. Finally, I comment on the linguistic value of the acquired field descriptions in Section 6.5. I discuss the results in Section 6.6.

6.1 Experimental Setup

While the performance of a supervised learning system can usually be easily evaluated by comparing the produced output to the desired output (the Gold Standard), there is, by definition, no *a priori* gold standard in unsupervised learning. It was therefore decided to evaluate the implementation of the learning architecture in two ways:

- *internally:* the behaviour of the system can be evaluated in the terms of parameters which are part of the system.
- *externally:* the output of the system can be applied to some external task for which an evaluation strategy is at hand, and can be compared to the expected behaviour.

As for internal evaluation, a desired characteristic of a learning architecture is stability of output in the face of different data of the same kind. What is of particular interest is the size of the input corpus which is necessary to produce stable output. Thus, for internal evaluation, some parameters are necessary that describe the output of the system on an abstract and numeric level. The most important parameters of the learning architecture at hand that fulfil this requirement are:

- the size of the precedence tables produced

model	climb	dectree	splitting
BASE			
CLIMB	x		
MAN	x		x
DEC	x	x	
FULL	x	x	x

Table 6.1: Broad classification of system parameter settings

- the number of fields of the field descriptions produced

Section 6.2 will mainly investigate these parameters.

While internal evaluation can answer the question of whether the system produces any predictable output at all, it cannot assess the quality or usefulness of this output. The remaining sections are therefore concerned with external evaluation.

Section 4.8 described how the word order rules can be used to convert a test input treebank to a treebank of topological structures. It is this treebank which can serve as one source of external evaluation. I follow two strategies in Section 6.3.

- counting the frequency of unpredicted word order configurations during export (recall in *Rule Application Task*)
- comparing the produced treebank with a different topological corpus, created with a different method, using standard bracketing measures (bracket recall in *Corpus Comparison*)

I will also present an experiment on applying the learned word order rules to parsing and generation in Section 6.4. This is in some sense the most realistic scenario of external evaluation, since it is not only able to demonstrate the value of the learned rules, but also their behaviour as a sub-module in a wider framework. The results remain however preliminary.

Finally, I will judge the quality of the produced rules by comparison to linguistic theories, as the ones presented in Chapter 3. Section 6.5 will present some output of the system in detail, and comment on the linguistic value of the field descriptions.

The goal of evaluation is not only to judge the system as a whole, but also to compare different parameter settings of the system. In fact, the behaviour of a learning algorithm often varies stronger between different parameter settings than between different algorithms (Daelemans and Hoste (2002)). Therefore all of the experiments outlined above were therefore run with different parameter settings of the model. Section 5.3 gave a detailed overview of the implemented parameters. A set of parameter settings is called a *model.* The possible models can be broadly classified according to whether each of the system's main submodules is switched on or off, as illustrated in Table 6.1.

BASE models neither use climbing, automatic feature adaption, nor splitting, i.e. they do not account for non-local phenomena, and keep the dependent feature selection function unaltered. All other models use the climbing module to convert the input trees, and include non-local realisations in the field descriptions. MAN (manual) models differ from CLIMB models in their use of the splitting rule, while DEC (decision tree) models do not use splitting, but automatic feature adaption with decision tree learning. FULL models use both splitting and decision tree learning.

MAN and especially FULL models will prove to be the best performing ones, and will be compared in mode detail. They are also the models where a number of additional system options are available that can influence the behaviour of the system further. Table 6.2 is a comprehensive list of all models evaluated. An important option is robust edge selection (Section 4.7.1), which is applicable to all models, but will be evaluated in detail on manual models only. Six different edge selection strategies were used, varying in the threshold value used for including an edge into the order graph (edgethresh), and the maximum conflict ratio of a pair that is admissible for an edge to be included (edgefactor). Different edge selection strategies are indicated in the model name by a suffix E. For full models, the parameter that turned out to be most important was the rule confidence value provided by the decision tree learner (decthresh). Different strategies of automatic feature selection are indicated by a suffix D (for decision tree learning) at the model name.

Since the behaviour of the system on different input corpora is of interest, the corpus was split into subsections. The creation of sub-corpora was also necessary because some of the corpus data had to be held out for testing. All experiments were performed on version 2 of the 20k sentence Negra corpus of German newspaper texts. As test data, the very same set of test sentences was used as Becker and Frank (2002) used in their experiments. Becker and Frank used 500 sentences as a gold standard, viz. every tenth sentence in the range 10,000 to 15,000, with sentence indices ending in 7, excluding 6, presumably ill-formed, sentences.

In order to investigate the influence of corpus size on performance, each model was trained on the first corpus subsection, a snapshot was generated, the whole model was reset, and then retrained on sections 1 and 2, and so forth, for sections 1,2,3, then 1,2,3,4. Obviously, this leads to a runtime of $O(n^2)$ with the number of corpus subsections n. With model running times in the magnitude of hours for some parameter configurations, care should be taken to minimise the number of corpus subsections while guaranteeing precise convergence figures. Therefore, the corpus subsections were not chosen to be of equal size, but logarithmically growing: Each corpus subsection is 1.5 times larger than its predecessor. In the following, training corpus subsections will be numbered from 0 to 13, as indicated in Figure 6.1. Corpus subsection n starts at sentence $50 \cdot 1.5^{n-1} + 1$ and ends at sentence $50 \cdot 1.5^{n}$ (with the exception of section 0, which starts at sentence number 1). It is essential to note that all plots in the following sections distribute the corpus subsections evenly along the x-axis. This means that I plot on a logarithmic x-scale. Thus, the logarithmic growth of some parameters is easily visible. Training the models on sentences 1 to 9,730 proved to be a workable compromise between running times and enough data. Figure 6.1 sums up this discussion graphically.

Besides the influence of corpus size, it is also desirable to show that the system behaves similar on different input corpora of equal size. Dietterich (1998) proposes to split the training corpus into sections of equal size, and use the same test corpus, if enough training data is available. According to him, cross-

model	X-valid	trace	climb	dectree	adaptthresh	adaptfactor	decthresh	split	uniqueHead	edgethresh	edgefactor
BASE		14	n	n	-	-	-	n	n	1	100
BASE-E4		14	n	n	-	-	-	n	n	2	95
CLIMB		14	y	n	-	-	-	n	n	1	100
CLIMB-E4		14	y	n	-	-	-	n	n	2	95
DEC		14	y	y	10	95	0.50	n	n	1	100
DEC-E4		14	y	y	10	95	0.50	n	n	2	95
MAN-E1		14	y	n	-	-	-	n	y	1	100
MAN-E2		14	y	n	-	-	-	n	y	2	100
MAN-E3		14	y	n	-	-	-	n	y	1	95
MAN-E4	y	14	y	n	-	-	-	n	y	2	95
MAN-E5		14	y	n	-	-	-	n	y	1	50
MAN-E6		14	y	n	-	-	-	n	y	2	50
FULL-D1		14	y	y	10	95	0.50	n	y	2	95
FULL-D2	y	14	y	y	10	95	0.70	n	y	2	95
FULL-D3		14	y	y	10	95	0.80	n	y	3	95

Table 6.2: Parameter settings of all models evaluated. X2 models are trained on different, but equally sized corpus data for cross validation.

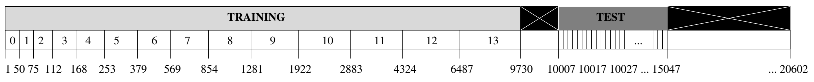


Figure 6.1: Corpus Subsections

BASE	CLIMB	DEC	MAN	FULL
14min	60min	1d 21h	5h 45min	1d, 19h

Table 6.3: Running times for different models, with training trace

validation Stone (1977) is not necessary in this case. For this experiment, the corpus was split into 9 subsections of 2010 sentences each. The test corpus was excluded.

Figure 6.3 shows the running times of the different models on a cluster machine with two 1266 MHz Pentium III processors, with training trace option set to true. Some of the models were trained in parallel. Obviously, decision tree learning with its feed-back architecture is expensive, while the other models can be trained within hours on 10,000 sentences.

6.2 Convergence Behaviour

Two parameters were mentioned in the previous section as possible measures of system convergence: First, *Precedence Table Size* is the number of elements in a precedence table. In the following, [pos:NN], [pos:NE], [pos:VAFIN,rel:±root] and [pos:VVFIN,rel:±root] are chosen as representative precedence tables, with nouns exhibiting significantly less non-local dependents than verbs. Second, *Field Description Size* is the number of fields in a field description. The field descriptions constructed from the precedence tables of the same heads were selected and will be plotted against the corpus size in this section.

The plots in Figure 6.2 show that the growth of precedence tables with increasing corpus size dependents heavily on the model used. Reminding the reader that the x-scale is logarithmic, it is obvious that precedence table size converges for the model BASE after about 3,000 sentences at 30 elements for nouns and between 20 and 25 for verbs. This is not surprising, since the feature selection strategy of the BASE models restricts the number of possible elements to the number of syntactic relations. Models that allow climbing though, exhibit logarithmic growth of their precedence tables, visible in the plots by the linear appearance of the graph. Here, unseen climbed elements can enter the field descriptions. Precedence tables of DEC and MAN models grow faster, since they also allow for elements to be automatically sub-classified respectively split.

The plots in Figure 6.3 show the behaviour of different manual models when constructing field descriptions from the precedence tables underlying Figure 6.2(d). Models MAN-E1 and MAN-E5 illustrate two extreme cases of what could be termed non-robust edge selection: MAN-E1 selects only non-conflicting precedence pairs (edgefactor=100), while MAN-E5 selects all precedence pairs, and infers the directionality of an edge from the majority of cases in the data (edgefactor=50). Models on the left hand side in the figure do not employ a

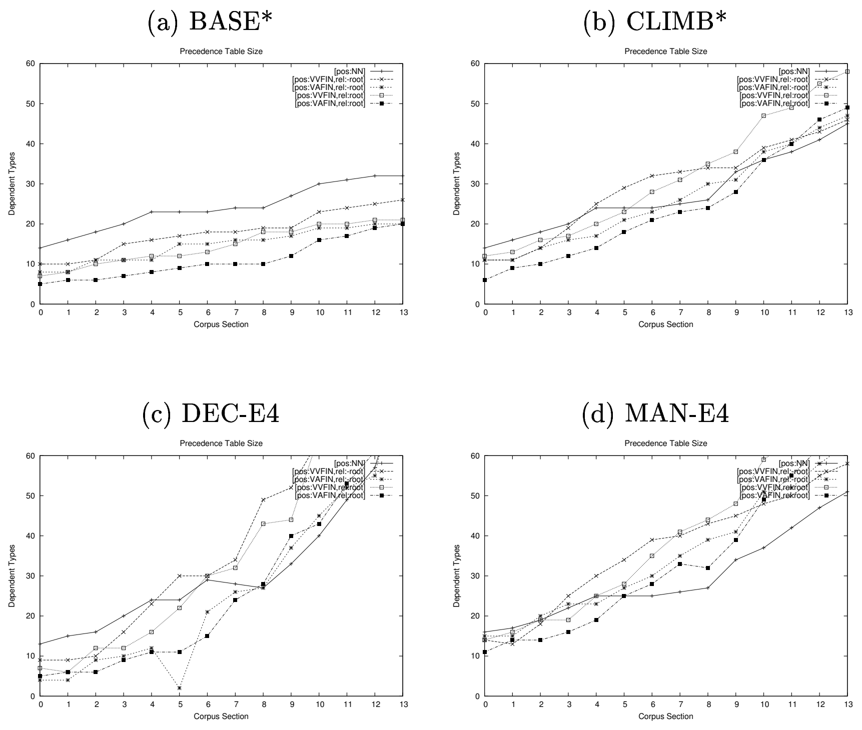


Figure 6.2: Growth of Precedence Table Size with Corpus Size for different models

frequency threshold in including edges into the graph. Models on the right hand side, in contrast, employ frequency threshold 2. The parameter edgefactor decreases from 100 for models in the top row to 95 for models in the second row to 50 for models in the third row.

The plots make clear that the introduction of a threshold is an essential characteristic in order to make the field descriptions robust: field description size is volatile for models on the left hand side, while it varies only slightly for models in the right column. The figures also suggest that an edgefactor of slightly less than 100 is a good choice for robust edge selection. This makes MAN-E4 the best manual model.

The growth of field descriptions seems to be logarithmic too, in the plot range, which is more surprising than for precedence tables. Field Description Size grows much slower than Precedence Table Size, though.

Figure 6.4 plots the same data as the previous figure for the remaining models. Only E1 and E4 models where computed for the BASE, CLIMB and DEC strategies. Field Description Size clearly converges for the BASE models, where also Precedence Table Size converged. For DEC strategies, the size of field descriptions is volatile, both for robust and non-robust edge selection.

Up to now, FULL models have not yet been considered, because their behaviour depends on additional parameters besides robust edge selection. I assume at this point that robust edge selection has proved successful, and only evaluate models with edgethresh=2 and edgefactor=95.

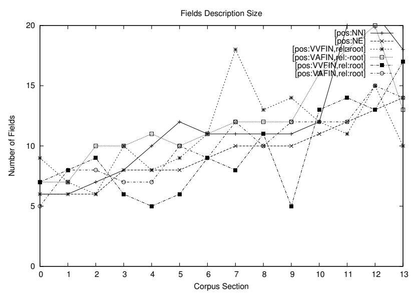
Figure 6.5 plots Precedence Table Growth in FULL models, comparing the parameter *decthresh*, which influences the inclusion of rules found by the decision tree learner into the feature selection strategy. FULL-D1 includes every rule found by the learner (decthresh=0.5, a rule has at least a score of 50%). FULL-D2 only includes rules with an error rate of less than 30% (decthresh=0.7). Again, precedence tables seem to grow logarithmically, but faster for FULL-D1.

Figure 6.6 plots the growth of the corresponding field descriptions. A higher decthresh value smooths the growth of field descriptions, but only slightly. Also with a higher decthresh value, decision tree learning still behaves in a more volatile manner than the corresponding manual model, and shows more growth. The reason is that fine-grained classes discovered by the decision tree learner may be collapsed to single fields on some data due to cycles in the order graph, while thresholds prevent this collapsing on slightly different data.

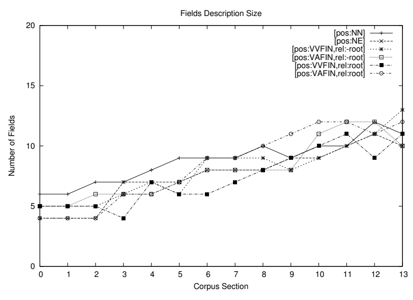
Figure 6.7 investigates the stability of field description size closer. The models MAN-E4 and FULL-D2 were trained on 9 disjoint corpus subsections of equal size (2010 sentences). In both models, there is considerable variance in the field sizes. For MAN-E4, field sizes vary by $\pm 16.5\%$ in average, for FULL-D2 by $\pm 19.4\%$. In the manual model, nominal fields vary considerably less than verbal fields ($\pm 12.6\%$ as opposed to $\pm 18.3\%$). The trend is inverted, and less clear for the automatic model ($\pm 21.2\%$ and $\pm 18.5\%$).

More parameters would be desirable to test for automatic feature adaption. However, running many automatic feature adaption models with training trace is quite time consuming. Therefore restrictions had to be applied to these experiments for the course of this thesis.

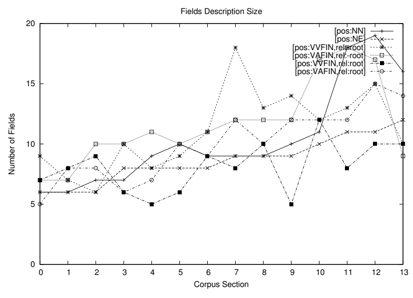
(a) MAN-E1



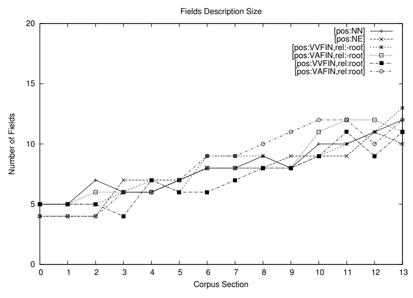
(b) MAN-E2



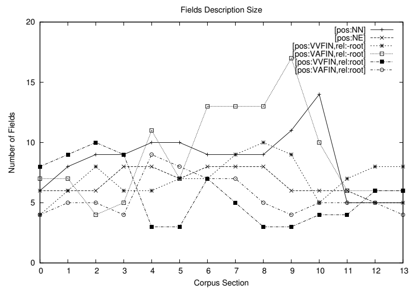
(c) MAN-E3



(d) MAN-E4



(e) MAN-E5



(f) MAN-E6

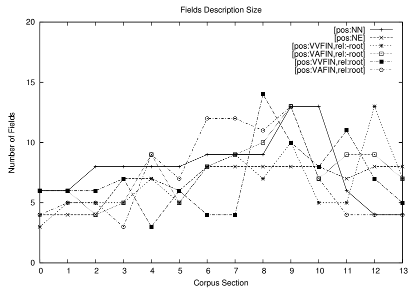
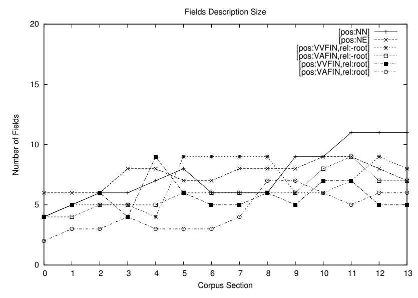
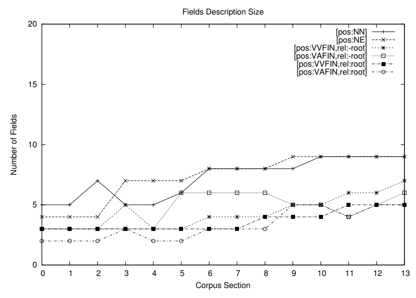


Figure 6.3: Growth of Field Description Size with Corpus Size for models MAN-E1 to MAN-E6. Models with uneven numbers have edge selection threshold 1, models with even numbers 2. Edge selection factor is 100% for models 1 and 2, and 50% for models 5 and 6.

