WOLFGANG MAIER

PARSING DISCONTINUOUS STRUCTURES

PARSING DISCONTINUOUS STRUCTURES

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der Philosophie
der Philosophischen Fakultät der
Eberhard Karls Universität Tübingen

VORGELEGT VON

Wolfgang Maier aus Göppingen

2013

Gedruckt mit Genehmigung der Philosophischen Fakultät
der Eberhard Karls Universität Tübingen

DEKAN:
Prof. Dr. Jürgen Leonhardt

HAUPTBERICHTERSTATTER:
Prof. Dr. Laura Kallmeyer, Universität Düsseldorf

MITBERICHTERSTATTER:
Prof. Dr. Erhard Hinrichs, Universität Tübingen
PD Dr. Frank Richter, Universität Tübingen

TAG DER MÜNDLICHEN PRÜFUNG:
16. Oktober 2012

TOBIAS-lib, Tübingen

ABSTRACT

The development of frameworks that allow to state grammars for natural languages in a mathematically precise way is a core task of the field of computational linguistics. The same holds for the development of techniques for finding the syntactic structure of a sentence given a grammar, *parsing*. The focus of this thesis lies on *data-driven parsing*. In this area, one uses probabilistic grammars that are extracted from manually analyzed sentences coming from a treebank. The probability model can be used for disambiguation, i. e., for finding the best analysis of a sentence.

In the last decades, enormous progress has been achieved in the domain of data-driven parsing. Many current parsers are nevertheless still limited in an important aspect: They cannot handle discontinuous structures, a phenomenon which occurs especially frequently in languages with a free word order. This is due to the fact that those parsers are based on Probabilistic Context-Free Grammar (PCFG), a framework that cannot model discontinuities.

In this thesis, I propose the use of Probabilistic Simple Range Concatenation Grammar (PSRCG), a natural extension of PCFG, for data-driven parsing. Thereby, I bring together developments from different areas, namely research on parsing German, on the quantification of discontinuity in treebanks, and on formalisms which can model discontinuous structures. Not only theoretical aspects are treated. For the first time, all techniques for direct data-driven parsing of discontinuities have been implemented and tested in a real-world data-driven parsing setting. The parser output quality and the parsing speed are encouraging and prove the point of this work: An exploration of the landscape of formal grammars beyond Context-Free Grammar with regard to data-driven parsing is worth the effort for data-driven parsing and opens the way for many new developments in the future, both in parsing and beyond.

ZUSAMMENFASSUNG

Die Entwicklung formaler Systeme für die mathematisch präzise Formulierung einer Grammatik der natürlichen Sprache ist eine Kernaufgabe der Computerlinguistik. Dasselbe gilt für die Entwicklung von Techniken für die Berechnung der syntaktischen Struktur eines Satzes auf Basis einer Grammatik, *Parsing*. In dieser Dissertation steht das *datengetriebene Parsing* im Vordergrund. Dabei werden von den manuell annotierten Sätzen einer Baumbank extrahierte probabilistische Grammatiken benutzt. Das Wahrscheinlichkeitsmodell kann zur Disambiguierung benutzt werden, d. h. zur Berechnung der besten Analyse für einen gegebenen Satz.

In den vergangenen Jahrzehnten wurden in der Forschung zu datengetriebenem Parsing enorme Fortschritte erzielt. Nichtsdestotrotz sind viele aktuelle Parser weiterhin in einem wichtigen Aspekt beschränkt: Sie können nicht mit diskontinuierlichen Strukturen umgehen, einer Art von Strukturen die besonders in Sprachen mit freier Wortstellung häufig auftritt. Der Grund hierfür ist, dass diese Parser auf Probabilistischer Kontextfreier Grammatik aufbauen, welche Diskontinuitäten nicht modellieren kann.

In dieser Dissertation plädiere ich für die Benutzung von *Probabilistic Simple Range Concatenation Grammar*, einer natürlichen Erweiterung der Probabilistischen Kontextfreien Grammatik, für datengetriebenes Parsing. Ich führe damit Entwicklungen aus verschiedenen Bereichen zusammen: Forschung zum Parsing des Deutschen, zur Quantifizierung von Diskontinuität in Baumbanken, und zu Formalismen, die diskontinuierliche Strukturen modellieren können. Nicht nur theoretische Aspekte werden behandelt. Zum ersten Mal wurden alle Techniken, die für direktes datengetriebenes Parsing von Diskontinuitäten benötigt werden, implementiert und auf einem realistisch großen Datensatz getestet. Die Qualität der Parserausgabe und die Parsinggeschwindigkeit sind ermutigend und sprechen für den Ansatz dieser Arbeit: Eine Erforschung der Landschaft der formalen Grammatiken jenseits der Kontextfreien Grammatik lohnt sich für das datengetriebene Parsing und ebnet den Weg für viele Entwicklungen in der Zukunft, im Parsing, und darüber hinaus.

功遂身退，
天之道載。

Daodejing (Wang Bi), ch. 9, 9-10
Laozi

ACKNOWLEDGMENTS

During the last years, many people have helped me to achieve the completion of this work. While I extend my gratitude to all of them, I want to mention those who particularly stand out.

In the first place, I wish to express my sincere thanks to my supervisor Laura Kallmeyer. She motivated and encouraged me at all times in the development and elaboration of my ideas in every way I could have wished for. I am deeply grateful for her support and patience – I could not have received a better supervision.

I very much enjoyed my time in the "Villa". Thanks to my office mate Timm Lichte, for countless fruitful academic discussions, and also for the enjoyable off-time. Thanks to Yannick Parmentier for our productive teamwork on TuLiPA, and beyond. Thanks to Anders Søgaard for our collaboration and the many discussions which sparked my work on statistical parsing. Thanks to Giorgio Satta for our collaboration and for giving my work a firm direction. I thank Kilian Evang for his invaluable support with the implementation of the statistical parser. Thanks also to Andreas van Cranenburgh, Armin Buch and Gerhard Jäger for discussions.

I am grateful for the support I have received from members and ex-members of the University of Tübingen. Thanks to Erhard Hinrichs and Sandra Kübler for our collaboration. Thanks also to Frank Richter and Johannes Kabatek! Finally, thanks to the best secretary in the world, Beate Starke, for making all administrative tasks a breeze.

I consider myself very lucky to have been given the chance to travel to many conferences around the world. Out of the many kind people I have met there, I would like to thank some people in particular for both academic and non-academic inspiration: Éric de la Clérgerie, Carlos Gómez Rodríguez, Arezoo Islami, Marco Kuhlmann, Lilja Øvrelid, Owen Rambow, and Anoop Sarkar.

I am grateful that I could finish my work at the University of Düsseldorf among nice and supportive colleagues. In particular I would like to thank Miriam Kaeshammer.

Even though it is said to be difficult, I did succeed in having an exciting life outside the university buildings as well. Two people had a big part in that. Thank you, Solange, for taking care of me, and thank you, Joachim, for reminding me that there is more to life than work!

Last but not least, I would like to express my gratitude to my family, for their support throughout this time, and for having brought me where I stand now.

Tübingen, im Frühjahr 2013

Wolfgang Maier

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1 INTRODUCTION

This thesis aims at clarifying the status of discontinuous structures in treebank annotation, at providing both symbolic and probabilistic parsing techniques for formalisms which can model them, and at confirming the usability of the probabilistic parsing techniques for data-driven parsing. In the following section, I outline the background of these research topics and provide motivation for my work. Section 1.2 summarizes the contributions of this thesis, and section 1.3 presents an overview of the following chapters.

1.1 BACKGROUND

The problem of computationally determining whether a sentence is admissible given a certain linguistic theory (*recognition*), as well as the problem of finding a structure for an admissible sentence which represents the abstraction of the linguistic theory (*parsing*) are two core research topics in computational linguistics. Both require linguistic theories to be stated in a mathematically rigorous way. An appropriate type of frameworks for this task is Generative Enumerative Syntax (GES) (Pullum and Scholz, 2001). Frameworks within GES typically consist of a *grammar formalism* which allows to state a finite set of rules (a *grammar*), with which an infinite set of strings (a *language*) can be *generated*. A series of rule applications is called a *derivation*, the structure resulting from recording the rule applications a *parse tree*.

Within the context of a grammar formalism, given a grammar, solving the recognition problem for a sentence amounts to determining if the grammar allows for the generation of the sentence. In order to solve the parsing problem, one must additionally keep track of the corresponding sequence of rule applications, i. e., one must find a parse tree for the sentence. Rules can be equipped with probabilities for *probabilistic parsing* (in contrast to *symbolic* parsing). The probabilities can be used for disambiguation, i. e., to determine the most likely parse tree of a sentence. Parsing with a probabilistic grammar obtained from

a *treebank* (a text collection annotated with linguistic information) is termed *data-driven parsing*.

From the point of view of a computational linguist, the choice of a formalism for the implementation of a linguistic theory depends on two parameters, its expressivity and its computational complexity. The expressivity of a formalism can be characterized in terms of the class of languages it generates. This property is called its *weak generative capacity*. Furthermore, it can be characterized via the class of parse trees it allows for. This property is called its *strong generative capacity*. Ideally, from a weakly generative point of view, a formalism should be capable of generating all and only those strings which constitute admissible sentences, while from a strongly generative point of view, it should provide all and only those structures which represent the abstractions of the underlying linguistic theory. *Overgeneration*, i. e., generating strings which are not admissible sentences, is thereby less problematic than *undergeneration*, especially in a probabilistic setting, where probabilities can help with disambiguation.

A higher expressivity generally implies an increased computational complexity with respect to recognition. Therefore, the central challenge in practice is to find a good compromise between expressivity and computational tractability.

1.1.1 *Discontinuous Structures and Syntactic Description*

Intuitively speaking, a *discontinuous structure* in a natural language sentence is a sequence of words which is discontinuous but forms a linguistically meaningful unit. Discontinuous structures are of particular interest because they are difficult to model. In the following, I present the corresponding issues in connection with both types of GES frameworks I use in this thesis, namely, constituency-based frameworks and dependency-based frameworks.

Constituency-based frameworks allow for the description of the hierarchical structure of constituent phrases (*constituents*) in a sentence. A constituent is a sequence of words which form a linguistically meaningful unit. It is labeled with its linguistic type, e. g., *noun phrase (NP)*, *verb phrase (VP)*, etc. A hierarchical structure of constituents is called a *constituency structure*. As a first framework of this type, Context-Free Grammar (CFG) was proposed (Chomsky, 1956). CFG allows for the modeling of constituency structures on the basis of *continuous* con-

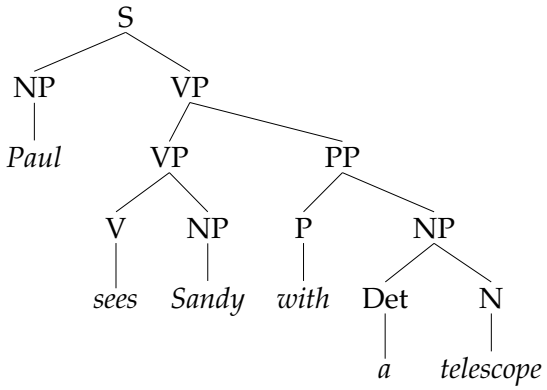


Figure 1: A CFG parse tree example

stituents, i. e., of constituents consisting of a continuous sequence of words. Figure 1 shows a possible analysis (a CFG parse tree) of an English sentence. Note that the sentence can be obtained by reading the leaves of the tree from left to right.

Several parsing techniques for CFG have appeared in the literature, such as the CYK algorithm (Cocke and Schwartz, 1970; Younger, 1967; Kasami, 1965), the Earley algorithm (Earley, 1970), and Left-Corner Parsing (Rosenkrantz and Lewis, 1970). They have served as a basis for the development of parsing algorithms for other formalisms.

In the early 80s, it was confirmed that CFG does not provide enough expressivity to model natural language. Data with so-called *cross-serial dependencies* were brought up (Huybregts, 1984; Shieber, 1985). These are configurations in which constituents are intertwined such that they mutually interrupt themselves. An example for a sentence which exhibits such dependencies is the Swiss German sentence (1), taken from Shieber (1985). The cross-serial dependencies in this sentence arise from the fact that a linguistically adequate modeling requires to associate the noun phrases with their corresponding verbs.

(1) ... mer em Hans$_i$ es Huus$_j$ hälfed$_i$ aastriiche$_j$
... we Hans-DAT the house-ACC help paint
... we help Hans paint the house

Note that such dependencies can be iterated, as shown in (2).