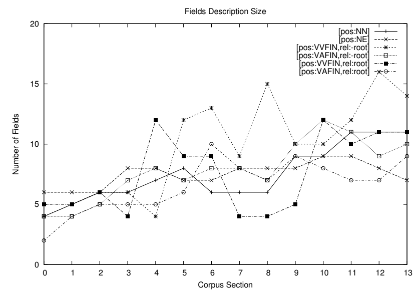
(a) BASE



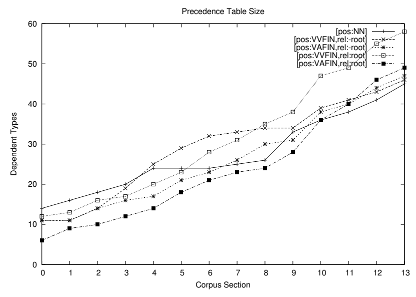
(b) BASE-E4



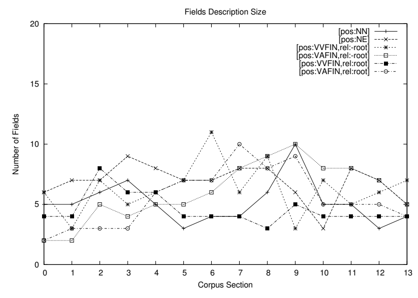
(c) CLIMB



(d) CLIMB-E4



(e) DEC



(f) DEC-E4

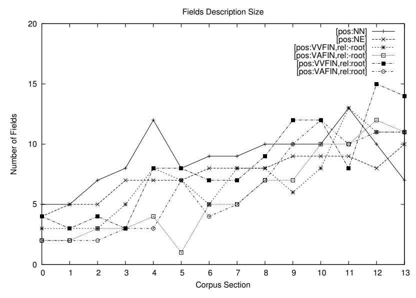


Figure 6.4: Growth of Field Description Size with Corpus Size for different models

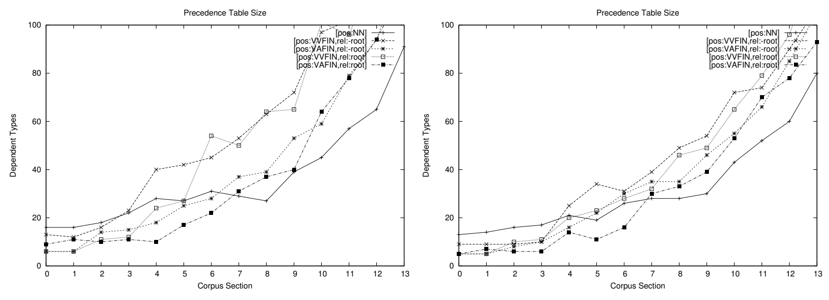


Figure 6.5: Growth of Table Size with Corpus Size for models FULL-D1 and FULL-D2

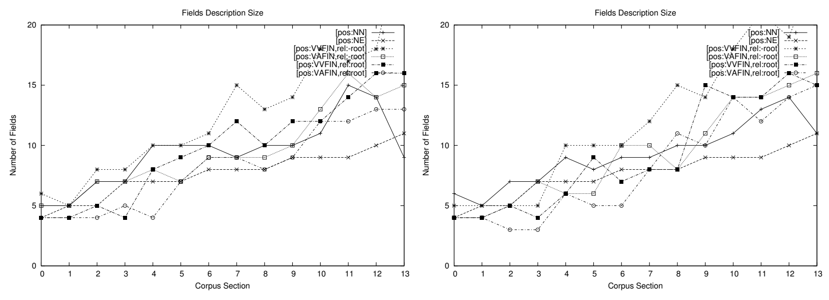


Figure 6.6: Growth of Field Size with Corpus Size for models FULL-D1 and FULL-D2

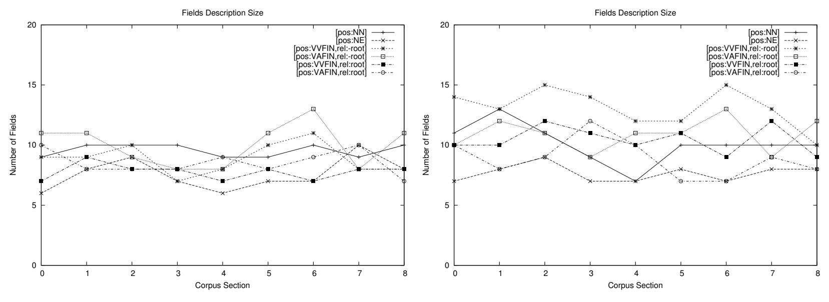


Figure 6.7: Precedence Table size of models MAN-E4-xval and FULL-D2-xval, each trained on disjoint training sets of 2010 each

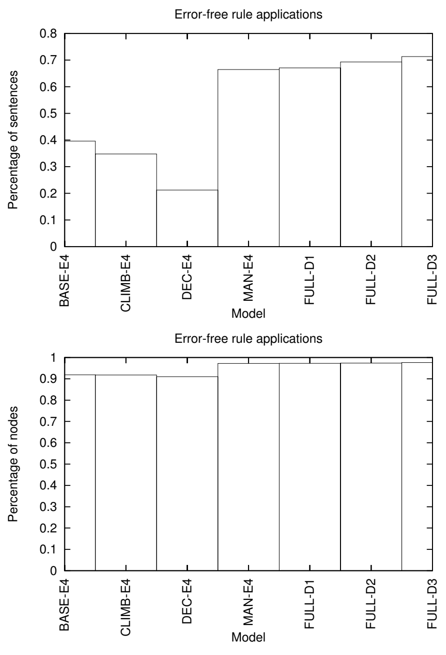


Figure 6.8: Different models' performance on Rule Application Task

6.3 Rule Recall

This section presents two experiments which both use the Export module (Section 4.8) to generate a corpus of topological field structures conforming to the learned word order rules and evaluate this corpus. This means the rules are evaluated indirectly against an external criterion of success.

In the first experiment, the number of failed rule applications during export was counted. A rule application fails if the word order configuration observed under a given head in the test corpus is unattested by the learned feature selection function and field description (see Section 4.8).

Two recall counts were generated from this data, the number of successful nodes (node recall), and the number of successful sentences (sentence recall)

- a node is considered successful if the order of all of its dependents is attested by the field description, and unsuccessful otherwise.
- a sentence is considered successful if it contains successful nodes only

There is a choice of whether to count trivial field descriptions which consist of a head only, and no dependents as a successful rule application, or exclude them from the calculation. Unless otherwise stated, trivial nodes are counted as successful rule applications. As a matter of fact, the difference is only marginal.

Figure 6.8 compares the different models with regard to their performance during the export task. It is clear from these figures that the most significant improvement in the results is due to the splitting rule. Models with splitting (MAN, FULL) outperform the others clearly. Model FULL-D3 achieves 71.3%

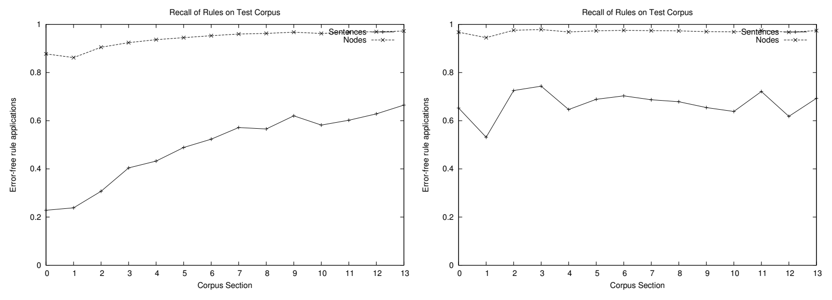


Figure 6.9: Influence of Corpus Size on Rule Application Task performance for models MAN-E4 and FULL-D2

recall on sentences and 97.6% recall on nodes, considerably outperforming MAN-E4 with 66.5% node recall and 97.2% node recall.

Figure 6.9 demonstrates the influence of corpus size on the export task for manual and automatic feature selection. While manual feature selection improves with growing corpus size, and outperforms automatic feature selection towards the end, the performance of automatic feature selection is nearly independent of corpus size. This is due to the behaviour of the integrated decision tree learning/splitting architecture of the FULL model: in the face of decision tree learning failing, there will be only very few, but very general classes, which will be split due to a high conflict ratio. This trivial field description performs quite well on the export task, and is superior to manual feature selection, as long as little data has been seen.

Figure 6.10 presents the same node recall data as in the previous figure, but sub-classified by the head class of the node. For this calculation, trivial nodes were excluded. Performance is consistently lower on verbal than on non-verbal material. The automatic model outperforms the manual model clearly for root-node finite auxiliary verbs. The score for [pos:KOUS] is 100% for the automatic model, since it does not find any rules there, and provides a trivial field description consisting of [split:pre-hd], [head:yes], [split:post-hd]. FULL is inferior to MAN for [pos:VVFIN,rel:root] and [pos:VVINF].

The previous experiment presented results on how reliable the learned rules are for assigning a topological field structure to a given word order configuration. It did not show, however, how closely the rules match with linguistic intuitions. The experiment was therefore complemented by a comparison of the produced topological corpus with the hand-crafted topological corpus of German created by Becker and Frank (2002).

The gold standard used in Becker and Frank (2002) for evaluation of their parser was kindly made available to the author by Anette Frank. The corpus was represented as Penn Treebank style bracket structures. The learned LP corpus was available in this format, too, on the basis of the algorithm sketched in Section 4.9. Figure 4.19 above presented an example. For the purpose of comparison, a dummy node in Becker and Frank's data encoding the sentence number and punctuation was removed.

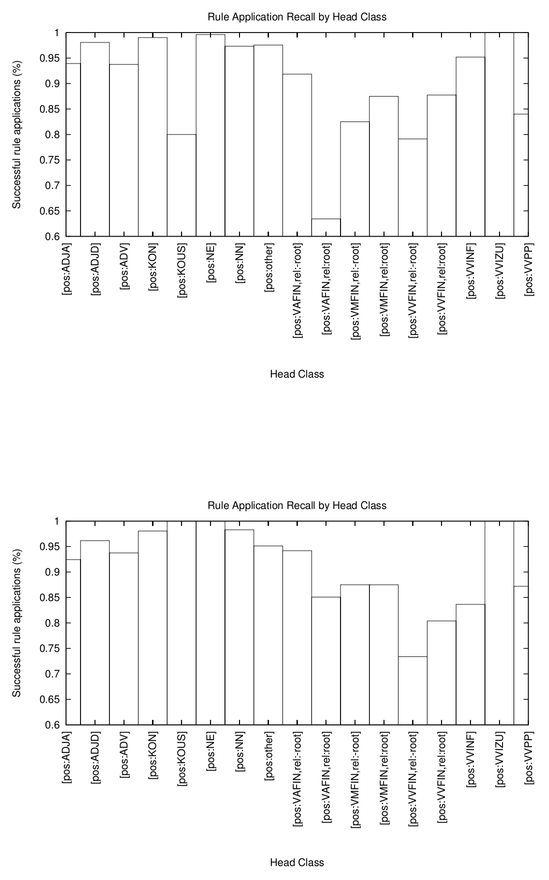


Figure 6.10: Recall in Rule Application Task by head class, for models MAN-E4 and FULL-D2, under exclusion of trivial nodes

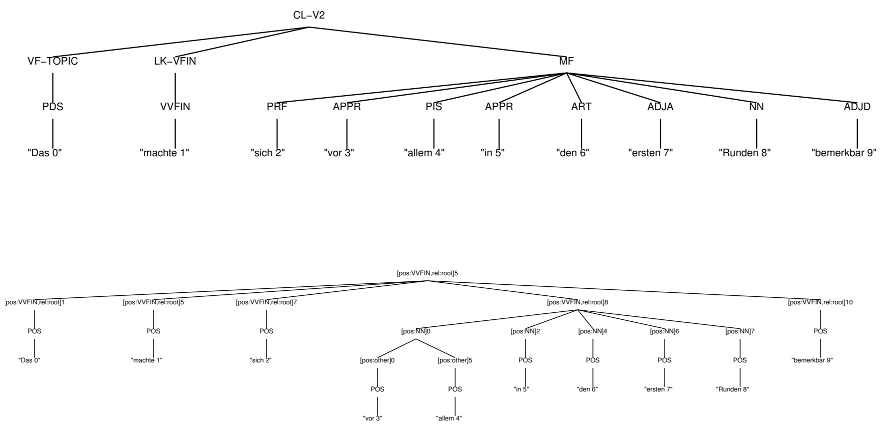


Figure 6.11: Comparison of (above) gold standard LP tree and (below) converted learned tree, with 3 of 4 brackets matching (75%), and 0 crossing brackets (Negra sentence 10097)

Bracket structures are normally compared with the PARSEVAL measures (see Collins (1997)). For reasons of comparability, the same tool was used for calculations of these figure as Becker and Frank were using, EVALB.

There are practical problems in applying these measures to the data at hand. Obviously, the tags of the trees to be compared differ. Becker and Frank use a linguistically motivated tag set, while the learning system produces numeric field labels. Therefore only unlabelled bracket measures could be applied. Another issue is whether trivial brackets should be counted. EVALB was configured to ignore all POS-tags. It cannot be configured to ignore the sentence-spanning root node though, neither can it be configured to ignore brackets with the same span. Brackets with the same span are extremely seldom in the data though, due to the very flat tree structures.

There is another and more severe problem with comparing the learned structures with Becker and Frank's data: Becker and Frank use a very flat annotation for noun phrase topology and middlefield configurations, while the learning architecture was designed to discover word order regularities of finer granularity. An example is the own field for the reflexive pronoun "sich" in Figure 6.11 (confer Figures 4.17 or 4.19, page 64, for alternative representations of the same sentence), which has to occur before other middlefield material; a fact that is not reflected in the gold standard data. Certainly the system should not be penalised for these fine-grained analyses. The analyses of "bemerkbar" also differ.

There are two PARSEVAL measures that can be employed in the light of the above considerations: unlabelled bracket recall, and non-crossing bracket measures. Figure 6.12 plots these measures for the different models tested. FULL-D2 achieves 74.5% bracket recall, and MAN-E4 73.8%. When we compare these figures to the 97.6% and 97.2% recall on nodes in the Rule Application Task, it is obvious that the Corpus Comparison figures are much lower.

There are two main reasons for the lower results. The Corpus Comparison

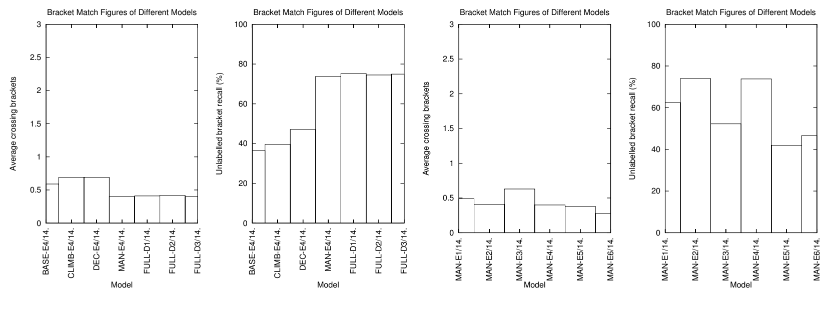


Figure 6.12: Different models' performance on comparison with Becker and Frank

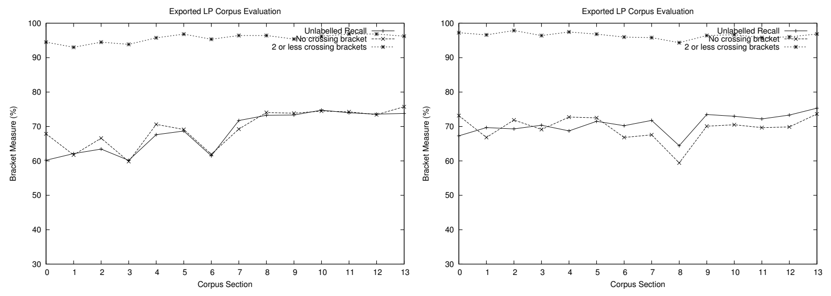


Figure 6.13: Influence of Corpus Size on performance on comparison with Becker and Frank for models MAN-E4 and FULL-D2

task penalises for different climbing decisions; sometimes it also penalises for assuming a more fine grained analysis than in the Gold Standard. Manual comparison of the exported corpus with the Gold Standard revealed that in fact many of the differences are due to differing, but both valid linguistic analyses. The learning system consistently prefers local attachment of extraposable material over non-local configurations in cases where the two are indistinguishable, while Becker and Frank opt for the alternative analysis. This applies to relative and complement clauses. It also applies to arguments of embedded verbal material, which is only predicted to be a topological dependent of the main verb in the face of scrambling by the system, but consistently "climbs" in Becker and Frank's data. There is also an own field for verb infinitive "zu" markers in the learning system, but not in Becker and Frank's data.[1]

Figure 6.13 shows the influence of the training corpus size on the bracket measures. Here, a similar picture as in Figure 6.9 is revealed, though less clearly: corpus size seems to matter in MAN models, but not in FULL models.