(2) 
...mer d'chind$_i$ em Hans$_j$ es Huus$_k$ lönd$_i$ hälfe$_j$ aastriiche$_k$
...we the kids Hans the house let help paint
...we let the kids help Hans paint the house

Shieber formally showed that neither the weak nor the strong generative capacity of CFG is enough to model cross-serial dependencies. In fact, more generally, no *non-local* or *long-distance dependencies* can be modeled. These are configurations in which two elements are separated but belong together. They are not a rare phenomenon, particularly in languages with a relatively free word order, such as German. Consider the German sentence (3) as a more general example. It contains a *discontinuous constituent*: The VP *Darüber nachgedacht* is interrupted since *Darüber* is fronted.

(3) Darüber muss nachgedacht werden
Thereof must thought be
"Thereof must be thought."

In order to model discontinuous constituents, we need a formalism with an expressivity beyond Context-Free Grammar. Since, as already mentioned, higher expressivity entails an increased computational complexity of recognition and parsing, it is desirable to characterize the minimal necessary amount of expressivity beyond Context-Free Grammar. *Mild Context-Sensitivity (MCS)* (Joshi, 1985) is an attempt of such a characterization. Informally speaking, a set of languages is mildly context-sensitive if it contains all context-free languages, if cross-serial dependencies are allowed to a certain extent, if the recognition problem is tractable for all languages,[1] and if the lengths of the words of all languages grows linearly. If the set of all languages which can be generated with grammars of a certain formalism is mildly context-sensitive, we call this formalism a *mildly context-sensitive formalism* or *MCS formalism*.

In the literature, such formalisms have been proposed for grammar implementation. One of them is Tree-Adjoining Grammar (TAG) (Joshi et al., 1975). A TAG consists of a set of *elementary trees*, small phrase structure templates which represent the grammatical structure (e. g., the subcategorization frame) that comes with a single lexical

1 More precisely, the recognition problem for all languages must be solvable in polynomial time (see section 2.3.2).

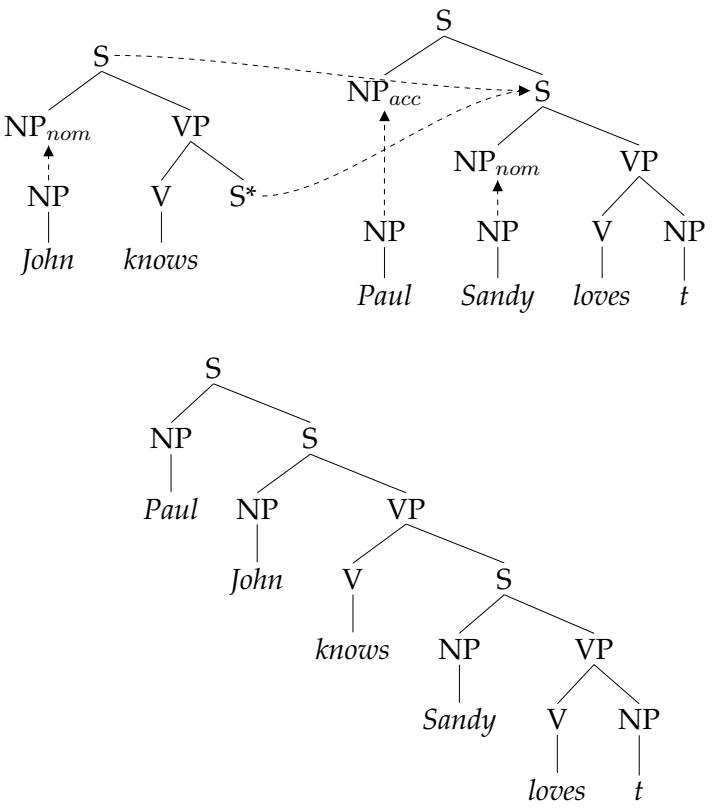


Figure 2: A TAG parse tree example

item. There are two operations for the combination of elementary trees. The *substitution* operation allows to plug a tree into the leaf of another tree. The *adjunction* operation allows to insert elementary trees of a certain type (*auxiliary trees*) at an internal node of another tree. As an example, figure 2 shows a simplified TAG analysis for a sentence with an extracted object NP. One elementary tree encodes the subcategorization requirements of the matrix verb *loves* with an extracted object. NP trees fill in the argument slots by substitution. The auxiliary tree for *knows* gets adjoined at the lower S node. Adjunction is the key feature of TAG, since it gives it its extended domain of locality: Elements which are encoded in a single elementary structure can end up arbitrarily far away from each other in the final derived tree (in our example the extracted object and the matrix sentence).

Since there are linguistic phenomena which cannot be modeled with TAG, e. g., scrambling in German (Becker et al., 1991), variants of

TAG with a higher expressivity have been introduced, such as different variants of Multi-Component Tree-Adjoining Grammar (MCTAG). TAG and its variants have been used for the grammar implementation in various languages, notably English (XTAG Research Group, 2001), French (Abeillé et al., 2003), Korean (Han et al., 2000) and German (Lichte, 2007; Kallmeyer et al., 2008b).

Various parsing algorithms for TAG have appeared in the literature. Some of them are TAG adaptations of the corresponding CFG parsers, e. g., TAG CYK (Vijay-Shanker and Joshi, 1985) and Earley-style parsers (Schabes and Joshi, 1988; Nederhof, 1997, 1999). Other work takes advantage of the fact that parsing with some formalisms which are equivalent to TAG is conceptually easier than parsing with TAG itself. Boullier (1996), e. g., presents a TAG parser which relies on Linear Indexed Grammar (LIG); i. e., he uses LIG as a *pivot formalism*.

Linear Context-Free Rewriting System (LCFRS) (Vijay-Shanker et al., 1987) and the equivalent formalism Multiple Context-Free Grammar (MCFG) (Seki et al., 1991) are other formalisms which are mildly context-sensitive. Their expressivity is higher than the expressivity of TAG. In contrast to TAG, they constitute a somewhat natural extension of Context-Free Grammar. They offer an extended domain of locality by dropping the requirement that every constituent must have a single continuous span. As an example, figure 3 shows a simplified possible LCFRS/MCFG parse tree of (3). While, to my knowledge, the formalisms have not been used for grammar implementation, they have found their share of attention in the parsing community (Seki et al., 1991; Nakanishi et al., 1997; Burden and Ljunglöf, 2005). They have furthermore been used in a pivot role for parsing TAG and variants of TAG (Kallmeyer et al., 2008b).