[1]EVALB does not offer the possibility to calculate recall figures for different types of constituents. Therefore a quantitative sub-classification was not possible.

6.4 Application to TDG Parsing and Generation

All experiments of Sections 6.2 and 6.3 tested only whether the learned rules are able to account for the word order *variation* found in a test corpus. In other words, they interpreted the learned rules as *parsing* rules, and evaluated whether the rules undergenerate in terms of recall. The experiments did not test whether the rules capture the *constraints* on word order variability found in the data well. This section presents a preliminary experiment that tries to answer the question whether the learned rules overgenerate when used for *generation*. At the same time, the experiment demonstrates how the rules can be used in a more realistic application than the export of an LP corpus. This eventual scenario is the embedding of the word order learning system into an existing grammar framework, and work towards learning submodules of this grammar framework.

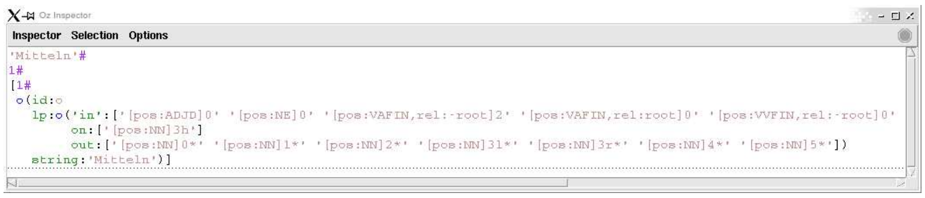
Section 4.8, Figure 4.20 presented an example of the XML format into which the learned rules are exported. In fact this format shows significant resemblance with some sections of the TDG grammar formalism presented earlier in Section 3.3, Figure 3.3, and was converted into a TDG grammar. For each `depclass` in the XML file, a corresponding lexical type was created that specified an `valencyLP` slot for each `offers` element. The same was done for `headclass` and the `edgeID` statement. The field labels offered by the tag `<depclass id="[head:yes]">` was converted into the set of internal field labels, and it was specified that each head class accepts its internal field label solely as an internal field.

The integrated TDG parser/generator had to be configured such that it accepted not only a sequence of word forms as input, but a sequence of word forms together with a given ID analysis. For this purpose, the ID structures of every sentence in the corpus were also converted into an XML representation and fed into the parser with the aid of an additional TDG principle (preparsed principle), which simply assumed the ID analysis as given. During parsing, a lexical entry was then assembled for each word form in the sentence by combining the preparsed ID features with LP features constructed by lexical inheritance. Each word form inherited from two LP lexical types, its head class and a set of possible dependent classes.

There is a set of possible dependent classes for each lexical item, because the same set of feature structures can be reduced to different "reduced feature structures" (lexical types) by different heads, and the eventual LP head of an item is not known in advance. This caused lexical ambiguity on the LP level. For simplification, each lexical entry was reduced by all reducers of transitive ID heads. Figure 6.14 shows an example of an automatically assembled lexical entry.

There are two possible TDG features which were not inferred from the learning system. Field cardinality constraints were not available at the time the experiment was run. For the course of the experiment, "*" (0, 1, or more occurrences) was assumed for all LP fields, the least constraining strategy possible. The second feature which was not learned is the *blocks* feature. Here, two strategies were followed and compared: First, blocks was always set to the empty set, which allows unrestricted climbing through all heads. Second, a feature was manually added to the lexical types for nouns, which blocked all dependents.

The experiment was run before the completion of implementation, with an



```
'Mitteln'#
1#
[1#
 o(id:o
   lp:o('in':['[pos:ADJD]0' '[pos:NE]0' '[pos:VAFIN,rel:-root]2' '[pos:VAFIN,rel:root]0' '[pos:VVFIN,rel:-root]0'
         on:['[pos:NN]3h']
         out:['[pos:NN]0*' '[pos:NN]1*' '[pos:NN]2*' '[pos:NN]31*' '[pos:NN]3r*' '[pos:NN]4*' '[pos:NN]5*'])
   string:'Mitteln')]
```

Figure 6.14: Learned LP level of a lexicon entry for "Mitteln", offering 5 nominal fields, where field number 3 identifies the position of the head in its own field, and accepting adjectives, proper names and finite auxiliary and main verbs as LP governors

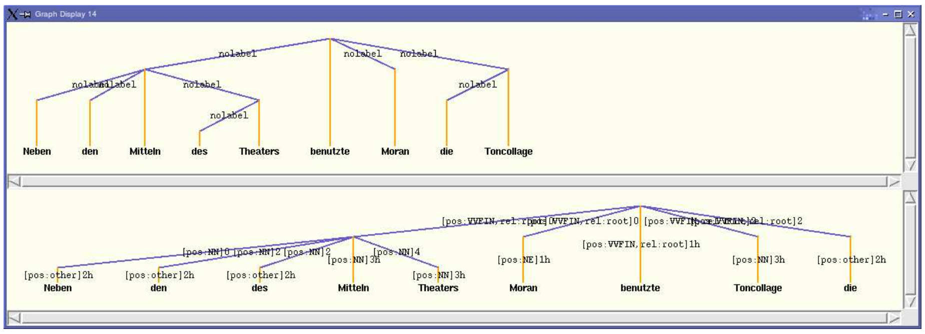
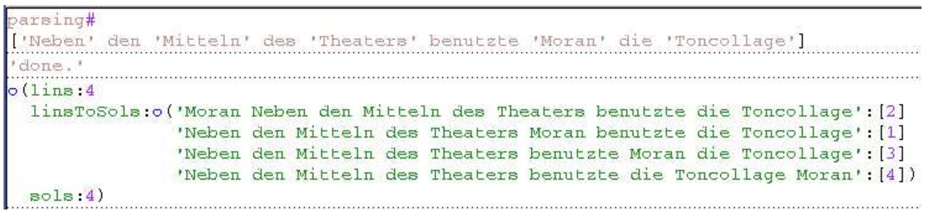


Figure 6.15: Wrong linearisation of a sentence, caused by non-prevented climbing of an article through an NP barrier, and the existence of an "other"-class at the verbal landing site

earlier version of the learning architecture, and with automatic feature adaption, trained on 1000 sentences only. The partially learned grammar was successfully interpreted by the parser/generator, and applied to selected sentences. Parsing and generation times were too high for a complete run over a test corpus, but some of the shorter sentences could be parsed within seconds. In parsing mode, all admissible LP analyses of a given, ordered ID analysis were computed. In generation mode, all admissible word permutations were computed, and their LP analyses were investigated.

Without the restriction on the *blocks* feature, many ungrammatical linearisations, as the one in Figure 6.15, were generated. In the example two articles, "des" and "die", climbed through their ID heads, and landed in fields of transitive heads. This was admitted by the grammar, because field descriptions include a class "other", created by the decision tree learner, which allows nearly all kinds of dependents. With the manually added *blocks* feature for nouns, though, the admissible linearisations of the same sentences generated by the grammar reduced to 4, out of a theoretically possible $9! = 362,880$ permutations of the input sentence. The generated linearisations are shown in Figure 6.16. The figure still includes two ungrammatical linearisations (number 1 and 2). These two instances are due to the missing cardinality constraints, which were not available at the time of the running of the experiment.



```
parsing#
['Neben' den 'Mitteln' des 'Theaters' benutzte 'Moran' die 'Toncollage']
'done.'
o(lins:4
  linsToSols:o('Moran Neben den Mitteln des Theaters benutzte die Toncollage':[2]
               'Neben den Mitteln des Theaters Moran benutzte die Toncollage':[1]
               'Neben den Mitteln des Theaters benutzte Moran die Toncollage':[3]
               'Neben den Mitteln des Theaters benutzte die Toncollage Moran':[4])
  sols:4)
```

Figure 6.16: All possible linearisations generated for the given ID structure of the sentence "Neben den Mitteln des Theaters benutzte Moran die Toncollage", with manually added NP-blocking. The two ungrammatical linearisations are due to missing cardinality constraints.

The early experiments with generation showed that not only parsing, but also generation with the learned grammar is in fact possible. Preliminary results suggest that not only variability found in word order, but also constraints are captured well by the learning system. Adaptations to the parser are necessary, though, to increase efficiency, before a quantitative evaluation experiment can be run.

It became clear in the experiment that the inclusion of field cardinality constraints and blocking constraints is a necessary step in order to avoid gross overgeneration. The computation of field cardinality constraints was outlined in Section 4.8, and seems to pose little problems. Automatic learning of *block* features would require more research into the climbing module. Some thoughts in this direction were indicated in Section 4.4, but learning constraints similar to TDG's *blocks* feature from the climbing configurations in the corpus on climbing paths is not an easy issue. A more promising short-term solution might be to exclude the *other* classes produced by the decision tree learner. This will not remove all cases of ungrammatical climbing, but probably significantly reduce the amount.

6.5 Linguistic Judgement

This section judges the field descriptions learned by the system on a linguistic basic. The appendices C.1 and C.2 present the complete field descriptions for all head classes, acquired by the models MAN-E4 and FULL-D2. For MAN-E4 the results from corpus section 14 were taken, run on sentences 1-9730; for FULL-D2 the results from corpus section 13 were taken, run on sentences 1-6487, because section 14 caused instability.[2] The appendices do not include precedence tables. For an explanation of corpus tags used, cf. Appendix A.

Some general remarks are necessary on reading the field descriptions. All models were run with a static head feature selection function which distinguished the most frequently occurring POS-tags of items that take dependents at all. Less frequent POS-tags were conflated into a class [pos:other]. A distinction between verbs at the root node of the tree, and verbs in subordinate structures was established, hoping to find differences with regard to verbal head position.

[2]At least this was the case at the point of time where the evaluation took place.

This is only a very coarse classification, though, and not suitable to describe the basic division of V1, V2, and VF clauses.

While element feature structures of manual models are guaranteed to be disjoint, the elements created with decision tree learning may subsume each other. It is only through the order on feature structures, theoretically introduced in Sections 2.4 and 4.7.2 that these field descriptions can be interpreted. The reader is reminded at this point that a more specific feature structure is always preferred to a less specific one, an element subsumed by, e.g. [pos:PRELS,rel:SB] and [rel:SB] will be reduced to the first feature structure. If there is also an element [pos:PRELS] in the field description, it will normally be reduced to [rel:SB], because the decision tree learner normally identifies syntactic relation as the most significant feature, which leads to preferring *rel* over *pos* in the global order on feature structures. In automatically adapted field descriptions, empty feature structures or empty split feature structures can be discovered. These are "dustbin"-categories, applicable only if no other feature structure subsumes an element.

In the following, the field descriptions of nouns [pos:NN], finite root-node auxiliary verbs [pos:VAFIN,rel:root], and finite non-root node main verbs [pos:-VVFIN, rel:-root] will be described in some detail, and differences between the models will be outlined. After that, the descriptions are linguistically evaluated. Model FULL-D2 consistently produces slightly larger field descriptions, with considerably more elements, and a higher percentage of split elements.

Normal nouns [pos:NN] have 12 fields, and contain 39 elements, 20% of which bear the split feature for model MAN-E4 (see page 110). For model FULL-D1 (see page 117), there are 14 fields and 59 elements, 30% of which are split. The head element occurs in field 5 for MAN-E4, and in field 7 for FULL-D2. Both models discovered that prepositions [pos:AC] are consistently at the beginning, and relative and complement clauses [rel:RC], [rel:OC] at the end. Genitives [pro:GL] are predicted to occur either before adjectival pre-modifiers, or closely after the head [pos:GR]. Since manual feature selection considered syntactic relation only, and determiners, adjectives and other pre-modifiers of nouns all bear the label NK (noun kernel) in Negra, MAN-E4 could not discover order rules with respect to determiners and adjectives. Automatic feature adaption however, picked [rel:NK] as an order relevant feature. This class occurred as a self-conflicting precedence pair in a later iteration, and was further split up into determiners, [pos:ART, rel:NK], which consistently occur before demonstrative pronouns [pos:PIDAT, rel:NK], adjectives [pos:ADJA, rel:NK] and other kinds of noun-kernel elements. MAN-E4 found some non-local elements, e.g. climbed appositions [dist:1, rel:MNR] and modifiers [dist:2, rel:MO], while in the field description of automatic feature adaption the feature "dist" does not occur. This does not mean that there are no rules for non-local elements, it merely means that there are no classes of elements discernible by their "dist" feature only, which behave differently. This is probably due to the fact that search started with the classes [] and [head:yes]. In both models, conjoints conjoined with the head noun are predicted to occur directly after the head, a characteristic which follows from the treatment of coordination during input tree conversion. Table 6.4 provides an alignment of the automatically acquired fields with the analysis of noun phrase topology proposed by Eisenberg (1999).

Eisenberg	-			ART		ADJ	SBST	Gen	PrGr	S
MAN-E4	0	1	2	3-4			5	6-7	8-10	11-12
example	[rel:NG]	[rel:MO]	[rel:AC]	[rel:NK]			[head:yes]	[rel:GR]		[rel:RC]
FULL-D2	0	1	2	4	5	6	7	8	9-11	12-13
example	{rel:NG]	[rel:MO]	[rel:AC]	[pos:ART]	[pos:PIDAT]	[pos:ADJA]	[head:yes]	[rel:GR]		[rel:RC]

Table 6.4: Comparison of traditional NP topological analyses with learned field descriptions

Eisenberg lists details about order within the adjectival fields, mainly determined by semantic factors the corpus is not annotated for. It seems possible that automatic feature adaption could discover these regularities, would it be provided with data that encodes adjective classes. Vice versa, both automatic models discovered regularities of pre-modification of PPs in fields 0 to 2, which Eisenberg does not mention.

Finite, root-node auxiliary verbs [pos:VAFIN, rel:root] have 12 fields with 53 elements, 64% of which are split for MAN-E4 (page 115), and 14 fields with 77 elements, 72% of which are split for FULL-D2 (page 125). 36 respectively 10 elements explicitly refer to non-local elements. Finite auxiliary verbs at the root of the sentence occur in V2 position in most cases, since newspaper sentences are mostly declarative. The learned field description can therefore be compared to traditional topological analyses of V2 clauses: The left sentence bracket corresponds to the field where the head occurs. This is field 3 for model MAN-E4 and field 4 for FULL-D2. The right sentence bracket corresponds to a field where non-finite verbal complements can occur. Such a field was also identified in both models. It is field number 6 for MAN-E4, where [rel:OC] occurs, and field number 7 for FULL-D2 ([dist:0,rel:OC]).[3]

Both models also identified junctors [rel:JU] as occurring at the very beginning of a verbal field description only. Table 6.5 summarises these similarities, comparing to the analyses of Höhle and Eisenberg.

With aid of the interactive GUI, it is easily possible to extract example sentences for each of the elements in the field descriptions. Below, corpus examples of selected elements from the model FULL-D2 are provided, together with the field in which they can occur and a reference to the sentence number in the Negra corpus.

[3]When identifying [dist:0,rel:OC] as the right sentence bracket, the definition of the domain of the feature selection function as a lattice of feature structures, with an additional order (Section 4.7.2) plays a crucial role: FULL-D2 contains an element [pos:VVPP,split:post-hd] in field 10. Normally, elements subsumed by [rel:OC] are also subsumed by [pos:VVPP]. Due to the order on the feature structures in the lattice, however, elements are reduced to [pos:VVPP,split:post-hd] only if they are *not* subsumed by [rel:OC]. An example of [pos:VVPP,split:post-hd] is given shortly in the course of the discussion. A similar situation arises in MAN-E4.

Eisenberg	Konj		Vorfeld		Fin
Höhle	KOORD	C	K_L	K	FINIT
MAN-E4	0		0-2		3
example	[pos:JU}		[rel:SB]		[head:yes]
FULL-D2	0		0-3		4
example	[pos:JU]		[pos:PPER,split:pre-hd]		[head:yes]
			[dist:1,pos:NN,rel:OA,split:pre-hd]		

Mittelfeld	Infiniter VK	Nachfeld
X	VK	Y
4-5	6	7-11
[rel:MO]	[rel:OC]	[rel:RC]
5-6	7	8-13
[pos:PPER,split:post-hd]	[rel:OC]	[dist:1,pos:NN,rel:OA,split:post-hd]
		[rel:RE,split:post-hd]

Table 6.5: Comparison of traditional topological analyses with learned field descriptions

1. [pos:PPER,split:post-hd] (field 5, frequency 247): Von dort aus wurde [es] zum Schornstein hochverlegt und von da an die Kabel drangehängt, die ich glücklicherweise schon im Haus hatte. (873)

2. [pos:PPER,split:pre-hd] (field 0, frequency 154): [Er] hat bereits ganze drei neue Opern fertiggestellt. (47)

3. [dist:1,pos:NN,rel:OA,split:pre-hd] (field 0): [Den Traum von der kleinen Gaststätte] hat er noch nicht aufgegeben (551)

4. [dist:1,pos:NN,rel:OA,split:post-hd] (field 8): Angekündigt hatte das Unternehmen bereits [den Abbau von 3000 Stellen in diesem Jahr] (853).

5. [pos:VVPP,split:post-hd] (field 10): Nach ihm ist das Hochheimer Heimatmuseum [benannt]. (222)

6. [rel:RE,split:post-hd] (field 12): Nach Angaben des US-Rechnungshofes würde es allein 300 bis 400 Milliarden Dollar kosten, [die militärischen Atomanlagen in den USA unschädlich zu machen]. (742)

Sentences 1 and 2 give examples of subject pronouns, which are directly attached to the auxiliary in the corpus. They are predicted to occur in the vorfeld or in the mittelfeld. The precedence table reveals that subject pronouns occur more often in the mittelfeld than in the vorfeld, while the trend is the opposite for full NPs. This is counter-intuitive at first sight, because pronouns are contextually bound, while full NPs are not. Thus, the data is evidence for type of syntactic construction where the vorfeld creates additional focus. Drodowski and Eisenberg (1995) calls this function *Ausdruckstellung* (p. 719).[4]

Sentences 3 and 4 provide examples of accusative NPs. Since non-subject material is not annotated under the auxiliary, but under the non-finite embedded predicator in the corpus, all accusative NPs in the field description of auxiliary verbs are non-local. Compactly realised accusatives, in contrast, are dealt with in the field description of non-finite embedded predicators. This is the reason

[4]Other examples include "Noch nie habe ich so viel gewählt" (902) and "Auf lange Sicht wird sie sich bezahlt machen" (566).

for accusatives not occurring in the “mittelfeld” positions 5 and 6 of FULL-D2. The field description offers field 0 (a “vorfeld” position), or field 8 (a “nachfeld” position). Non-local accusatives in the extraposition field are instances of partial fronting Kuthy and Meurers (1999).

Sentences 5 and 6 provide more examples of “nachfeld” material. Example 5 refers back to footnote 3, page 95. Example 6 is an instance of a placeholder construction: “es” takes a mittelfeld position, while the longer infinitive clause is extraposed to the nachfeld.

The border between the field for conjunctions and the vorfeld is fuzzy. Obviously, there was not enough evidence for junctors consistently preceding, for instance, question pronouns [pos:PWAV] in model FULL-D2, so they were included into the same field. The reader is reminded at this point that the strategy of ordering fields in the absence of clear data is left-alignment, as defined in Algorithm 5 in Section 4.6.

Main verbs occurring in subordinate clauses [pos:VVFIN,rel:-root] are the third class described in detail. There are 13 fields with 45 elements, 44% of which are split for MAN-E4 (see page C.1), and 19 fields with 88 elements, 63% of which are split for FULL-D2 (page C.2).

Since subordinate clauses can exhibit V2 or VF verbal position, the identification of a sentence bracket is less straightforward. I will focus on the fields to the left of the head in this discussion, and assume that the head occurs in final position. In MAN-E4, the head occurs in field 7, in FULL-D2, it occurs in field 11. This means that the middlefield positions are 3 to 6 in MAN-E4, and 2 to 10 in FULL-D2.

Automatic feature adaption created much more fine-grained middlefield rules. Table 6.6 shows that the rules discovered for the order of these middlefield elements correspond in fact largely to middlefield order rules known in the descriptive literature. The sequence of subject pronouns, reflexive accusative pronouns, full accusative NPs, dative pronouns, subject NPs and dative NPs is found exactly as predicted by Engel (1970) in the learned field description (see Section 3.2). Note that [rel:SB] refers to full NP subjects mainly, due to the existence of the more specific feature structure [pos:PPER,rel:SB], and [rel:DA] refers to dative pronouns mainly, due to the existence of [pos:NN,rel:DA]. There are only two slight differences: Engel’s definiteness effects could not be found, since the corpus is not explicitly annotated for definiteness. Actually, Hoberg (1981, p.42) identifies a problem with Engel’s rule with regard to definiteness, and proposes animaticity instead of definiteness. There is a reference to [pos:NE,rel:OA] in field 4, and to [pos:CARD,rel:OA] in field 8, which backs Hoberg’s finding: Proper names are mainly animate, while numeral values are certainly not. The second difference is that dative pronouns and full subject NPs occur in the same field in the learned field description.

Summary The analysis of some of the field revealed clear correspondences between a traditional topological field analysis, augmented by rules on middlefield order. There are however some important differences.

First, traditional topological field analyses start with the basic assumption of basic clause types of German (V1, V2, VF), where pragmatic as well as syntactic factors determine the choice of major clause type. Since automatic acquisition

Engel (simplified)	NPron	AReflPron	APron
FULL-D2	[pos:PPER,rel:SB]	[pos:PRF,rel:OA]	[pos:PPER,rel:OA]
	2	3	5

DPron	N	A	D
[rel:DA]	[rel:SB]	[pos:NN,rel:OA]	[pos:NN,rel:DA]
6	6	7	10

Table 6.6: Comparison of automatically acquired middle field order rules with descriptive analyses

class	total rule count	% ignored head rules
pos:VVFIN,rel:-root	210	0.63
pos:VAFIN,rel:-root	177	0.61
pos:other	318	0.54
pos:ADV	46	0.41
pos:VMFIN,rel:-root	87	0.40
pos:NN	97	0.34
pos:KON	39	0.33
pos:ADJD	31	0.22
pos:NE	38	0.21
pos:VVPP	40	0.20
pos:ADJA	44	0.15
os:VVINF	38	0.13
pos:VVIZU	12	0
pos:VVFIN,rel:root	78	0
pos:VMFIN,rel:root	32	0
pos:VAFIN,rel:root	59	0
pos:KOUS	1	0

Table 6.7: Quality of head feature selection function expressed as ignored head rules during decision tree learning, calculated from model FULL-D2

of the head feature selection function was excluded from the objectives of this thesis, only a coarse approximation of the major clause types was possible, by introducing the distinction of [pos:VxFIN,rel:root] and [pos:VxFIN,rel:-root]. It is possible, though, to extend the architecture towards automatic head feature selection function adaption. In fact, the decision tree learner produces even rules that refer to head features during feature adaption, but the algorithm ignores them, because the head feature selection function is static (see Section 4.7.2). Table 6.7 shows the number of ignored rules that referred to head classes. This count can serve as a measure of quality of the head feature selection function. The table shows that there is in fact a strong force to adapt the head feature selection function of verbs occurring in subordinate clauses. It may be conjectured that this is due to declarative main clauses being the dominant clause type in newspaper texts, while subordinate clauses with verb second position are also a peculiarity of newspaper style.

Second, there is a conceptual difference in traditional topological field analyses between rules establishing the overall field structure, and more fine-grained rules governing placement of elements within these fields. This conceptual difference is non-existent in the learned field description: a rule predicting that reflexive pronouns occur before full NPs is not conceptually different from a rule predicting that mittelfeld material occur before the right sentence bracket.

Third, there is a difference between the learned field descriptions and traditional analyses with regard to non-local material and the "verbal complex". Topological field analyses assume all non-finite verbal material to constitute a single field, and analyse arguments of any of these verbs in the same way. The "verbal complex" of the learned field description reduces to the occurrence of a single element [rel:OC], and further embedded verbal material is treated within the field descriptions of the embedded head, as long as it is realised continuously.

An issue which was only mentioned in passing in the discussion above is the treatment of coordinated structures. Standard topological analyses exclude conjunction from their scope. In the field descriptions learned, coordinated structures occur as [rel:CJ] directly to the right of the head. This is an epiphenomenon of the input corpus conversion procedure outlined in Section 4.3. There is also a separate field description for subordinating and coordinating conjunctions ([pos:KON], [pos:KOUS]).

6.6 Discussion

In the preceding sections, different parameterisations of the implemented word order learning architecture were evaluated in terms of their convergence behaviour, using precedence table size and field description size as a quantitative, internal measure. Furthermore, the rules were applied to two external tasks: converting a test corpus of ID structures into a topological treebank (rule application task, corpus comparison), and using the rules for TDG generation. The learned field descriptions were also linguistically judged, and compared to descriptive analyses in the literature.

The evaluation of the convergence behaviour showed that Precedence Table Size and Field Description Size converges only for BASE models (without splitting and automatic feature adaption) within 500 sentences. The reason is that there is a priori maximum on the number of elements for these models. All other models show logarithmic growth of Precedence Table Size and Field Description Size with increasing Corpus Size. However, field descriptions grow slowly, even if precedence tables may grow fast. The basic topological structures known from the literature are picked up first, and in the magnitude of some hundred sentences. Chapter 4 gave examples of field descriptions learned from small corpora. More fine-grained rules are added later. Logarithmic growth has been observed on similar learning tasks in the literature, e.g. by Krotov et al. (1998), who propose a method of *compacting* the phrase structure rules they acquire from the Penn Treebank. Modified thresholding techniques could achieve a similar result in the word order learning system.

Without robust edge selection, the system gets into loops of over- and underfitting. Robust edge selection (*edgethresh*=2, *edgefactor*=95) was discovered as the relevant parameter which remedies this problem for manual (MAN) models. The effect was less clear for automatic (FULL) models. Here, the parameters *adaptfactor* and *decthresh* had an influence on Field Description Size, but the behaviour of the system remained more volatile. Since Field Description Size shows more variation than Precedence Table Size in FULL model, this seems to be due to nodes being established as separate fields, or collapsed into equivalent classes if the edge selection threshold is not reached. This explanation is conclusive, because decision tree rules seldom achieve a confidence value as high as the edge selection threshold. There is a number of possible steps which could make automatic feature adaption more robust. I mention two. The inclusion of more linguistic features into the input data could result in the discovery of more reliable decision tree rules. The use of *Splitting on Failure* rather than *Head Splitting* (Section 4.7) might be more stable. This strategy was not tested due to long running times.

External evaluation showed that models with splitting and climbing clearly outperform models without splitting and climbing. In the Rule Application Task, sentence recall was 1.5 to 3 times higher for models MAN and FULL than for BASE, CLIMB and DEC. The best-performing model in the Rule Application Task was FULL-D3, which achieved 97.6% recall on nodes, and 71.3% recall on sentences, slightly outperforming the other FULL models (FULL-D2 97.4% node recall) and MAN-E4 (97.3% node recall). These figure mean that the learned word order rules cannot account for 2.3% of the word order configurations observed under a given head in the training corpus. In 28.7% of the sentences of the training corpus, there is at least one such node. Becker and Frank (2002) report 92.9% recall on brackets (which corresponds to nodes) in their supervised learning scenario. The two figures are not directly comparable: On the one hand, their task is harder, since they learn from POS tags rather than tree structures; on the other hand, their task is easier since they have a supervised learning scenario. Also the results of Klein and Manning (no year) are not directly comparable. They report an F1 figure of 71%. Recall seems to be around 80%. However, their task is considerably harder, since the system is unsupervised and has POS tags only as input. Furthermore, their aim is the acquisition of phrase structure rules rather than word order rules.

In the Rule Application Task, also a more detailed evaluation by head class was provided. The most remarkable trend is that the system performs better on non-verbal material than on verbal material, which is not surprising due to the higher amount of word order freedom and non-local elements under verbs.

The comparison of the produced LP corpus with Becker and Frank's Gold Standard provided another possibility of comparison. Here, FULL-D2 achieved 74.5% unlabelled bracket recall, with MAN-E4 performing slightly worse. It was argued that the main reason for this figure being significantly lower than FULL-D2's 97.4% node recall in the Rule Application Task is a different linguistic analysis in the Gold Standard. This is a common problem in unsupervised learning, also noted by Klein and Manning (no year). A substantial difference is the treatment of mittelfeld material depending on the embedded non-finite predicator. Such material "climbs" in the Gold Standard, but is treated within the field description of the embedded predicator in the word order learning system. This behaviour could be changed through an adaptation of the Climbing Module: constituents would have to be assumed to climb even if there is no discontinuity observable in the concrete sentence, a notion which corresponds to TDG's "forced climbing" (compare Section 4.4).

Only a preliminary experiment could be presented on using the learned rules for generation. This experiment showed that the inclusion of field cardinality constraints is a necessary step left for further research, when the rules are to be used for generation. Field cardinality constraints can, of course, still be added manually, which is far easier a task than manually specifying a complete set of field descriptions. Also TDG's *blocks* statement is a preliminary to using the rules for generation. Learning of *blocks* relates to the possible adaptation to the climbing module mentioned in the last paragraph, and is considered a hard learning problem.

The linguistic judgement of the rules showed considerable similarities between the topological field models proposed in the descriptive literature and the learned instantiations of a general topological field model. In the field descriptions of finite auxiliary verbs, the left and right sentence bracket could be clearly identified. Examples of extraposed and topicalised material were given and related to the work discussed in Section 3.2. The graphical user interface offers the possibility to extract corpus evidence for each of the elements in the field description, and for each precedence pair in the precedence table. This makes the implementation a valuable linguistic research tool. Automatic feature selection was able to discover fine-grained linguistic rules on mittelfeld order, as the ones proposed by Engel (1970) and Hoberg (1981).

Differences between topological field analyses in the literature and the learned ones were the treatment of the verbal cluster and the verbal vorfeld. In the present stage of the system, the V2 constraint of the German data is only very implicitly present in the data. It is likely, though, that the system can be scaled up to include automatic head class selection, which could enable the discovery of the basic devision of V1, V2 and VF sentences of German. As an alternative, this information could be fed into the system, and separate field descriptions for V1, V2, VF could be learned.

Chapter 7

Conclusion

In this thesis, a new machine learning approach was developed, which induces symbolic word order rules from a syntactically annotated corpus. The rules take the form of a set of topological field descriptions, following a generalized version of the well-known topological fields approach. In order to arrive at these descriptions, a graph algorithm was developed which computes a partial order on elements occurring under the same class of head item. A conversion procedure on the input trees (climbing) ensures that non-local configurations can be accounted for. In manual feature selection, the linguistic elements are manually reduced to a set of descriptive primitives; in automatic feature selection, classes of elements are automatically learned, selecting word order relevant features. Decision tree learning is used as a part of the learning system in a feedback loop with the other modules. A prototype of the system was implemented in Java, and evaluated on the German Negra corpus. For evaluation, the input treebank was converted into a topological treebank, yielding 97.6% recall on nodes, and compared to an independently and manually created topological treebank, yielding 74.5% recall on brackets. The system produces reliable results on around 1000 sentences already, but shows volatile behaviour for some parameter configurations, which may improve with the inclusion of richer data. The system was able to discover basic topological facts of German, e.g. the sentence bracket at the sentence level, as well as fine-grained tendencies, e.g. on the order in the mittelfeld. Besides the topological treebank, the output of the system is a set of formalism-independent, human-readable field descriptions, which were shown to be applicable to parsing and generation in a preliminary experiment.

In contrast to Villavicencio (2000), the aim of the system is not to model human learning, but to be useful for natural language engineering applications. Hence, the goal was not the derivation of "basic" word order only, but to account for both constraints and variability occurring in a language.

The system is clearly modularized externally and internally by linguistic criteria, which means it has a relatively high amount of linguistic abstraction in the sense of Kinyon and Prolo (2002). It is therefore a contribution to the development of linguistically abstract, yet partially learned grammars for natural language applications. This type of grammars is necessary due to maintenance and coverage problems of manually developed grammars, and poor linguistic value of earlier induced grammars.

The system makes also a theoretical contribution. It was shown that a generalized version of the topological fields approach, as assumed by Kruijff (2001), Duchier and Debusmann (2001), is applicable to "real" linguistic data, and can even be automatically acquired. It was argued that the approach is also cross-linguistically valid, because it makes less assumptions about linguistic structure than, e.g. in a principles-and-parameters approach like Villavicencio (2000). Linguistically, the produced word order rules and the data from which they were constructed can serve as a testbed for descriptive claims. An example is the rules on mittelfeld order by Engel (1970).

In the machine learning architecture developed, decision tree learning is used embedded as a heuristic submodule during search for an appropriate feature selection function. Such an embedded architecture is uncommon, and of theoretical interest to the development of machine learning architectures. It seems that the approach slightly resembles the "wrapper approach" suggested by Kohavi and John (1997), who filter irrelevant features from the example descriptions before actually running a learning algorithm, and integrate these steps closely.

In half a years time, not all features of the approach could be fully implemented. A step that is immediately possible is the calculation of field cardinality constraints, by applying the rules to a test treebank, and counting the number of elements in each exported realization. Further experiments with linguistically different feature selection strategies are also desirable. It was not tested, e.g., how automatic feature adaption behaves with a different starting point for search, e.g. one where non-local elements are distinguished from local ones. A related issue, whose solution is in sight, is the learning of the head feature selection function, which was assumed as static for the course of this thesis. It was pointed out, though, that the decision tree learner produces in fact suggestions for adapting the head feature selection function, which were ignored in the implementation. If the interaction between head feature selection and dependent feature selection proves to be well-behaved this might result in increased convergence stability of the system, and even in the automatic discovery of features which determine the V1, V2, or VF verbal position of German, together with a clearer rendering of the V2 constraint.

Applying the architecture to different types of corpus data is another possible future advancement. Treebanks of other languages are available in the required format, and could yield interesting topological insights. It is also desirable to apply the system to more richly annotated corpora. This could allow the discovery of definiteness, thematic role, animaticity, semantic verb class, or topic-focus effects on word order, and increase stability and granularity further.

Other issues require more research to be integrated into the approach. It would be desirable to extend the algorithm employed by the climbing module to calculate climb trees from ID trees. A more sophisticated strategy could learn constraints on climbing in certain local tree contexts, and use this knowledge to infer the appropriate climbing distance. This amounts to learning TDG's *blocks* feature, which had to be manually added in the experiments of Section 6.4. This is considered a hard learning task though, and data sparseness problems are more severe than for the calculation of order graphs.

Another issue left for future research is to make use of the probabilistic information inherent in the precedence tables created by the system. The learned word order rules are not a probabilistic language model. While it is easily possible to include probabilistic information into the field descriptions, the interpretation of these probabilities depends on the concrete application at hand.

There is no statistical parser for TDG, nor has a computational framework for ranking TDG parses been developed. Research into this direction might, however, make use of the learned word order rules, and integrate probabilities into the framework. The exact nature of these probabilities requires further research.

The main insight that follows from this thesis is that grammar development can profit from machine learning techniques, applied to linguistically annotated data. With the implemented word order learning system, a linguistic ressource has been created which can serve for testing the predictions of linguistic theories and formal grammar formalisms against empirical data. The application of theories to such resources is the only way in which consistency and practical use of a theory can be shown, in the light of a vast number of competing theories around.