LCFRSs are not only interesting with respect to frameworks based on constituency, but also with respect to *dependency-based frameworks*. In such frameworks, syntactic description is based on directed word-to-word relations, so-called *dependencies*. They are typically organized such that each word of a sentence has exactly one incoming edge from its *head*, and one or more outgoing edges to its *dependents*. Dependencies can have labels describing their linguistic type. As an example, figure 4 shows a dependency analysis (a *dependency structure*) for (3). I adopt the graphical representation of dependencies from recent literature (Kuhlmann, 2007). Each word is represented by a circle, dependencies are drawn as arrows between those circles, and the relation

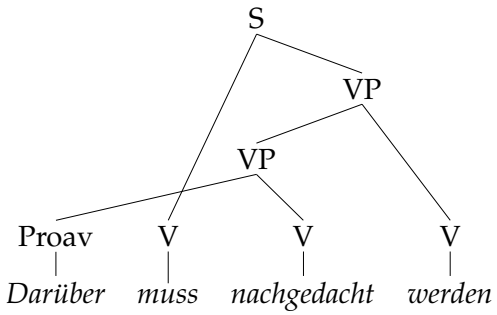


Figure 3: A LCFRS parse tree example

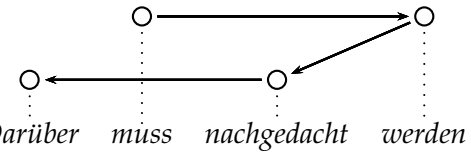


Figure 4: A dependency structure example

between words and circles is indicated by dotted lines. The use of dependencies for syntactic description has a long-standing tradition. It can be traced as far back as to the Sanskrit grammarian Pāṇini (4th century BC) (Kruijff, 2002). Contemporary dependency frameworks, which I will subsume under the term Dependency Grammar (DG), are for instance Meaning Text Theory (MTT) (Mel'čuk, 1988), Weighted Constraint Dependency Grammar (WCDG) (Foth et al., 2004) and eXtensible Dependency Grammar (XDG) (Debusmann et al., 2004). They are all based on Lucien Tesnières pioneering work (Tesnière, 1959).

In this thesis, I treat DG as belonging to Generative Enumerative Syntax. Under this view, a grammar consists of a finite set of rules which, instead of describing constituent relationships, describe word-to-word dependencies or bundles of such dependencies (Hays, 1964; Gaifman, 1965). Again, the choice of a formalism for formulating such a grammar determines the class of dependency structures we can obtain. Gaifman (1965) shows the equivalence of CFG and the class of *projective dependency structures*. In those structures, each word and its dependents form a continuous sequence of words. However, for many linguistic phenomena, *non-projectivity* is needed. None of the current

dependency frameworks requires projectivity. Our example in figure 4 is also non-projective, since the word (*nachgedacht*) is separated from its dependent (*Darüber*). Unrestricted non-projectivity results in intractable recognition (Neuhaus and Bröker, 1997). For this reason, researchers have attempted to find a DG equivalent for Mild Context-Sensitivity. *Mild non-projectivity* is such a characterization (Kuhlmann, 2007). On its basis, Kuhlmann and Satta (2009) show how to model restricted non-projective dependencies in a straight-forward way using LCFRS.

Some linguistic phenomena have been argued to lie beyond the expressive power of LCFRS. Becker et al. (1992) mention scrambling of arguments in German. They argue that there is no bound on the number of arguments to be scrambled and on the distance over which arguments can be scrambled. This impedes the modeling of scrambling with LCFRS. Consider (4) as an example. In this sentence, almost all complement orders are acceptable.

(4) ... dass [dem Kunden]$_i$ [den Kühlschrank]$_j$ bisher noch
... that [the client]-DAT [the fridge]-AKK so far yet
niemand t_i [[t_j zu reparieren] versprochen] hat
nobody t_i [[t_j to repair] promised] has
"... that so far, nobody has promised the client to try to repair the fridge"

Range Concatenation Grammar (RCG) (Boullier, 1998) is a formalism with which scrambling and other phenomena of this type can potentially be modeled (Boullier, 1999). Grammar development with RCG is also feasible (Sagot, 2005). The key feature of RCG is that it allows for copying operations, i. e., it allows for a substring to be part of two constituents at the same time. Figure 5 shows an RCG analysis for a case of right node raising. The fact that *bananas* is shared by both VPs can be expressed directly. RCG has several interesting formal properties. It is not mildly context-sensitive. Nevertheless, its recognition problem is not computationally intractable (Bertsch and Nederhof, 2001). Simple Range Concatenation Grammar (SRCG), a variant of RCG with less expressivity, is equivalent to LCFRS and MCFG. RCG (and SRCG) can also take the role of pivot formalisms for parsing, since for many formalisms, their grammars can easily be formulated as equivalent RCGs (Boullier, 1998). Literature on RCG parsing algorithms is sparse. Basic

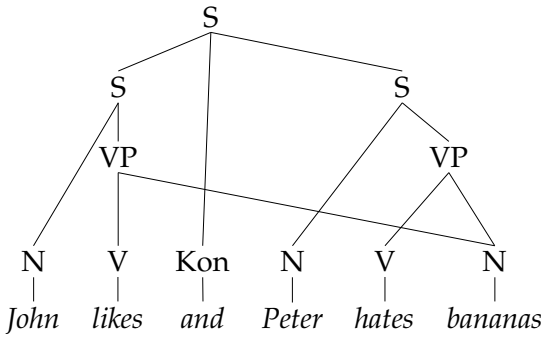


Figure 5: A RCG parse tree example

algorithms have been presented by Boullier (1998) and Barthélemy et al. (2001).

As a summary, figure 6 shows simplified parse tree schemas of CFG, TAG, LCFRS and RCG in order to visualize their different domains of locality. The domain of locality of each formalism is characterized by the type of interruption of constituents it allows for. CFG allows for the insertion of one continuous string in a constituent (β in the example). The adjunction operation of TAG delivers more expressivity, since it allows for the simultaneous insertion of two strings (β_1 and β_2 in the example). LCFRS allows for the insertion of an arbitrary number strings. In the example, the interrupting strings are β and γ. However, other strings $\delta_1, \ldots, \delta_n$ could interrupt each of α_1, α_2 and α_3. RCG, as mentioned above, additionally allows for copying operations. In the example, α_3 is part of both the B and the C constituent.

1.1.2 *Discontinuous Structures and Data-Driven Parsing*

Treebanks are text collections (*corpora*) with linguistic annotation. The data sources of the treebanks can stem from different domains, ranging from newspapers over transcribed spoken language to historical texts. The annotation can have various description levels corresponding to different linguistic modules, e. g., semantic role annotation, syntactic annotation, morphological annotation or phonological annotation. The syntactic annotation, which is relevant for this thesis, is generally either based on dependency grammar (in a *dependency treebank*) or on a constituency-based description (in a *constituency treebank*). The trees of a treebank can be interpreted as the parse trees of a latent

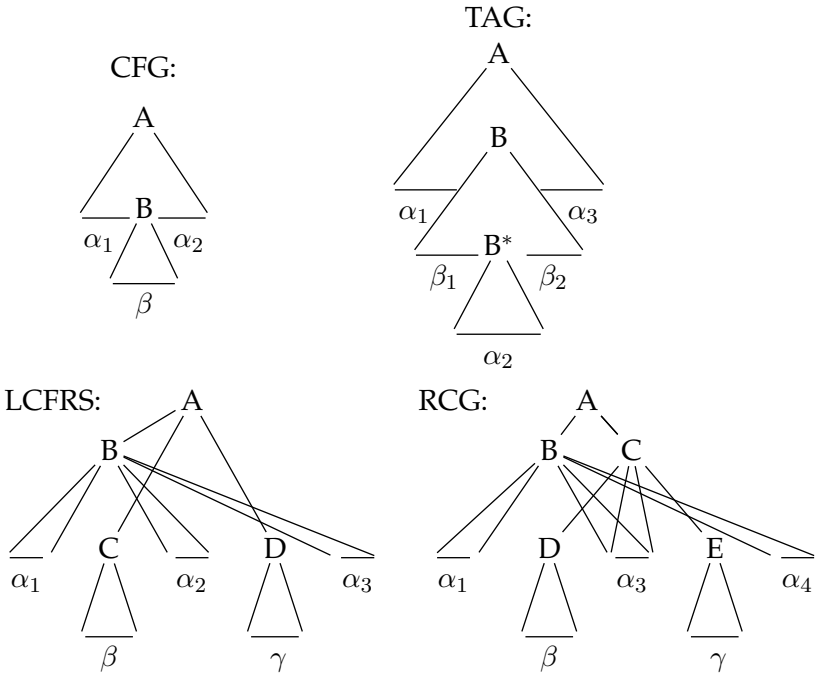


Figure 6: Locality in various formalisms

grammar. Given an appropriate algorithm, this grammar can be reconstructed. Since treebanks only deliver a limited amount of data, one does not interpret the extracted grammars as the implementation of some linguistic theory. Instead, during the extraction, one estimates the occurrence probabilities of grammar rules, and uses them to determine the most likely parse tree(s) of a sentence. This approach, as already mentioned, is called *data-driven parsing*.

Treebank annotation must account for discontinuous structures. In most constituency treebanks, the syntactic annotation takes the form of CFG parse trees, which do not show crossing branches (cf. fig. 1). Discontinuity is represented with an add-on mechanism, such as labeling conventions. The motivation behind this approach is that trees without crossing branches are easier to process: When discarding the add-on mechanism, one only needs the expressivity of CFG. Examples for such treebanks are the Penn Treebank (PTB) (Marcus et al., 1993) and the German Tübingen Treebank of Written German (TüBa-D/Z) (Telljohann et al., 2006).

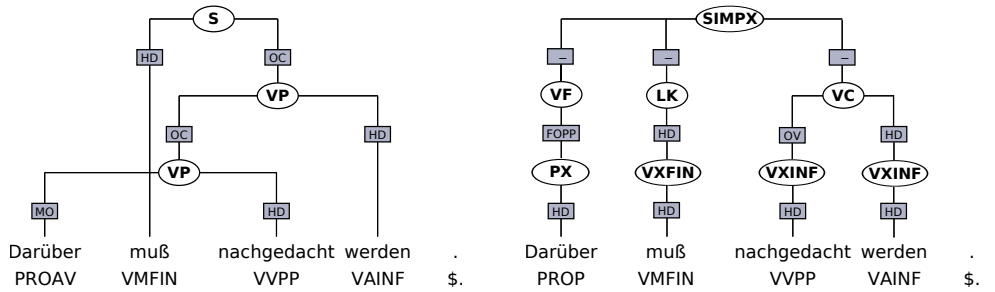


Figure 7: Annotation of discontinuous constituents

In a few treebanks, such as in the German TIGER (Brants et al., 2002) and NeGra (Skut et al., 1997) treebanks, and in the Bulgarian BulTreebank (BTB) (Osenova and Simov, 2004), a different approach is pursued. Discontinuities are represented with crossing branches. As an example, consider figure 7 which contrasts the NeGra annotation of (3) (left) with a TüBa-D/Z-style annotation of the same sentence (right). While in the former, the discontinuous VP is annotated directly, in the latter, the edge label FOPP (facultative PP object) establishes the connection between *Darüber* and *nachgedacht*.

The situation is different for dependency treebanks. Dependency treebanks such as the Prague Dependency Treebank (PDT) (Hajič et al., 2000) or NeGra dependency treebank (NeGra-Dep) (Daum et al., 2004) allow both projective and non-projective dependencies. The dependency tree in figure 4 is actually taken from the NeGra dependency treebank.

Interpreting constituency trees without crossing branches as the parse trees of a Probabilistic CFG (PCFG) is a natural choice, since reading off a grammar and estimating the probabilities is immediate. Charniak (1996) uses a PCFG extracted from the Wall Street Journal part of the PTB to show that such grammars give parsing results of acceptable quality. Charniak's data have been and continue to be the basis for most newly developed probabilistic constituency parsers. Milestones are the work of Michael Collins (Collins, 1999), the work of Mark Johnson (Johnson, 1998), the work of Eugene Charniak (Charniak, 2000), the work of the Stanford NLP Group (Klein and Manning, 2003a,b,c), and the work of the Berkeley NLP Group (Petrov et al., 2006; Petrov and Klein, 2007; Petrov, 2009), which is the current state-of-the-art.

For data-driven constituency parsing with PCFG, non-local information cannot be used directly, i. e., syntactic dependencies which cannot be expressed with CFG get lost. However, with methods other than plain PCFG parsing, this information can be used nevertheless.

- Non-local information can be used for the improvement of the performance of parsers which produce trees without crossing branches. Within this paradigm, one can recur to formalisms which have a higher expressivity than CFG but produce parse trees which, on the surface, do not exhibit crossing branches. Non-local dependencies can be encoded in complex categories, such as in Combinatory Categorial Grammar (CCG) (Steedman, 2000; Hockenmaier, 2003), or by derivational mechanisms, such as in TAG (see above) (Chiang, 2003).