Many contemporary approaches to grammar learning are linguistically shallow. Through clear modularization, however, it is possible to identify sub-tasks and create a grammar development scenario where manually written and acquired modules coexist. Symbolic, human-interpretable rules are a prerequisite to such a scenario. For example, the approach could be integrated with a system for subcategorisation-frame acquisition, as the one by Sarkar and Zeman (2000).

Word order is a clearly identifiable sub-phenomenon of grammatical knowledge, which has not deserved enough attention in the literature yet. Many approaches that work for languages with fixed word order fail for languages with freer word order. The learned rules are most likely to prove useful in generation and parsing scenarios, particularly for languages which exhibit interesting word order regularities.

Thus, this thesis is a contribution to the development of linguistically insightful, better maintainable, robust, partially learned wide-coverage grammars for natural language applications.

Appendix A

Negra Annotation Scheme

Tag	Negra phrase tag
--	not bound
AA	superlative phrase with "am"
AP	adjective phrase
AVP	adverbial phrase
CAC	coordinated adposition
CAP	coordinated adjektive 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 zu-marked infinitive
DL	discourse level constituent
ISU	idiosyncratis unit
MTA	multi-token adjective
NM	multi-token number
NP	noun phrase
PN	proper noun
PP	adpositional phrase
QL	quasi-languag
S	sentence
VP	verb phrase (non-finite)
VZ	zu-marked infinitive

Table A.1: Negra phrase labels

Tag	Negra edgelabel
--	--
AC	adpositional case marker
ADC	adjective component
AG	genitive attribute
AMS	measure argument of adj
APP	apposition
AVC	adverbial phrase component
CC	comparative complement
CD	coordinating conjunction
CJ	conjunct
CM	comparative concjunction
CP	complementizer
CVC	collocational verb construction (Funktionsverbgefüge)
DA	dative
DH	discourse-level head
DM	discourse marker
EP	expletive es
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
OC	clausal object
OG	genitive object
OP	prepositional object
PAR	parenthesis
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	unit component
VO	vocative

Table A.2: Negra edge labels (syntactic function)

Tag	Negra POS-Tag
$(	Sonstige Satzzeichen; satzintern
$,	Komma
$.	Satzbeendende Interpunktion
--	--
ADJA	Attributives Adjektiv
ADJD	Adverbiales oder prädikatives Adjektiv
ADV	Adverb
APPO	Postposition
APPR	Präposition; Zirkumposition links
APPRART	Präposition mit Artikel
APZR	Zirkumposition rechts
ART	Bestimmter oder unbestimmter Artikel
CARD	Kardinalzahl
FM	Fremdsprachliches Material
ITJ	Interjektion
KOKOM	Vergleichspartikel, ohne Satz
KON	Nebenordnende Konjunktion
KOUI	Unterordnende Konjunktion mit zu und Infinitiv
KOUS	Unterordnende Konjunktion mit Satz
NE	Eigennamen
NN	Normales Nomen
NNE	Kombination aus Nomen und Eigenname
PDAT	Attribuierendes Demonstrativpronomen
PDS	Substituierendes Demonstrativpronomen
PIAT	Attribuierendes Indefinitpronomen
PIS	Substituierendes Indefinitpronomen
PPER	Ersetzbares Personalpronomen
PPOSAT	Attribuierendes Possessivpronomen
PPOSS	Substituierendes Possessivpronomen
PRELAT	Attribuierendes Relativpronomen
PRELS	Substituierendes Relativpronomen
PRF	Reflexives Personalpronomen
PROAV	Pronominaladverb
PTKA	Partikel bei Adjektiv oder Adverb
PTKANT	Antwortpartikel
PTKNEG	Negationspartikel
PTKVZ	Abgetrennter Verbzusatz
PTKZU	zu vor Infinitiv
PWAT	Attribuierendes Interrogativpronomen
PWAV	Adverbiales Interrogativ- oder Relativpronomen
PWS	Substituierendes Interrogativpronomen
TRUNC	Kompositions-Erstglied
UNKNOWN	Unbekanntes Tag aus Einlesen aus Korpusdatei
VAFIN	Finites Verb, aux
VAIMP	Imperativ, aux
VAINF	Infinitiv, aux
VAPP	Partizip Perfekt, aux
VMFIN	Finites Verb, modal
VMINF	Infinitiv, modal
VMPP	Partizip Perfekt, modal
VVFIN	Finites Verb, voll
VVIMP	Imperativ, voll
VVINF	Infinitiv, voll
VVIZU	Infinitiv mit zu, voll
VVPP	Partizip Perfekt, voll
XY	Nichtwort, Sonderzeichen

Table A.3: Negra POS tags

Appendix B

Mathematical definitions

A *directed graph* $G =< V, E >$ is a tuple of a set of nodes, and a set of edges $V \subseteq N \times N$. A path is a sequence of nodes $n_1, n_2, ...n_i$ such that $< n_1, n_2 >$ $, < n_2, n_3 >, ... < n_{i-1}, n_i > \in V$. A *cycle* is a path from n to n. The *length* of a path is the number of edges on the path. n' is *reachable* from n if there is a path from n to n'. The *indegree* and *outdegree* of a node are the number of ingoing and outgoing edges. A *strongly connected component* G' of a graph G is a maximal subgraph in which any node can be reached from any node. The *adjacency list representation* of a graph is a data structure, which holds a set of nodes, each of which holds a set of outgoing and incoming edges.

A *tree* $T =< N, V, r >$ is a triple of a set of nodes N, a set of directed edges $V \subset N \times N$, and a designated node $r \in N$, the *root.* For every node $n \in N$ except for the root, there is exactly one edge $< n', n > \in V$, which leads towards n. There is no edge that leads towards r. n' is called the parent, or (linguistically) the governor of n. n is called a child, or (linguistically) a dependent of n'. A node n' on a path towards the root from n is a *transitive parent (governor)* of n. A transitive parent which is not a direct parent is called an *indirect parent (governor).*

A *non-recursive feature structure* F, or attribute-value matrix, AVM, is a function from a set of features to a set of values. Both features and values are atomic. $feat(F)$ designates the value of feature $feat$. An AVM can be depicted as [f1:v1,f2:v2,...]. A feature structure *contains* a feature, if the function is defined for this feature. A feature structure F_1 *subsumes* a feature structure F_2 (in symbols, $F_1 \sqsubseteq F_2$) if F_2 contains all the features of F_1, and has the same values as F_1 for these features. The *unification* of two feature structures F_1 and F_2 is the feature structure which contains all features from F_1 and from F_2. If there is a feature f which both F_1 and F_2 contain, but $f(F_1) \neq f(F_2)$, unification is not defined (*fails*).

PARSEVAL measures: *Recall* is the number of correctly identified brackets divided by the number of gold standard brackets. *Precision* is the number of correctly identified brackets divided by the number of identified brackets. *f-value* is the average of precision and recall. Recall, precision and f-value are *unlabelled* if the non-terminals of brackets in test corpus and gold standard are ignored, or *labelled* if they need to match. A bracket whose span crosses any bracket's span in the gold-standard is a *crossing bracket.*

Given a function $f : D \rightarrow R$, the *inverse image* of $f(v)$, $f^{-1}(v)$ is the subset of $d \in D$ for which $f(d) = v$.

A *partial order* is a reflexive, antisymmetric, transitive binary relation on a set. A *total order* is a reflexive, antisymmetric, transitive, total binary relation on a set.

Appendix C

Data

This appendix includes the full set of field descriptions learned by models MAN-E4 and FULL-D2, trained on 9730 resp. 4387 sentences of the Negra corpus. A field description consists of a head class, and a table indicating the field index together with the elements that can be realized in this field. The head selection function distinguished head classes by POS tag, with a further distinction between verbal heads at the root node of the sentence, and verbal heads in subclauses. The position of the head within the field description is represented by the feature structure [head:yes]. Elements referred to in the main text of Section 6.5 are in bold face.

C.1 Model MAN-E4

[pos:NN]

0	[rel:NG] [rel:JU] [rel:CM] [dist:1,rel:APP] [rel:SB,split:pre-hd] [rel:PD,split:pre-hd]
1	[rel:MO,split:pre-hd]
2	**[rel:AC]**
3	[rel:NMC,split:pre-hd] [rel:GL]
4	**[rel:NK]**
5	**[head:yes]** [dist:2,rel:OC]
6	[rel:CD] [rel:DA,split:post-hd] [dist:1,rel:RE] [rel:PNC] [dist:1,rel:CC] [rel:PH] [rel:UC] [rel:SVP] **[dist:2,rel:MO]** [rel:RS]
7	**[rel:CJ]** **[rel:GR]** [rel:PG]
8	[dist:1,rel:CJ] [rel:OA] [rel:CC] [rel:MNR] [rel:PD,split:post-hd]
9	[rel:OC] [rel:SB,split:post-hd] **[dist:1,rel:RC]**
10	[rel:MO,split:post-hd]
11	[rel:APP] [rel:RC] [dist:1,rel:MNR]

[pos:VAFIN,rel:-root]

0 [rel:JU] [rel:CM] [dist:2,rel:MNR] [dist:1,rel:NK] [rel:DA] [rel:CJ,split:pre-hd] [rel:DM] [dist:1,rel:OC,split:pre-hd] [dist:1,rel:PD] [rel:AC] [dist:2,rel:PD] [dist:1,rel:CJ,split:pre-hd]
1 [rel:CP] [rel:MO,split:pre-hd] [rel:NK]
2 [rel:OA,split:pre-hd] [rel:GR] [dist:1,rel:DA,split:pre-hd] [rel:NG,split:pre-hd] [dist:1,rel:OA,split:pre-hd] [rel:SVP,split:pre-hd] [dist:2,rel:OA] [rel:SP]
3 [rel:MNR] [dist:1,rel:MO,split:pre-hd]
4 [dist:2,rel:MO,split:pre-hd] [rel:SB,split:pre-hd]
5 [dist:2,rel:DA] [dist:1,rel:PG] [rel:PD,split:pre-hd]
6 [rel:OC,split:pre-hd] [rel:MO,split:post-hd] [dist:1,rel:MNR] [head:yes] [rel:OA,split:post-hd] [rel:NG,split:post-hd] [rel:SVP,split:post-hd] [rel:SB,split:post-hd] [dist:1,rel:DA,split:post-hd] [dist:2,rel:MO,split:post-hd]
7 [dist:2,rel:RE] [dist:2,rel:CC] [dist:1,rel:CJ,split:post-hd] [dist:1,rel:RE] [dist:3,rel:MNR] [dist:1,rel:OC,split:post-hd] [dist:3,rel:CC] [dist:1,rel:OA,split:post-hd] [dist:1,rel:APP] [dist:1,rel:CC,split:post-hd] [dist:2,rel:RC]
8 [dist:1,rel:MO,split:post-hd]
9 [rel:OC,split:post-hd] [rel:PD,split:post-hd]
10 [dist:1,rel:RC] [rel:CJ,split:post-hd]

[pos:KON]

0 [rel:CM] [dist:1,rel:MO,split:pre-hd] [rel:NG] [dist:1,rel:CP] [rel:MO,split:pre-hd]
1 [dist:1,rel:SB] [rel:AC]
2 [rel:NK,split:pre-hd] [dist:1,rel:OA]
3 [rel:GR,split:pre-hd] [rel:CJ,split:pre-hd]
4 [head:yes]
5 [rel:AVC] [rel:OA,split:post-hd] [rel:CD] [rel:CJ,split:post-hd] [rel:SB]
6 [rel:RC] [rel:NK,split:post-hd] [dist:1,rel:OC] [rel:MO,split:post-hd] [dist:1,rel:MO,split:post-hd] [rel:APP] [rel:OC] [rel:GR,split:post-hd] [rel:DH] [dist:2,rel:APP]
7 [rel:MNR]

[pos:NE]

0 [rel:CM] [rel:NG] [rel:SB,split:pre-hd]
1 [rel:MO,split:pre-hd]
2 [rel:AC]
3 [rel:GL]
4 [rel:NK]
5 [head:yes] [dist:3,rel:OA]
6 [rel:CD] [rel:OA] [rel:ADC] [rel:PNC] [rel:UC] [rel:SB,split:post-hd]
7 [rel:RS] [rel:CJ] [rel:PD,split:post-hd] [rel:MNR] [rel:GR] [dist:1,rel:RE] [rel:PG] [rel:MO,split:post-hd] [rel:OC]
8 [rel:APP]
9 [rel:RC]

[pos:VMFIN,rel:-root]

0	[rel:CP] [rel:CJ,split:pre-hd] [dist:2,rel:MNR] [rel:CM] [dist:1,rel:PD] [dist:1,rel:NK] [rel:JU] [dist:1,rel:OC,split:pre-hd] [dist:2,rel:OA,split:pre-hd] [rel:MO,split:pre-hd]
1	[dist:1,rel:OA,split:pre-hd] [rel:SB,split:pre-hd] [dist:2,rel:MO,split:pre-hd] [rel:OA,split:pre-hd] [dist:1,rel:DA]
2	[dist:1,rel:MO,split:pre-hd]
3	[rel:NG,split:pre-hd]
4	[rel:OC,split:pre-hd]
5	[head:yes]
6	[dist:1,rel:OA,split:post-hd] [dist:2,rel:MO,split:post-hd] [dist:2,rel:OC] [dist:1,rel:RE] [dist:2,rel:OA,split:post-hd] [dist:1,rel:MO,split:post-hd] [dist:1,rel:OC,split:post-hd] [dist:2,rel:RC] [dist:3,rel:RC] [rel:OA,split:post-hd] [dist:2,rel:RE] [dist:2,rel:CC]
7	[rel:SB,split:post-hd] [rel:MO,split:post-hd]
8	[rel:NG,split:post-hd] [rel:SVP] [dist:1,rel:MNR]
9	[rel:OC,split:post-hd]
10	[rel:CJ,split:post-hd] [dist:1,rel:RC]

[pos:other]

0	[dist:2,rel:MO,split:pre-hd] [rel:MNR,split:pre-hd] [rel:OA,split:pre-hd] [rel:CM] [rel:DA,split:pre-hd] [rel:JU] [dist:1,rel:OA] [rel:CP] [rel:RE,split:pre-hd]
1	[rel:AC] [rel:MO,split:pre-hd] [rel:NG,split:pre-hd] [dist:1,rel:MO,split:pre-hd]
2	[rel:SB,split:pre-hd] [rel:APP,split:pre-hd] [rel:PD,split:pre-hd] [rel:GL]
3	[rel:NK,split:pre-hd]
4	[rel:NK,split:post-hd] [rel:PNC] [rel:CJ] [rel:OC] [head:yes] [rel:CD] [rel:NMC] [rel:PM] [rel:SB,split:post-hd] [rel:OA,split:post-hd] [rel:UC] [rel:ADC]
5	[rel:MNR,split:post-hd] [dist:1,rel:CC] [rel:PG] [rel:MO,split:post-hd] [dist:2,rel:RE] [rel:DA,split:post-hd] [rel:GR] [rel:CC] [dist:2,rel:CC] [rel:RC] [dist:1,rel:CD] [rel:AVC] [rel:NG,split:post-hd] [dist:1,rel:CJ] [dist:1,rel:APP] [dist:1,rel:MO,split:post-hd] [dist:1,rel:OC] [dist:2,rel:OC] [dist:1,rel:RE] [dist:2,rel:RC] [dist:1,rel:RC]
6	[rel:SVP] [rel:APP,split:post-hd] [rel:PH] [rel:PD,split:post-hd]
7	[rel:RE,split:post-hd]

[pos:KOUS]

0	[rel:CJ,split:pre-hd]
1	[head:yes]
2	[rel:CD] [rel:CJ,split:post-hd] [rel:SB] [rel:MO] [rel:NG]
3	[rel:OC] [rel:PD]

[pos:VVINF]

0 [rel:CP] [rel:SBP] [rel:SVP] [rel:CM] [rel:NK]
1 [rel:DA] [rel:GR] [rel:CJ,split:pre-hd] [rel:NG]
2 [rel:MO,split:pre-hd] [rel:OA]
3 [dist:1,rel:MNR,split:pre-hd] [rel:OC,split:pre-hd] [dist:1,rel:CJ] [rel:PD]
4 [rel:PM]
5 [head:yes]
6 [rel:RC] [rel:MO,split:post-hd] [dist:1,rel:CC] [dist:2,rel:APP] [rel:OC,split:post-hd] [dist:1,rel:APP] [dist:2,rel:RC] [dist:1,rel:RC] [dist:2,rel:CC] [dist:1,rel:OC] [rel:APP] [dist:2,rel:RE] [dist:1,rel:RE] [rel:CD] [dist:1,rel:MNR,split:post-hd]
7 [rel:CJ,split:post-hd]

[pos:ADV]

0 [rel:RE,split:pre-hd] [dist:2,rel:MO,split:pre-hd] [rel:MO,split:pre-hd] [dist:1,rel:JU] [rel:CM] [rel:PD,split:pre-hd] [rel:AMS] [rel:SB,split:pre-hd] [dist:1,rel:MNR,split:pre-hd] [dist:1,rel:SB,split:pre-hd] [rel:JU] [rel:NG,split:pre-hd]
1 [rel:CJ,split:pre-hd] [rel:AC] [dist:1,rel:MO]
2 [rel:NK,split:pre-hd]
3 [head:yes]
4 [rel:CD] [rel:OC] [dist:2,rel:MO,split:post-hd] [rel:PH] [rel:AVC] [rel:NK,split:post-hd] [rel:NG,split:post-hd] [rel:DA,split:post-hd] [rel:PD,split:post-hd] [rel:CC] [rel:SB,split:post-hd] [rel:DM] [dist:1,rel:CC] [rel:OA,split:post-hd]
5 [rel:MNR] [rel:GR] [rel:RE,split:post-hd] [rel:MO,split:post-hd] [rel:APP] [rel:CJ,split:post-hd]