- Non-local information can be used directly in parsers which can build parse trees that directly encode non-local dependencies. This can be done by augmenting PCFG parses in a pre- or post-processing step, with techniques such as pattern matching (Johnson, 2002) or machine learning methods (Levy and Manning, 2004). Another possibility is the use of formalisms which represent crossing branches directly. Plaehn (2004) uses Probabilistic DPSG (PDPSG) (Bunt et al., 1987); Levy (2005) uses a Probabilistic MCFG (PMCFG) parser.

While there exists work on grammar-based parsing of projective dependencies (Nasr and Rambow, 2004; Bangalore et al., 2009), to my knowledge, there exists no grammar-based parser for non-projective dependencies. Dependency parsers which can handle non-projectivity, such as the MaltParser (Nivre et al., 2007) and the MSTParser (McDonald et al., 2005), do not rely on grammar formalisms.

1.2 CONTRIBUTIONS

1.2.1 *What this Thesis is About*

The focus of this thesis lies on symbolic and probabilistic parsing with grammar formalisms that can model discontinuous structures in Natural Language.

With respect to non-probabilistic applications, Range Concatenation Grammar is a formalism with interesting properties. Its expressivity exceeds Mild Context-Sensitivity, which makes it a potential candidate for the modeling of various complex natural language phenomena lying beyond MCS. Additionally, grammar development with RCG is feasible. At the same time, RCG is still fairly efficiently processable. Still, not many parsing algorithms can be found in the literature. I will therefore present different parsing strategies for RCG, formulated as deduction systems. Simple RCG, which is equivalent to LCFRS and MCFG, is also an interesting formalism which has already proven useful for parsing TAG in a pivot role. I will present a novel Earley strategy for SRCG, together with optimization techniques.

The area of data-driven probabilistic parsing with constituency data has been and continues to be dominated by Probabilistic Context-Free Grammar. This is mainly due to treebank annotation practices: Syntactic annotation generally takes the form of CFG parse trees, extended by an additional mechanism that handles discontinuities. For parsing, this information is then simply discarded. This is especially undesirable for languages with a free word order, where discontinuities occur frequently. There is work which aims at reconstructing discontinuities, most of it on the basis of a PCFG parser equipped with pre-, resp. post-processing; however, very rarely, formalisms are used which can encode discontinuities directly. I follow up on previous work by Levy (2005) and dedicate the main part of this thesis to the presentation of the first data-driven parser for Probabilistic SRCG (PSRCG), to be used for the parsing of both discontinuous constituents and non-projective dependencies. The parser uses optimization techniques, such as novel context-summary estimates for PSRCG A^* parsing. A treebank study shows that SRCG is a good candidate for modeling discontinuities. An evaluation on real-world data from the Czech Prague Dependency Treebank and the German NeGra treebank yields promising results and shows that parsing can be done in an acceptable time.

PSRCG, a natural extension of CFG, provides an intuitive way of direct parsing of discontinuities. Grammars can be obtained immediately from treebanks with directly annotated discontinuities, without the need of using linguistic knowledge. Techniques from PCFG parsing can be transferred to PSRCG parsing. However, PSRCGs behave differently from PCFGs – the best parameters for a parsing model can only be determined in practice. A parser implementation is publicly

available at http://www.wolfgang-maier.net/rparse/. It constitutes a major novelty, since it is the first published parser that directly creates overt discontinuities and delivers evaluable output on real-world sized data sets; in other words, it is the first time that the feasibility of data-driven parsing with discontinuities is shown at all. The implementation is innovative since it brings together developments from different areas, namely recent research on parsing German, on quantification of discontinuous structures in treebanks, and research on formalisms which can model them. Implementing the parser was a highly non-trivial enterprise. Important challenges had to be overcome, on both a practical and a theoretical level. The properties of the extracted grammars had to be investigated, program design decisions had to be taken such that the higher parsing complexity could be handled, and new techniques for optimization had to be developed.

To sum up, this thesis makes the following contributions.

- A novel Earley-style parsing algorithm for RCG with optimization techniques, as well as an Earley-style parsing algorithm for SRCG, both with empirical evaluations.
- A grammar extraction algorithm for treebanks with discontinuous annotation which produces SRCGs, as well as a formal characterization of discontinuity and synchronous rewriting in constituency treebanks and in grammars extracted from those treebanks with measures relating to the corresponding measures in DG.
- Techniques and algorithms for parsing Probabilistic SRCG, such an algorithm for weighted deductive parsing, binarization algorithms, markovization strategies and context summary estimates of parse items for A^* parsing, resp. best-first parsing.
- Evaluation methods for data-driven Probabilistic SRCG parsing and qualitative results, moreover, first results on grammar-based non-projective dependency parsing using Probabilistic SRCG.

1.2.2 *What this Thesis is not About*

The research presented in this thesis could easily be expanded in different directions. In favor of a clearly delimited scope, some topics will not be discussed.

- Discontinuous structures are in the focus of this thesis. However, arguing in favor or against a particular linguistic theory of them is not subject of this work.
- I will make use of formalisms with an expressivity beyond CFG. However, the choice of an adequate formalism for modeling natural language, especially with regard to phenomena which need expressivity beyond context-freeness, is not subject of this work.
- With regard to formalisms beyond CFG, Model-Theoretic Syntax (MTS) (Pullum and Scholz, 2001) will not be used. I limit myself to the GES paradigm, particularly to MCS formalisms.
- In order to formulate certain algorithms, I will use the paradigm of parsing as deduction (Shieber et al., 1995). An exploration of the formal machinery behind deduction systems is not subject of this work.
- With respect to the contributions of this thesis to the field of probabilistic parsing, major developments in the literature are presented in chapter 4. Certain techniques are transferred to PSRCG parsing in chapter 6. In principle, nothing stands in the way of applying the remaining techniques as well, in particular lexicalized parsing and k-best parsing. This is left for future work.

1.3 OVERVIEW

In chapter 2, I present basic mathematical definitions and terminology. I furthermore introduce the relevant grammar formalisms and give the basic definitions for symbolic parsing.

In chapter 3, I present an Earley-style parsing algorithm for RCG, an incremental parsing strategy for SRCG and an experimental evaluation of both.

Chapter 4 is dedicated to an introduction to probabilistic parsing and an overview of data-driven parsing. I give an overview of the development of probabilistic constituency parsing and present the limitations of previous approaches. These limitations provide further motivation for my work. Furthermore, I introduce relevant work in the field of data-driven dependency parsing and provide motivation for switching to a grammar-based approach.

In chapter 5, I investigate in detail which linguistic phenomena cause discontinuity and analyze the annotation of constituency treebanks and dependency treebanks which accounts for them. I present a grammar extraction algorithm for SRCG for both discontinuous constituency treebanks and non-projective dependency treebanks. I then show for both how their degree of discontinuity can be characterized in terms of the two measures of *gap degree* and *well-nestedness*. Moreover, I investigate to what extent treebank annotation exhibits synchronous rewriting.

In chapter 6, I present the adaptation of various techniques for PCFG parsing for the parsing of Probabilistic SRCG. These include methods for binarization and markovization, a CYK style algorithm for weighted deductive parsing of SRCG, and a series of context summary estimates of parse items which are used to speed up parsing.

In chapter 7, I apply the techniques presented in the preceding chapter to the German NeGra treebank. In order to provide a meaningful evaluation, I discuss two different evaluation measures. A discussion of the experimental results is presented which, one the one hand, clarifies the interaction of the different parser parameters, and on the other hand highlights the differences between a probabilistic SRCG model and a PCFG model.

In chapter 8, I apply my parser to dependency treebanks. While the results do not reach the state of the art, they are the first reported results for grammar-based parsing of non-projective dependencies.

Chapter 9 concludes this thesis and presents perspectives for future work.

Note that there is ample related work on every aspect of this thesis. It will be presented throughout the thesis in the respective relevant context. Furthermore, note that on the beginning of each chapter, it is indicated if and where material has been published previously.

2

DEFINITIONS

In this chapter, I introduce the mathematical basis of this dissertation. Section 2.1 contains basic definitions. Section 2.2 introduces definitions of grammar formalisms and related definitions necessary for formalizing grammatical description. Section 2.3 contains the basic definitions for symbolic parsing.

2.1 BASIC DEFINITIONS

In this section, I define basic mathematical concepts which are relevant for my work. Whenever possible, I follow standard notation and definitions (Hopcroft and Ullman, 1979; Schöning, 2001).

2.1.1 *Sets, Alphabets, Words, and Languages*

Sets

$\mathbb{N}$ denotes the set of positive natural numbers, excluding 0. We set $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A total order on natural number is given by the relation $\leqslant$. $\mathbb{R}$ denotes the set of real numbers.

As usual, sets are notated with upper case letters. For a set A, we write $|A|$ for its *cardinality* and $\mathfrak{P}(A)$ for its *powerset*, the set which contains all of its subsets. If A is infinite, $\mathfrak{P}_{\text{fin}}(A) \subseteq \mathfrak{P}(A)$ designates the set of all finite subsets of A. We introduce the functions *min* and *max* which yield the least, resp. the greatest element of A with respect to a given total order $\leqslant$ on A. A *partition* of A is a set of pairwise disjoint non-empty subsets of A (*blocks*) the union of which is exactly A. A partition of A is given by an equivalence relation R on A which is such that for all $a, b \in A$, a and b are in the same block iff $a \sim_R b$. Eventually, $\emptyset$ denotes the *empty set*.

We first define an inventory of symbols.

Alphabets and Words

DEFINITION 2.1 (Alphabet). An *alphabet* is a finite non-empty set Σ.

Given an alphabet, we can define words (strings of symbols) and operations on them.

DEFINITION 2.2 (Word). Let Σ be an alphabet.

1. A *non-empty word*[1] over Σ is a finite sequence $\sigma_1 \cdots \sigma_n$, where $n \in \mathbb{N}$ and $\sigma_i \in \Sigma$ for $1 \leqslant i \leqslant n$. We define Σ^+ to be the set of all non-empty words over Σ.

2. The *concatenation* of any two non-empty words $w, v \in \Sigma^+$ with $w = w_1 \cdots w_n$ and $v = v_1 \cdots v_m$ such that $n, m \in \mathbb{N}$ and $w_i, v_j \in \Sigma$ for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$ is defined as $w \circ v = wv = w_1 \cdots w_n v_1 \cdots v_m$.

3. A symbol $\varepsilon \notin \Sigma$ called the *empty word* is introduced as the neutral element of concatenation. For any $w \in \Sigma^+$, it holds that $w\varepsilon = \varepsilon w = w$. We put $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

DEFINITION 2.3 (Operations on words). Let Σ be an alphabet and let $w \in \Sigma^*$.

1. The *exponentiation* of w and some $m \in \mathbb{N}_0$, w^m, is defined as follows: $w^0 = \varepsilon$ and $w^{m+1} = w^m \circ w$.

2. The *length* of w, $|w|$, is defined as follows. If $w = \varepsilon$, then $|w| = 0$. If $w = v\sigma$ with $v \in \Sigma^*, \sigma \in \Sigma$, then $|w| = |v| + 1$.

A language is a subset of all possible words over some alphabet.

Languages and closure properties

DEFINITION 2.4 (Language). Let Σ be an alphabet. A *language* over Σ is any set $\mathcal{L} \subseteq \Sigma^*$.

The closure properties of a set of languages describe the behavior of the languages under certain operations.

DEFINITION 2.5 (Closure properties). Let Σ be an alphabet.

1. For two languages $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$,
   a) *union* is defined as $\mathcal{L}_1 \cup \mathcal{L}_2 = \{w \mid w \in \mathcal{L}_1 \text{ or } w \in \mathcal{L}_2\}$,
   b) *intersection* is defined as $\mathcal{L}_1 \cap \mathcal{L}_2 = \{w \mid w \in \mathcal{L}_1 \text{ and } w \in \mathcal{L}_2\}$,
   c) *concatenation* is defined as $\mathcal{L}_1\mathcal{L}_2 = \{wv \mid w \in \mathcal{L}_1 \text{ and } v \in \mathcal{L}_2\}$, and
   d) *complementation* as $\overline{\mathcal{L}_1} = \Sigma^* \setminus \mathcal{L}_1$.

2. Let $\mathfrak{R}$ be a set of languages. $\mathfrak{R}$ is *closed under union* (resp. *intersection, concatenation, complementation*) iff for all $\mathcal{L}_1, \mathcal{L}_2 \in \mathfrak{R}$, it holds that $\mathcal{L}_1 \cup \mathcal{L}_2$ (resp. $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1\mathcal{L}_2$, $\overline{\mathcal{L}_1}$) $\in \mathfrak{R}$.

1 We use the terms *word* and *string* interchangeably.

2.1.2 *Graphs and Trees*

Graphs and trees are basic tools for the description of syntactic information. A directed graph essentially consists of a set of nodes and directed edges between them. The number of incoming, resp. outgoing edges of a node is called its in-degree, resp. out-degree; a sequence of nodes is called a path.