[pos:VVPP]

0 [rel:SB,split:pre-hd] [rel:NG] [rel:OA] [dist:1,rel:CJ] [rel:CM] [rel:AC] [rel:CJ,split:pre-hd] [rel:SBP,split:pre-hd] [rel:DA]
1 [rel:MO,split:pre-hd] [rel:PD]
2 [rel:OG] [dist:1,rel:MNR]
3 [head:yes]
4 [dist:2,rel:RC] [dist:2,rel:CC] [rel:OC] [dist:1,rel:APP] [rel:CD] [rel:SBP,split:post-hd] [dist:1,rel:RE] [dist:1,rel:OC] [dist:1,rel:RC] [rel:CJ,split:post-hd] [dist:1,rel:CC]
5 [rel:MO,split:post-hd]

[pos:VVFIN,rel:-root]

0 [rel:OA2] [rel:AC] [rel:CM] [rel:JU] [rel:CJ,split:pre-hd] [dist:2,rel:MO]
1 [rel:NK]
2 [rel:CP]
3 [rel:DA,split:pre-hd] [dist:1,rel:DA] [dist:1,rel:MNR] [rel:OA,split:pre-hd] [dist:1,rel:MO,split:pre-hd] [dist:1,rel:OA,split:pre-hd] [rel:SB,split:pre-hd]
4 [rel:PD,split:pre-hd] [rel:NG,split:pre-hd] [rel:MO,split:pre-hd]
5 [rel:OC,split:pre-hd] [dist:1,rel:CJ,split:pre-hd] [dist:1,rel:AC]
6 [dist:1,rel:NK]
7 **[head:yes]**
8 [dist:1,rel:MO,split:post-hd] [dist:2,rel:RC] [rel:MNR] [dist:1,rel:OA,split:post-hd] [dist:2,rel:RE] [dist:2,rel:MNR] [rel:SB,split:post-hd] [dist:1,rel:CC] [rel:OA,split:post-hd] [rel:DA,split:post-hd]
9 [rel:MO,split:post-hd] [rel:NG,split:post-hd] [rel:OG]
10 [rel:RC] [rel:PD,split:post-hd] [rel:SVP] [rel:CJ,split:post-hd]
11 [dist:1,rel:RE] [dist:1,rel:RC] [dist:1,rel:OC]
12 [rel:OC,split:post-hd] [dist:1,rel:APP]

[pos:VMFIN,rel:root]

0 [rel:CJ,split:pre-hd] [rel:OA,split:pre-hd] [dist:2,rel:OA,split:pre-hd] [rel:JU] [rel:DM] [dist:2,rel:MNR] [dist:1,rel:DA,split:pre-hd] [dist:1,rel:PD] [dist:1,rel:OC,split:pre-hd] [rel:CP] [dist:1,rel:MNR,split:pre-hd] [dist:2,rel:MO,split:pre-hd] [dist:1,rel:RE,split:pre-hd]
1 [dist:1,rel:MO,split:pre-hd] [rel:SB,split:pre-hd] [dist:1,rel:OA,split:pre-hd] [rel:MO,split:pre-hd]
2 [head:yes]
3 [dist:2,rel:OA,split:post-hd] [dist:1,rel:OA,split:post-hd] [dist:1,rel:DA,split:post-hd] [rel:SVP] [dist:2,rel:MO,split:post-hd] [rel:OA,split:post-hd] [dist:2,rel:DA,split:post-hd]
4 [rel:SB,split:post-hd] [dist:1,rel:MO,split:post-hd] [rel:MO,split:post-hd]
5 [dist:1,rel:MNR,split:post-hd] [rel:NG]
6 [rel:OC] [dist:1,rel:RE,split:post-hd]
7 [dist:1,rel:APP] [rel:CJ,split:post-hd] [dist:1,rel:RC]

[pos:VVIZU]

0 [rel:NG] [rel:CP]
1 [rel:DA] [rel:OA] [rel:MO,split:pre-hd]
2 [dist:2,rel:RC] [head:yes]
3 [rel:OC] [dist:1,rel:RC] [dist:1,rel:RE] [rel:CD] [rel:RC] [rel:MO,split:post-hd]
4 [rel:CJ]

[pos:VAFIN,rel:root]

0	[dist:1,rel:MNR,split:pre-hd] [rel:CJ,split:pre-hd] [rel:OA,split:pre-hd] **[rel:JU]** [dist:1,rel:CJ,split:pre-hd] [rel:SP] [rel:DA,split:pre-hd] [rel:VO] [dist:1,rel:SBP] [rel:CP] [dist:1,rel:RE,split:pre-hd] [dist:2,rel:MO,split:pre-hd] [dist:2,rel:DA] [dist:2,rel:MNR,split:pre-hd] [dist:2,rel:OA,split:pre-hd] [dist:1,rel:NK,split:pre-hd] [dist:1,rel:PD] [rel:DM]
1	[dist:1,rel:MO,split:pre-hd] [dist:1,rel:DA,split:pre-hd] [dist:1,rel:OC,split:pre-hd] [rel:SB,split:pre-hd] [dist:1,rel:OA,split:pre-hd]
2	[rel:MO,split:pre-hd] [rel:PD,split:pre-hd]
3	**[head:yes]**
4	[dist:2,rel:MNR,split:post-hd] [rel:DA,split:post-hd] [dist:1,rel:MNR,split:post-hd] [dist:1,rel:DA,split:post-hd] [dist:1,rel:NK,split:post-hd] [rel:MO,split:post-hd] [dist:1,rel:OA,split:post-hd] [dist:2,rel:MO,split:post-hd] [dist:2,rel:OA,split:post-hd]
5	[rel:NG] [dist:1,rel:MO,split:post-hd]
6	[dist:1,rel:NG] [rel:OC] [dist:1,rel:CJ,split:post-hd]
7	[rel:SB,split:post-hd] [dist:2,rel:CC]
8	[rel:DH] [dist:1,rel:RC] [dist:2,rel:RC] [rel:OA,split:post-hd] [rel:RC]
9	[rel:PD,split:post-hd] [rel:SVP]
10	[dist:1,rel:APP,split:post-hd] [dist:1,rel:RE,split:post-hd] [dist:1,rel:OC,split:post-hd]
11	[rel:CJ,split:post-hd]

[pos:ADJD]

0	[rel:CJ,split:pre-hd] [rel:CM] [rel:AC] [rel:PM] [rel:OA,split:pre-hd] [rel:DA,split:pre-hd] [rel:JU] [rel:OC,split:pre-hd]
1	[rel:MNR] [rel:SB,split:pre-hd] [rel:NG] [rel:GL]
2	[rel:NK,split:pre-hd]
3	[rel:MO,split:pre-hd] [rel:GR] [rel:AMS]
4	[dist:2,rel:CC] [head:yes]
5	[rel:PG] [rel:PD] [dist:1,rel:CC] [dist:1,rel:RE] [rel:OC,split:post-hd] [rel:CD] [rel:OA,split:post-hd] [rel:DA,split:post-hd] [dist:1,rel:MNR] [rel:PH] [dist:1,rel:APP] [rel:APP] [rel:CJ,split:post-hd]
6	[rel:MO,split:post-hd] [rel:RE]
7	[rel:CC] [rel:SB,split:post-hd]
8	[rel:NK,split:post-hd]

[pos:VVFIN,rel:root]

0	[rel:VO] [rel:RS,split:pre-hd] [dist:1,rel:OC,split:pre-hd] [dist:1,rel:CJ,split:pre-hd] [dist:2,rel:MO,split:pre-hd] [rel:JU] [dist:2,rel:OA] [dist:1,rel:CC,split:pre-hd] [dist:3,rel:MO] [rel:DM] [dist:1,rel:GL] [rel:CJ,split:pre-hd] [dist:1,rel:NK,split:pre-hd] [dist:1,rel:PG] [rel:OG,split:pre-hd]
1	[rel:MO,split:pre-hd] [dist:1,rel:OA,split:pre-hd] [rel:CP,split:pre-hd] [rel:DA,split:pre-hd] [dist:1,rel:MNR,split:pre-hd]
2	[rel:OC,split:pre-hd] [rel:OA,split:pre-hd] [rel:SB,split:pre-hd]
3	[dist:1,rel:SB] [rel:PD,split:pre-hd]
4	[dist:1,rel:MO] [head:yes]
5	[rel:DA,split:post-hd] [dist:1,rel:NK,split:post-hd] [dist:1,rel:DA] [dist:1,rel:OA,split:post-hd] [rel:CD] [rel:SB,split:post-hd]
6	[dist:2,rel:RE,split:post-hd] [dist:1,rel:MNR,split:post-hd] [rel:MO,split:post-hd] [dist:2,rel:MO,split:post-hd] [rel:NG] [rel:OA,split:post-hd]
7	[rel:RC] [rel:RS,split:post-hd] [rel:PD,split:post-hd] [rel:OG,split:post-hd] [dist:1,rel:APP]
8	[rel:SVP]
9	[dist:2,rel:RC] [rel:OC,split:post-hd] [dist:2,rel:CC] [dist:1,rel:OC,split:post-hd] [dist:1,rel:RE] [dist:1,rel:RC] [rel:DH] [dist:2,rel:OC] [dist:1,rel:CC,split:post-hd] [dist:1,rel:CJ,split:post-hd]
10	[rel:CJ,split:post-hd]

[pos:ADJA]

0	[rel:GL] [rel:JU] [rel:NG] [dist:1,rel:MO] [rel:CM]
1	[rel:AC]
2	[rel:NK,split:pre-hd]
3	[rel:AMS] [rel:OG] [rel:CC] [rel:CP] [rel:OA] [rel:SBP] [rel:DA,split:pre-hd]
4	[rel:PM] [rel:MO]
5	[rel:PD] [head:yes]
6	[rel:CJ] [rel:NMC] [rel:PNC] [rel:CD]
7	[rel:NK,split:post-hd]
8	[rel:SB,split:post-hd] [rel:GR] [dist:1,rel:CC] [rel:PG] [rel:OC]
9	[rel:MNR]
10	[rel:APP] [rel:RC]

C.2 Model FULL-D2

[pos:NN]

0	[rel:MO,split:pre-hd] [pos:PROAV,rel:MO] **[rel:NG]** [pos:VAFIN,split:pre-hd] [pos:KON,split:pre-hd] [pos:PIS,rel:MO,split:pre-hd] [pos:VVFIN,split:pre-hd] [pos:NN,rel:MO,split:pre-hd] [pos:CARD,rel:CJ,split:pre-hd] [pos:NE,split:pre-hd] [rel:CM] [pos:PIAT]
1	[pos:ADV,rel:MO,split:pre-hd]
2	[pos:ADJD,rel:MO] **[rel:AC]**
3	[pos:PPOSAT,rel:NK] **[pos:NE,rel:GL]** [pos:PDAT,rel:NK] [pos:PIAT,rel:NK]
4	**[pos:ART,rel:NK]**
5	[rel:NK] **[pos:PIDAT,rel:NK]**
6	[pos:CARD,rel:NK] **[pos:ADJA,rel:NK]**
7	[pos:KON,rel:CD] [pos:NN,rel:NK] [] **[head:yes]**
8	[pos:VMFIN] [pos:VVINF] [rel:MO,split:post-hd] [rel:CC] [pos:CARD,rel:CJ,split:post-hd] [pos:NE,rel:PNC] [pos:ADV,rel:MO,split:post-hd] [pos:NE,split:post-hd] [pos:NN] [rel:OA] [pos:KON,rel:GR] [pos:NE,rel:PG] [pos:KON,rel:PG] [pos:KON,split:post-hd] [pos:PIS,rel:MO,split:post-hd]
9	[rel:PD] **[rel:CJ]**
10	[pos:PIS,rel:CJ] [pos:NE,rel:MNR] [pos:VAFIN,split:post-hd] **[pos:NE,rel:GR]** [pos:KON,rel:MNR] [pos:NN,rel:MO,split:post-hd]
11	[rel:MNR] [pos:VVFIN,split:post-hd] [rel:APP]
12	[pos:VAFIN,rel:SB] **[rel:RC]** [pos:VVFIN,rel:MO] **[rel:OC]**
13	[pos:KON,rel:APP]

[pos:VAFIN,rel:-root]

0 [pos:KOUS] [dist:0,pos:PWS,rel:SB] [pos:PRF,split:pre-hd] [pos:PPER] [dist:0,rel:AC] [dist:0,pos:VVINF,rel:SB] [pos:PDS] [dist:0,rel:JU] [rel:CJ,split:pre-hd] [pos:VAFIN,split:pre-hd] [pos:PWAV] [pos:PRELS] [pos:PRELS,rel:OA] [dist:0,rel:CM]

1 [] [rel:CP] [dist:0,rel:NK]

2 [dist:0,pos:PDS,rel:SB,split:pre-hd] [dist:0,pos:CARD,rel:SB,split:pre-hd] [dist:0,pos:PPER,rel:SB,split:pre-hd] [pos:NN,split:pre-hd] [dist:0] [dist:1,rel:OA,split:pre-hd] [dist:0,pos:ADJA,rel:SB] [dist:2,rel:OA] [dist:1,pos:NE,rel:MO]

3 [dist:2,rel:MO,split:pre-hd] [dist:0,rel:MO,split:pre-hd] [dist:1,pos:ADV,rel:MO,split:pre-hd] [dist:0,pos:ADJD,split:pre-hd] [dist:0,pos:ADV,rel:MO,split:pre-hd] [dist:1,pos:NN,rel:MO,split:pre-hd]

4 [dist:0,pos:NE,rel:SB,split:pre-hd] [dist:1,rel:MO,split:pre-hd] [dist:0,rel:SB,split:pre-hd] [dist:0,pos:CARD] [dist:0,pos:PIS,rel:SB,split:pre-hd]

5 [rel:NG,split:pre-hd] [dist:0,pos:NN,rel:OA,split:pre-hd] [dist:0,pos:NN,rel:SB,split:pre-hd] [dist:0,pos:NN,split:pre-hd]

6 [pos:NN,rel:PD,split:pre-hd] [pos:VVPP,rel:PD,split:pre-hd] [rel:PD,split:pre-hd] [dist:0,rel:OA] [dist:0,rel:OC,split:pre-hd]

7 [head:yes]

8 [dist:0,pos:NE,rel:SB,split:post-hd] [dist:0,pos:PDS,rel:SB,split:post-hd] [dist:1,rel:MO,split:post-hd] [pos:NN,split:post-hd] [dist:0,rel:SB,split:post-hd] [rel:RC] [pos:VVINF,rel:MO] [dist:0,pos:PIS,rel:SB,split:post-hd] [rel:APP] [pos:PRF,split:post-hd] [dist:0,pos:PPER,split:post-hd] [dist:0,pos:CARD,rel:SB,split:post-hd] [dist:2,rel:MO,split:post-hd] [dist:0,pos:PPER,rel:SB,split:post-hd]

9 [dist:1,rel:OA,split:post-hd] [dist:0,pos:ADV,rel:MO,split:post-hd] [rel:NG,split:post-hd]

10 [dist:0,pos:NN,rel:OA,split:post-hd] [dist:1,pos:ADV,rel:MO,split:post-hd] [dist:1,pos:NN,rel:MO,split:post-hd] [dist:0,pos:NN,rel:SB,split:post-hd] [dist:0,pos:ADJD,split:post-hd]

11 [rel:SVP] [dist:0,rel:MO,split:post-hd] [pos:VMINF] [dist:0,rel:OC,split:post-hd]

12 [dist:0,pos:NN,split:post-hd] [rel:CC,split:post-hd] [rel:PD,split:post-hd]

13 [pos:VVPP,rel:PD,split:post-hd] [pos:VAFIN,split:post-hd] [pos:NN,rel:PD,split:post-hd] [rel:RE]

14 [rel:CJ,split:post-hd]

[pos:KON]

0 [pos:FM,split:pre-hd] [split:pre-hd] [rel:CM] [pos:VAFIN,split:pre-hd] [pos:PRELS] [pos:VVPP,split:pre-hd] [pos:VMFIN,split:pre-hd] [rel:MO,split:pre-hd] [dist:0,rel:MO,split:pre-hd] [pos:VVFIN,rel:CJ,split:pre-hd]
1 [dist:1,split:pre-hd] [rel:AC]
2 [pos:TRUNC,split:pre-hd] [pos:ART,rel:NK]
3 [rel:NK,split:pre-hd] [pos:ADJA,rel:NK] [pos:CARD,rel:NK]
4 [pos:NE,split:pre-hd] [rel:CJ,split:pre-hd]
5 [pos:ADV,rel:CJ] [head:yes]
6 [pos:VMFIN,split:post-hd] [pos:VVFIN,rel:CJ,split:post-hd] [dist:0,rel:SB] [pos:TRUNC,split:post-hd] [pos:PROAV] [dist:1,split:post-hd] [rel:AVC] [dist:0,rel:MO,split:post-hd] [pos:NE,split:post-hd] [pos:FM,split:post-hd] [pos:VVPP,split:post-hd]
7 [pos:VAFIN,split:post-hd] [rel:CJ,split:post-hd] [rel:CD] [rel:MO,split:post-hd]
8 [rel:NK,split:post-hd] [pos:NE,rel:NK] [rel:MNR,split:post-hd] [rel:OC] [rel:GR] [rel:APP] [split:post-hd]

[pos:NE]

0 [split:pre-hd] [pos:ADJD,rel:MO,split:pre-hd] [pos:NN,rel:CJ,split:pre-hd] [rel:NG] [rel:CM] [rel:MO,split:pre-hd]
1 [pos:ADV,rel:MO,split:pre-hd]
2 [rel:AC]
3 [rel:GL]
4 [rel:NK]
5 [head:yes]
6 [rel:PNC] [pos:NN,rel:CJ,split:post-hd] [rel:GR] [rel:CD] [split:post-hd] [rel:OA] [rel:ADC] [rel:RE]
7 [rel:PG] [rel:CJ] [rel:MNR] [rel:PD] [pos:VVFIN] [pos:ADV,rel:MO,split:post-hd] [rel:RS]
8 [pos:KON] [rel:APP]
9 [rel:RC] [rel:MO,split:post-hd]

[pos:VMFIN,rel:-root]

0 [dist:0,pos:KON] [pos:NN,split:pre-hd] [pos:PWAV] [pos:PWS] [dist:0,pos:PRELS,rel:SB] [dist:0,split:pre-hd] [pos:PRELS] [dist:0,pos:PWAV] [pos:KOUS] [dist:0,pos:PWS,rel:SB]
1 [dist:0,pos:ADJA,rel:SB,split:pre-hd] [rel:MO,split:pre-hd] [dist:0,pos:PPER,split:pre-hd] [dist:0,pos:NE,rel:SB,split:pre-hd]
2 [rel:DA] [split:pre-hd] [dist:0,pos:ADV,split:pre-hd]
3 [pos:NN,rel:MO,split:pre-hd] [dist:0,rel:SB,split:pre-hd] [dist:0,pos:PIS,split:pre-hd]
4 [rel:NG,split:pre-hd] [dist:0,rel:MO,split:pre-hd]
5 [rel:OC,split:pre-hd] [pos:VVINF,rel:OC,split:pre-hd]
6 [head:yes]
7 [pos:NN,rel:MO,split:post-hd] [rel:RE] [dist:0,pos:NE,rel:SB,split:post-hd] [dist:0,pos:PIS,split:post-hd] [pos:NN,split:post-hd] [dist:0,rel:MO,split:post-hd] [dist:0,pos:PPER,split:post-hd] [dist:1,rel:OC,split:post-hd] [rel:MO,split:post-hd]
8 [split:post-hd] [dist:0,rel:SB,split:post-hd]
9 [dist:0,pos:ADV,split:post-hd] [rel:RC]
10 [dist:0,pos:ADJA,rel:SB,split:post-hd] [rel:NG,split:post-hd]
11 [rel:OC,split:post-hd] [pos:VVINF,rel:OC,split:post-hd]
12 [dist:0,split:post-hd]

[pos:other]

0 [dist:0,rel:CM] [dist:0,rel:MNR,split:pre-hd] [dist:1] [dist:0,pos:PIAT] [dist:0,rel:OA,split:pre-hd] [dist:0,pos:APPR,rel:AC] [dist:0,pos:KON,split:pre-hd] [pos:NN,split:pre-hd] [pos:ADV,rel:MO] [dist:0,rel:RE,split:pre-hd] [rel:MO,split:pre-hd] [dist:0,pos:NN,rel:MO,split:pre-hd] [dist:0,pos:NE] [dist:0,rel:CP] [dist:0,rel:DA] [dist:0,pos:CARD,rel:NK] [dist:0,rel:CJ,split:pre-hd] [dist:0,rel:AC,split:pre-hd]

1 [dist:0,rel:NK,split:pre-hd] [pos:ART,rel:NK] [dist:0,rel:APP,split:pre-hd] [dist:0,rel:GL] [dist:0,rel:NG,split:pre-hd] [dist:0,pos:ADV,rel:MO,split:pre-hd] [dist:0,pos:PIDAT,rel:MO] [dist:0,rel:MO,split:pre-hd] [dist:0,pos:PDAT,rel:NK] [dist:0,rel:SB,split:pre-hd]

2 [dist:0,pos:APPRART,rel:AC] [dist:0,rel:PD,split:pre-hd] [dist:0,pos:NN,split:pre-hd] [dist:0,pos:PIDAT,rel:NK]

3 [dist:0,rel:OC] [dist:0,pos:ADJA,rel:NK,split:pre-hd] [dist:0,pos:NN,rel:NK,split:pre-hd]

4 [dist:0,pos:NE,rel:NK,split:pre-hd] [dist:0,rel:PM]

5 [head:yes]

6 [rel:RC] [dist:0,pos:VAFIN,rel:MO] [pos:VVINF,rel:MO] [dist:0,pos:PROAV] [dist:0,rel:CD] [dist:0,rel:NG,split:post-hd] [dist:1,rel:OC] [pos:NN,split:post-hd] [dist:0,pos:VAFIN] [dist:0,rel:UC] [dist:0,pos:APZR,rel:AC] [rel:CC] [dist:2] [dist:0,rel:SB,split:post-hd] [dist:0,pos:NN,rel:CC] [dist:0,rel:NMC] [dist:0,rel:AVC] [pos:VAFIN,rel:MO] [dist:0,pos:NN,rel:NMC] [dist:0,rel:AC,split:post-hd] [dist:0,rel:PD,split:post-hd] [dist:0,rel:PNC] [rel:MO,split:post-hd] [dist:0,pos:CARD,rel:NMC] [dist:0,pos:TRUNC] [rel:RE] [dist:0,rel:OA,split:post-hd] [pos:NE,rel:MO] [dist:0,rel:CC,split:post-hd]

7 [dist:0] [dist:0,rel:NK,split:post-hd] [dist:0,pos:NN,rel:CJ] [dist:0,rel:PG] [dist:0,pos:APPR,rel:CJ] [dist:0,pos:KON,split:post-hd] [dist:0,pos:ADV,rel:MO,split:post-hd] [dist:0,rel:CJ,split:post-hd] [dist:0,pos:ADJA,rel:CJ] [dist:0,pos:NE,rel:APP] [dist:0,rel:RE,split:post-hd]

8 [dist:0,pos:CARD,rel:CJ]

9 [dist:0,pos:ADJA,rel:NK,split:post-hd]

10 [dist:0,pos:NN,rel:NK,split:post-hd]

11 [dist:0,pos:NN,split:post-hd] [dist:0,rel:APP,split:post-hd] [dist:0,rel:MNR,split:post-hd] [dist:0,rel:GR] [dist:0,pos:NN,rel:PG,split:post-hd]

12 [dist:0,rel:RC] [dist:0,pos:NN,rel:MO,split:post-hd] [dist:0,pos:NN,rel:MNR] [dist:0,pos:NE,rel:NK,split:post-hd]

13 [dist:0,pos:VVFIN] [dist:0,rel:MO,split:post-hd]

[pos:KOUS]

0 [split:pre-hd]

1 [head:yes]

2 [split:post-hd] [rel:MO]

[pos:VVINF]

0 [dist:0,pos:ADJA] [dist:0,pos:PPER] [split:pre-hd] [rel:CP] [pos:KOUI] [dist:1,split:pre-hd]
1 [pos:PRF,rel:OA] [pos:PPER,rel:OA]
2 [rel:DA] [pos:ADV] [dist:0,pos:NN] [rel:MO,split:pre-hd] [pos:PRF]
3 [pos:PIS] [dist:0,pos:ADJD]
4 [rel:OA] [pos:PROAV] [dist:0,pos:CARD]
5 [rel:OC,split:pre-hd] [pos:NE,rel:MO]
6 [pos:PTKNEG]
7 [pos:PTKZU]
8 [head:yes]
9 [rel:MO,split:post-hd] [split:post-hd] [pos:VAFIN] [dist:1,rel:RC] [rel:APP] [dist:2] [pos:VVIZU,rel:MO] [rel:OC,split:post-hd] [pos:VMFIN,split:post-hd] [pos:VVINF,rel:MO] [dist:1,split:post-hd] [pos:VVFIN]

[pos:ADV]

0 [rel:NK,split:pre-hd] [dist:0,pos:NN,split:pre-hd] [split:pre-hd] [rel:AMS] [pos:CARD,split:pre-hd] [rel:CM] [pos:VAFIN,rel:CJ] [dist:0,pos:NN,rel:MO,split:pre-hd] [dist:1,split:pre-hd] [rel:RE,split:pre-hd] [rel:MO,split:pre-hd] [dist:0,pos:NN,rel:PD,split:pre-hd] [rel:NG,split:pre-hd] [dist:0,pos:NN,rel:AMS] [rel:CJ,split:pre-hd]
1 [pos:ADV,rel:MO,split:pre-hd] [rel:AC]
2 [head:yes]
3 [pos:VVFIN,rel:MO] [rel:MO,split:post-hd] [rel:NK,split:post-hd] [rel:PH] [dist:0,pos:NN,rel:MO,split:post-hd] [rel:AVC] [rel:CD] [split:post-hd] [pos:ADV,rel:MO,split:post-hd] [rel:NG,split:post-hd] [dist:1,split:post-hd] [rel:CC] [dist:0,pos:NN,rel:PD,split:post-hd]
4 [rel:RE,split:post-hd] [dist:0,pos:NN,split:post-hd] [rel:CJ,split:post-hd] [dist:0,pos:NN,rel:MNR]
5 [pos:CARD,split:post-hd]

[pos:VVPP]

0 [dist:1,split:pre-hd] [pos:VAFIN,split:pre-hd] [pos:PPER,rel:OA] [pos:PRF,rel:OA] [pos:PRF] [rel:SBP,split:pre-hd]
1 [rel:NG] [rel:DA] [dist:0,rel:MO,split:pre-hd] [pos:ADV] [dist:0,pos:NN,rel:MO,split:pre-hd]
2 [dist:0,pos:PROAV,rel:MO,split:pre-hd]
3 [rel:OA] [dist:0,pos:NE,rel:MO,split:pre-hd] [pos:ADJD]
4 [dist:0,split:pre-hd]
5 [head:yes]
6 [rel:OC] [dist:2] [dist:0,pos:PROAV,rel:MO,split:post-hd] [pos:VVINF] [pos:VVIZU] [pos:VVFIN] [dist:0,split:post-hd] [pos:VMFIN,split:post-hd] [dist:0,rel:MO,split:post-hd] [rel:CD] [dist:1,split:post-hd] [dist:0,pos:NE,rel:MO,split:post-hd] [dist:0,pos:NN,rel:MO,split:post-hd] [pos:VAFIN,split:post-hd] [rel:SBP,split:post-hd]
7 [dist:0,rel:CJ]

[pos:VVFIN,rel:-root]

0 [pos:PWS,rel:OA] [rel:CM] [pos:PRELS] [pos:VAFIN,split:pre-hd] [pos:VVFIN,split:pre-hd] [pos:KON,rel:JU] [rel:AC] [pos:PWAV] [pos:PWS]

1 [rel:CP] [rel:NK]

2 [pos:CARD,rel:SB] [pos:NN,rel:NK] **[pos:PIAT,rel:SB]** **[pos:PPER,rel:SB,split:pre-hd]** **[pos:PDS,rel:SB,split:pre-hd]**

3 [pos:PIS,rel:MO] **[pos:PRF,rel:OA,split:pre-hd]** **[pos:NN,rel:SB,split:pre-hd]** [pos:CARD,rel:MO,split:pre-hd]

4 **[pos:NE,rel:OA,split:pre-hd]** [pos:NE,rel:MO,split:pre-hd] [pos:PIS,rel:SB,split:pre-hd] [pos:TRUNC,rel:MO] [pos:PRF,rel:MO,split:pre-hd]

5 **[pos:PPER,rel:OA,split:pre-hd]** [pos:ADJA,rel:MO,split:pre-hd]

6 **[rel:SB,split:pre-hd]** [pos:PROAV,rel:MO,split:pre-hd] **[rel:DA,split:pre-hd]**

7 **[pos:NN,rel:OA,split:pre-hd]** [rel:MO,split:pre-hd] [pos:NN,rel:MO,split:pre-hd] [rel:NG,split:pre-hd]

8 [split:pre-hd] [pos:CARD,rel:OA,split:pre-hd] [pos:NN,split:pre-hd]

9 [pos:ADJD,rel:MO,split:pre-hd]

10 **[pos:NN,rel:DA,split:pre-hd]**

11 [pos:PRF,rel:OA,split:post-hd] [pos:ADJA,rel:OA,split:pre-hd] [pos:PPER,rel:OA,split:post-hd] [pos:PPER,rel:SB,split:post-hd] **[head:yes]** [pos:VVINF,rel:OC,split:pre-hd] [pos:NN,rel:SB,split:post-hd] [rel:OC,split:pre-hd] [pos:ADV,rel:MO,split:pre-hd] [pos:KON,rel:OA]

12 [rel:APP] [pos:KON] [pos:VMFIN] [pos:PIS,rel:SB,split:post-hd] [pos:VAFIN,split:post-hd] [rel:SB,split:post-hd] [pos:VVINF,rel:MO] [dist:1,rel:OC] [rel:DA,split:post-hd] [pos:NE,rel:OA,split:post-hd] [pos:PDS,rel:SB,split:post-hd] [rel:CC,split:post-hd] [pos:ADJA,rel:MO,split:post-hd]

13 [pos:ADV,rel:MO,split:post-hd] [pos:NE,rel:MO,split:post-hd] [pos:CARD,rel:MO,split:post-hd] [pos:PROAV,rel:MO,split:post-hd] [rel:NG,split:post-hd]

14 [pos:NN,rel:OA,split:post-hd] [pos:NN,rel:DA,split:post-hd] [rel:MO,split:post-hd] [pos:NN,rel:MO,split:post-hd] [pos:ADJA,rel:OA,split:post-hd] [pos:NN,rel:RE]

15 [pos:CARD,rel:OA,split:post-hd] [rel:OA] [pos:NN,split:post-hd] [rel:SVP] [pos:VVFIN,split:post-hd] [split:post-hd]

16 [pos:NN,rel:CC] [pos:ADJD,rel:MO,split:post-hd]

17 [pos:PTKVZ]

18 [pos:VVIZU,rel:MO] [rel:OC,split:post-hd] [rel:RE] [rel:RC] [pos:VVINF,rel:OC,split:post-hd] [pos:NN,rel:APP]

[pos:VMFIN,rel:root]

0	[pos:PROAV,split:pre-hd] [dist:1,rel:MO,split:pre-hd] [pos:CARD,rel:OA] [pos:VMFIN,rel:SB] [pos:PIS,rel:SB,split:pre-hd] [dist:2,split:pre-hd] [pos:ADJD,split:pre-hd] [rel:JU] [rel:CJ,split:pre-hd] [pos:KON,rel:SB,split:pre-hd]
1	[split:pre-hd] [pos:NN,rel:MO,split:pre-hd] [rel:SB,split:pre-hd] [pos:NN,split:pre-hd] [pos:ADV,split:pre-hd] [pos:NN,rel:SB,split:pre-hd] [pos:PWAV]
2	[rel:CP] [pos:PDS]
3	[head:yes]
4	[rel:SB,split:post-hd] [pos:PRF] [pos:ADJD,split:post-hd] [pos:PIS,rel:SB,split:post-hd] [dist:0,pos:NN,rel:OA] [pos:PROAV,split:post-hd] [dist:1,rel:MO,split:post-hd] [dist:2,split:post-hd] [pos:TRUNC,rel:SB]
5	[pos:NN,rel:SB,split:post-hd] [pos:ADV,split:post-hd]
6	[pos:NN,split:post-hd]
7	[split:post-hd] [pos:NN,rel:MO,split:post-hd] [rel:NG]
8	[rel:RE] [rel:OC] [pos:KON,rel:SB,split:post-hd]
9	[pos:VVINF,rel:OC]
10	[rel:CJ,split:post-hd] [rel:RC,split:post-hd]

[pos:VVIZU]

0	[rel:CP] [pos:PRF,rel:OA]
1	[split:pre-hd] [pos:NN,rel:MO] [pos:ADV] [rel:DA] [pos:ADJD] [rel:OA]
2	[pos:NN]
3	[head:yes]
4	[pos:VVFIN] [rel:RC] [split:post-hd]
5	[pos:VVINF]

[pos:VAFIN,rel:root]

0 [pos:PIS,split:pre-hd] [pos:VMFIN,split:pre-hd] [pos:KON,rel:SB,split:pre-hd] **[pos:PPER,split:pre-hd]** [pos:VVIZU,split:pre-hd] [split:pre-hd] [pos:VAINF,split:pre-hd] [pos:VVPP,split:pre-hd] [pos:PROAV,rel:MO,split:pre-hd] [dist:2,rel:MNR] [dist:0,pos:PWAV,rel:PD] [pos:VAFIN,rel:OC] [dist:2,rel:MO,split:pre-hd] [dist:0,pos:PPER,rel:PD,split:pre-hd] [dist:1,pos:NN,rel:OA,split:pre-hd] **[rel:JU]** [pos:ADJA] [dist:1,pos:ADJD,rel:MO,split:pre-hd] [pos:NN,split:pre-hd] [dist:1,pos:NN,split:pre-hd] [dist:0,pos:ADJD,rel:PD,split:pre-hd] [pos:KOUS] [rel:CJ,split:pre-hd] [rel:RE,split:pre-hd]

1 [dist:0,rel:PD,split:pre-hd] [dist:0,rel:OA,split:pre-hd] [pos:NN,rel:SB,split:pre-hd] [dist:0,pos:ADJD,split:pre-hd] [dist:1,pos:NN,rel:MO,split:pre-hd] [rel:SB,split:pre-hd] [pos:PPER,rel:SB,split:pre-hd] [pos:PDS,split:pre-hd]

2 [rel:MO,split:pre-hd] [pos:NE,rel:SB,split:pre-hd] [dist:1,rel:MO,split:pre-hd]

3 [pos:PRF,split:pre-hd]

4 **[head:yes]**

5 [pos:PROAV,rel:MO,split:post-hd] [dist:2,rel:MO,split:post-hd] [dist:0,pos:PPER,rel:PD,split:post-hd] [split:post-hd] **[pos:PPER,split:post-hd]** [pos:NN,split:post-hd] [dist:1,pos:ADJD,rel:MO,split:post-hd] [pos:PDS,split:post-hd] [pos:PPER,rel:SB,split:post-hd] [dist:0,pos:PROAV,rel:PD] [pos:PIS,split:post-hd]

6 [rel:NG] [rel:MO,split:post-hd] [dist:0,pos:ADJD,split:post-hd]

7 [dist:0,rel:OC] [dist:0,pos:CARD,rel:PD] [dist:1,pos:NN,split:post-hd]

8 [rel:CC] [dist:1,pos:NN,rel:MO,split:post-hd] [pos:NN,rel:SB,split:post-hd] [pos:PRF,split:post-hd] [dist:0,rel:OA,split:post-hd] [rel:SB,split:post-hd] [pos:NE,rel:SB,split:post-hd] [dist:1,pos:NN,rel:OA,split:post-hd]

9 [dist:0,pos:KON,rel:PD] [dist:1,rel:MO,split:post-hd] [rel:SVP]

10 [pos:VVIZU,split:post-hd] [dist:0,pos:ADJD,rel:PD,split:post-hd] **[pos:VVPP,split:post-hd]**

11 [pos:VMFIN,split:post-hd] [dist:0,rel:PD,split:post-hd] [rel:CJ,split:post-hd] [pos:KON,rel:SB,split:post-hd]

12 [pos:VAINF,split:post-hd] [rel:APP] **[rel:RE,split:post-hd]** [rel:DH]

13 [rel:RC]

[pos:ADJD]

0 [rel:PM] [rel:AMS] [rel:AC] [rel:DA,split:pre-hd] [pos:PROAV,rel:MO,split:pre-hd] [rel:CJ,split:pre-hd] [rel:OA] [rel:MO,split:pre-hd] [rel:CM]

1 [split:pre-hd] [rel:NG] [rel:NK]

2 [pos:NN,split:pre-hd] [pos:ADV,rel:MO]

3 [pos:PTKA,rel:MO] [pos:ADJD,rel:MO]

4 [head:yes]

5 [rel:CD] [rel:PD] [pos:VVINF] [pos:VAFIN,split:post-hd] [rel:APP] [split:post-hd] [pos:VMFIN] [rel:CJ,split:post-hd] [rel:MO,split:post-hd] [pos:PROAV,rel:MO,split:post-hd] [pos:VVFIN] [rel:OC,split:post-hd]

6 [rel:CC]

7 [pos:NN,split:post-hd]

[pos:VVFIN,rel:root]

0 [pos:ADJD,rel:PD,split:pre-hd] [pos:CARD,split:pre-hd] [pos:PPER,rel:DA,split:pre-hd] [pos:CARD,rel:MO,split:pre-hd] [dist:0,pos:NE,split:pre-hd] [pos:VVPP,split:pre-hd] [rel:JU] [rel:RS,split:pre-hd] [rel:OC,split:pre-hd] [dist:0,pos:PDS,rel:OA,split:pre-hd] [dist:1,rel:OC,split:pre-hd] [dist:0,rel:CJ,split:pre-hd] [pos:VVIZU,split:pre-hd] [pos:ADV,rel:OC] [pos:PWAV,split:pre-hd] [rel:PG] [pos:VVINF,rel:OC,split:pre-hd] [split:pre-hd] [pos:NN,rel:OC] [pos:PWS] [dist:0,pos:KON,rel:OA,split:pre-hd] [pos:VVINF,split:pre-hd] [dist:0,rel:OA,split:pre-hd] [pos:KON,rel:MO,split:pre-hd]

1 [pos:ADJD,rel:MO,split:pre-hd] [pos:PDS,split:pre-hd] [dist:1,pos:NN,split:pre-hd] [dist:0,pos:NE,rel:SB,split:pre-hd] [rel:CP] [pos:ADJA,rel:SB,split:pre-hd] [rel:DA,split:pre-hd] [pos:VVFIN,split:pre-hd] [pos:KON,split:pre-hd]

2 [pos:PPER,split:pre-hd] [rel:SB,split:pre-hd]

3 [dist:0,pos:NN,rel:OA,split:pre-hd] [pos:VVIMP] [dist:0,pos:NN,rel:MO,split:pre-hd]

4 [rel:MO,split:pre-hd]

5 [pos:ADV,rel:MO,split:pre-hd] [pos:FM]

6 [head:yes] [pos:PIAT]

7 [pos:ADV,rel:SB] [rel:CD] [pos:PPER,split:post-hd] [pos:PWAV,split:post-hd] [pos:PTKNEG,rel:MO] [pos:VVINF,split:post-hd] [pos:PDS,split:post-hd] [dist:0,pos:PDS,rel:OA,split:post-hd] [pos:PPER,rel:DA,split:post-hd]

8 [dist:0,pos:PPER,rel:OA] [dist:0,pos:NE,rel:SB,split:post-hd] [pos:PRF]

9 [pos:ADV,rel:MO,split:post-hd] [dist:0,pos:NN,rel:MO,split:post-hd] [rel:DA,split:post-hd] [rel:OG] [pos:VVFIN,split:post-hd] [pos:CARD,rel:MO,split:post-hd] [dist:0,pos:NE,split:post-hd] [rel:SB,split:post-hd]

10 [pos:ADJA,rel:SB,split:post-hd] [pos:VVPP,split:post-hd] [dist:1,pos:KON] [pos:CARD,split:post-hd] [dist:0,rel:OA,split:post-hd] [rel:SVP] [rel:NG] [pos:VVPP,rel:OC] [pos:KON,split:post-hd] [dist:0,pos:KON,rel:OA,split:post-hd]

11 [pos:ADJD,rel:MO,split:post-hd] [dist:0,pos:NN,rel:OA,split:post-hd]

12 [pos:VVIZU,split:post-hd] [pos:ADJD,rel:PD,split:post-hd] [dist:0,pos:ADV,rel:CJ] [dist:0,pos:NN,rel:CJ] [pos:KON,rel:MO,split:post-hd]

13 [pos:PTKVZ] [rel:MO,split:post-hd] [rel:RE]

14 [pos:VAINF] [rel:RC] [dist:1,rel:OC,split:post-hd] [rel:APP] [rel:CC,split:post-hd] [split:post-hd] [rel:DH] [pos:VVINF,rel:OC,split:post-hd] [dist:1,pos:NN,split:post-hd] [rel:OC,split:post-hd]

15 [dist:0,rel:CJ,split:post-hd]

[pos:ADJA]

0	[rel:GL] [pos:PIAT] [rel:NG] [pos:PIS] [rel:CM]
1	[rel:AC]
2	[pos:PDAT] [pos:CARD,split:pre-hd] [pos:ART] [pos:PPOSAT]
3	[pos:CARD,rel:MO] [rel:DA,split:pre-hd] [pos:NE,rel:MO] [pos:ADJD,split:pre-hd] [pos:ADJA,split:pre-hd] [pos:PIDAT]
4	[split:pre-hd] [rel:OA,split:pre-hd] [rel:SBP] [pos:NN,rel:MO]
5	[rel:MO] [pos:ADJD,rel:MO] [pos:ADJD,split:post-hd]
6	[rel:PM] [rel:AMS]
7	[head:yes]
8	[rel:CJ] [rel:CD] [pos:TRUNC]
9	[pos:ADJA,split:post-hd] [pos:CARD,split:post-hd]
10	[pos:NE,rel:NK] [pos:NN,rel:NK]
11	[rel:PG] [split:post-hd] [rel:GR] [rel:OC]
12	[rel:MNR]
13	[rel:RC] [rel:APP]

Bibliography

Anne Abeille, Silvia Hansen-Schirra, and Hans Uszkoreit, editors. *Proceedings of the Workshop on Lingusitically Interpreted Corpora (LINC-03).* x, 2003.

Jason Baldridge. *Lexically Specified Derivational Control in Combinatory Categorial Grammar.* PhD thesis, Edinburgh University, 2002.

Gunnar Bech. *Studien über das deutsche Verbum Finitum.* Munksgaard, no year.

Markus Becker and Anette Frank. A stochastic topological parser for German. In *Proceedings of 19th COLING*, 2002.

Robert C. Berwick. *The acquisition of syntactic knowledge.* MIT Press, 1985.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol, 2002.

Joan Bresnan. *Lexical Functional Syntax.* Blackwell, 2001.

Garry Briscoe and Terry Caelli. *A Compendium of Machine Learning*, volume 1: Symbolic Machine Learning. Ablex, 1996.

Norbert Broeker. A projection architecture for dependency grammar and how it compares to LFG, 1998.

Eugene Charniak. Treebank grammars. In *Proceedings of 13th AAAI*, 1996.

Chomsky. *Lectures on Government and Binding.* Dordrecht, 1981.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings ACL'97*, 1997.

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, 1999a.

Michael Collins. A statistical parser for czech. *ACL*, 1999b.

Walter Daelemans and Vèronique Hoste. Evaluation of machine learning methods for natural language processing tasks. In *Proceedings of LREC 2002 Workshop on Language Resources and Evaluation*, 2002.

B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge UP, 1994.

Ralph Debusmann. A declarative grammar formalism for dependency grammar. Master's thesis, University of the Saarland, Saarbrücken, 2001.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised claaification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

Erich Drach. *Grundgedanken der deutschen Satzlehre.* Wiss. Buchgesellschaft Darmstadt, 1963.

Günther Drodowski and Peter Eisenberg, editors. *Duden Grammatik der deutschen Gegenwartssprache.* Duden, 5. völlig neu bearbeitete und erweiterte auflage edition, 1995.

Denys Duchier and Ralph Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of 39th ACL*, Toulouse, France, 2001.

Peter Eisenberg. *Grundriß der deutschen Grammatik*, volume 2. Der Satz. Metzler, 1999. Kapitel 13. Wortstellung.

Ulrich Engel. Regeln zur wortstellung. In *Forschungsberichte des Instituts für deutsche Sprache*, volume 3, pages 75–148. 1970.

Martin Fowler and Kendall Scott. *UML distilled.* Adison Wesley, second edition edition, 2000.

Michael Gamon, Eric Ringger, Zhu Hang, Robert Moore, and Simon Corston-Oliver. Extraposition: A case strudy in German sentence realization. In *Proceedings of 19th COLING*, pages 301–307, 2002.

E. M. Gold. Language identification in the limit. *Information and Control*, 10: 447–474, 1967.

John H. Greenberg. Some universals of grammar with particular reference to the order of meaningful elements. In John H. Greenberg, editor, *Universals of Language.* MIT Press, 1966.

XTAG Research Group. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

Jan Hajič. Building a syntactically annotated corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning*, pages 106–132. Karolinum, Prague, Czech Republic, 1998.

Hawkins. *A Performance Theory of order and constitutents.* Cambridge UP, 1984.

John A. Hawkins. *Word Order Universals.* Academic Press, 1983.

Erhard W. Hinrichs and Tsuneko Nakazawa. Linearizing AUXs in German verbal complexes. In Erhard Hinrichs and Tsuneko Nakazawa, editors, *Aspects of German VP Structure: An HPSG Account*, number 01-93, pages 25–52. Seminar für Sprachwissenschaft, Universität Tübingen, 1993.

Ursula Hoberg. *Die Wortstellung in der geschriebenen deutschen Gegenwartssprache.* Hueber, 1981.

Tilman Höhle. Der Begriff Mittelfeld: Anmerkungen über die Theorie der topologischen Felder. In Albrecht Schöhne, editor, *Akten des VII. Internationalen Germanisten-Kongresses Göttingen 1985.*, volume 3, pages 329–340. Tübingen: Niemeyer, 1986.

Alan Hutchinson. *Machine Learning.* Clarendon, 1994.

Java. Java 2 platform standard edition version 1.4.1 API specification. http://java.sun.com/j2se/1.4.1/docs/api/index.html, 2003.

Andreas Kathol. *Linear Syntax.* Oxford UP, 2000.

Alexandra Kinyon and Carlos A. Prolo. A classification of grammar development strategies. In *Proceedings of Grammar Development and Engineering Workshop at COLING'02*, 2002.

Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. Computer Science Department Stanford University, no year.

Ron Kohavi and George A. John. Wrappers for feature subset selection. *Artificial Intelligence*, 1–2(97):273–323, 1997.

Christian Korthals and Geert-Jan Kruijff. Unsupervised learning of topological fields. *submitted to ACL'03*, 2003.

Alexander Krotov, Mark Hepple, Robert J. Gaizauskas, and Yorick Wilks. Compacting the penn treebank grammar. In *COLING-ACL*, pages 699–703, 1998. URL `citeseer.nj.nec.com/article/krotov97compacting.html`.

Geert-Jan Kruijff. *A categorial-modal logical architecture of informativity.* PhD thesis, Charles University, Prague, 2001.

Daniela Kurz. Wortstellungspräferenzen im Deutschen. Master's thesis, Universität des Saarlandes, Computerlinguistik, 2000.

K. De Kuthy and W. Meurers. On partial constituent fronting in german, 1999. URL `citeseer.nj.nec.com/dekuthy99partial.html`.

Mitchell Marcus et al. The Ultimate Penn Treebank Bible. Technical Report CD, Linguistic Data Consortium (UPenn), 1995. Technical Report.

Kurt Mehlhorn. *Graph Algorithms and NP-Completeness.* Number 2 in Data Structures and Algorithms. Springer, 1984.

Kurt Mehlhorn. Graph Algorithms and NP-Completeness. Revised online version at www.mpi-sb.mpg.de/~mehlhorn/DatAlgbooks.html, Version 19.10.99, Chapter 4, 1999.

Igor Melčuk. *Dependency syntax: Theory and practice.* Albany: State Univ of NY, 1988.

Tom M. Mitchel. The need for biases in learning generalizations. Technical Report CBM-TR-117, Computer Science, Rutgers University, New Jersey, 1980.

Tom M. Mitchell. *Machine Learning.* WCB/McGraw-Hill, 1997.

Stefan Müller. *Deutsche Syntax deklarativ.* Niemeyer, 1999.

Balas K. Natarajan. *Machine learning. A theoretical approach.* Kaufmann, 1991.

Partha Niyogi and Robert C. Berwick. A language learning model for finite parameter spaces. *Cognition*, (61):161–193, 1996.

John Ross Quinlan. *C4.5: Programs for Machine Learning.* Kaufmann, 1998.

M. Reape. Domain union and word order variation in german. In J. Nerbone, K. Netter, and C. Pollard, editors, *German in Head-Driven Phrase Structure Grammar*, pages 309–331. CSLI Stanford, 1994.

Russel and Norvig. *Artificial Intelligence. A modern approach.* Clarendon, 1995.

Anoop Sarkar and Daniel Zeman. Automatic extraction of subcategorization frames for czech. In *Proceedings of 18th COLING*, 2000.

Scheepers. Linking syntactic functions with thematic roles: Psych-verbs and the resolution of subject-object ambiguity. In B. Hemford and L. Konieczny, editors, *German Sentence Processing*, pages 95–135. Kluwer, 2000.

Anne Schiller, Simone Teufel, and Chritine Stöckert. Guidelines für das tagging deutscher textcorpora mit stts. http://www.ims.uni-stuttgart.de/projekte/corplex/TagSets/stts-1999.pdf.

Petr Sgall, Eva Hajicova, and Jarmila Panenova. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects.* Reidel, 1986.

Wojciech Skut, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. A linguistically interpreted corpus of German newspaper text. In *Proceedings of the ESSLLI Workshop on Recent Advances in Corpus Annotation*, 1998.

Susan Steele. Word order variation. A typological study. In Joseph H. Greenberg, editor, *Universals of Human Language*, volume 2. Syntax, pages 585–623. Stanford UP, 1978.

M. Stone. Asymptotics for and against cross-validation. *Biometrika*, 1977.

Lucien Tesnière. *Elèments de syntaxe structurale.* Klincksieck, 1965.

Hans Uszkoreit. *Word Order and Constituent Structure in German.* CSLI, 1987.

Hans Uszkoreit, Th Brants, Denys Duchier, Brigitte Krenn, Lars Konieczny, Stefan Oepen, and Wojciech Skut. Studien zur performanzorientierten linguistik: Aspekte der relativsatzextraposition im deutschen. *Kognitionswissenschaft*, 7 (3):129–133.

L.G. Valiant. A theory of the learnable. *Communications of the A.C.M.*, 1984.

Aline Villavicencio. The acquisition of word order by a computational learning system. In *Proceedings of the Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop*, pages 209–218. Lisbon, Portugal, 2000.

Fei Xia. Extracting tree adjoining grammars from bracketed corpora. In *Proc of NLPRS-99*, 1999.

Fei Xia, Chung hye Han, Martha Palmer, and Aravind Joshi. Automatically extracting and comparing lexicalized grammars for different languages. In *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 2001.

Naoki Yoshinaga and Y. Miyao. Grammar conversion from LTAG to HPSG. In *European Student Journal on Language and Speech, Special Issue of 6th ESSLLI Student Session*, 2002.

Zdenek Zabokrtsky. Automatic functor assignment in the Prague Dependency Treebank. Master's thesis, Charles University Prague, 2001.