DEFINITION 2.6 (Directed graph). *Graphs*

1. A *directed graph* is a tuple $\mathcal{G} = (V, E)$ with V a finite set of vertices[2] and $E \subseteq V \times V$ a set of arcs.
2. E^+ is the transitive closure of E, E^* its reflexive transitive closure.
3. The *in-degree* function $f_{in} : V \to \mathbb{N}_0$ is such that for all $v \in V$, $f_{in} = |\{u \mid \langle u, v\rangle \in E\}|$.
4. The *out-degree* function $f_{out} : V \to \mathbb{N}_0$ is such that for all $v \in V$, $f_{out}(v) = |\{u \mid \langle v, u\rangle \in E\}|$.
5. Let $r \in V$. If $f_{in}(r) = 0$, then r is a *root node* of the graph.
6. A *path* is a sequence $(v_1, \ldots, v_n)$, where $n \in \mathbb{N}$, $v_i \in V$ for all $1 \leqslant i \leqslant n$, and for all $1 \leqslant j < n$, it holds that $\langle v_j, v_{j+1}\rangle \in E$. v_1 is its *start vertex*, v_n is its *end vertex*. An *elementary path* is a path which contains no repeated vertices. A *cycle* is a path which has identical start and end vertices. An *elementary cycle* is a cycle which, apart from the start and end vertices, contains no repeated vertices.

Cyclicity (there is a path which starts and ends with the same node), connectedness (regarding the edges as undirected, there is a path from every node in the graph to every other node in the graph), and emptiness (the set of nodes is empty) are important properties of a graph.

DEFINITION 2.7 (Graph properties). Let $\mathcal{G} = (V, E)$ be a directed graph.

1. $\mathcal{G}$ is *acyclic* iff it contains no cycles.

2 We use the terms *vertex* and *node* interchangeably.

2. $\mathcal{G}$ is *weakly connected*[3] iff for all $u, v \in V$, $u \neq v$, it holds that $\langle u, v\rangle \in E$ or $\langle v, u\rangle \in E$ or there are $u_1, \ldots u_n \in V$, $n \in \mathbb{N}$, which fulfill the following conditions:
   a) $\langle u, u_1\rangle \in E$ or $\langle u_1, u\rangle \in E$,
   b) $\langle u_n, v\rangle \in E$ or $\langle v, u_n\rangle \in E$, and
   c) $\langle u_i, u_{i+1}\rangle \in E$ or $\langle u_{i+1}, u_i\rangle \in E$ for $1 \leqslant i < n$.
3. $\mathcal{G}$ is *empty* iff $V = \emptyset$.

Labeling nodes and edges allows us to add linguistic information to the graph.

DEFINITION 2.8 (Labeling and labeled graph). Let $\mathcal{G} = (V, E)$ be a directed graph.

1. A *node labeling* over an alphabet L_V is a function $\Lambda_V : V \to L_V$.
2. An *edge labeling* over an alphabet L_E is a function $\Lambda_E : E \to L_E$.
3. $\mathcal{G}$ is *node labeled*, resp. *edge labeled*, resp. *completely labeled*[4] iff it has a node labeling over an alphabet L_V, resp. an edge labeling over an alphabet L_E, resp. both.

A tree is just a special kind of a graph. It must have a single root node, be acyclic and weakly connected. Furthermore, all nodes must have a single incoming edges, except the root, which must have no incoming edges.

Trees

DEFINITION 2.9 (Tree). A *tree* is a triple $\mathcal{D} = (V, E, r)$ where

1. (V, E) is a directed graph which is acyclic and weakly connected,
2. $r \in V$ a distinguished root node, and
3. for all $v \in V \setminus \{r\}$, $f_{in}(v) = 1$ and $f_{in}(r) = 0$.

We want to be able to order the nodes, label nodes and edges, and talk about dominance between nodes.

DEFINITION 2.10 (Properties of trees). Let $\mathcal{D} = (V, E, r)$ be a tree.

3 Since with respect to directed graphs, we are only concerned with weakly connected graphs, in the following, by *connected* we mean *weakly connected*.

4 If it is not of importance if a graph is node labeled, edge labeled or completely labeled, it may just be called *labeled*.

1. $\mathcal{D}$ is *ordered* iff it has a relation $\prec \subseteq V \times V$ which is antisymmetric, irreflexive and transitive.

2. $\mathcal{D}$ is *node labeled*, resp. *edge labeled*, resp. *completely labeled* iff (V, E) is node labeled, resp. edge labeled, resp. completely labeled.

3. For all $v \in V$, if $f_{out}(v) = 0$, then u is a *leaf*, otherwise v is an *internal node*.

4. For all $v', v'' \in V$, we say that v' *directly dominates* v'' iff $\langle v', v'' \rangle \in E$. If v' directly dominates v'', then v'' is called a *daughter node* or *child node* of v' and v' is called the *parent node* of v''. We say that v' *dominates* v'' iff there is a $\langle v', v'' \rangle \in E^*$.

Now we are ready to define the actual linguistic structures we want to have over natural language sentences. They are different kinds of ordered labeled trees. We define them such that the nodes which correspond to the actual words are labeled with integers denoting position indices of the respective words in the sentence. On their basis, we can then define an ordering.

Syntactic structures

DEFINITION 2.11 (Dependency and constituency structure). Let w be a non-empty word over some alphabet Σ. Let $\mathcal{D} = (V, E, r)$ be an ordered tree with a node labeling Λ_V where $\prec$ is such that for all $v_1, v_2 \in V$ with $\Lambda_V(v_1), \Lambda_V(v_2) \in \mathbb{N}_0$ and $\Lambda_V(v_1) < \Lambda_V(v_2)$, it holds that $v_1 \prec v_2$.

1. $\mathcal{D}$ is a *dependency structure* for w if $|V| = |w| + 1$, $L_V = \{0, \ldots, |w|\}$, and Λ_V is such that

   a) $\Lambda_V(r) = 0$, and

   b) for all $1 \leqslant i \leqslant |w|$, there is exactly one $v_i \in V \setminus \{r\}$ with $\Lambda_V(v_i) = i$.

   For all $v \in V$ in a dependency structure, the parent node of v is called its *head* and the daughters of v are called its *dependents*.

2. $\mathcal{D}$ is a *constituency structure* for w if $L_V = \{1, \ldots, |w|\} \cup N$, N being a set of syntactic category labels disjoint from $\{1, \ldots, |w|\}$, and Λ_V is such that

   a) for all $1 \leqslant i \leqslant |w|$, there is exactly one $v_i \in V$ with $f_{out}(v_i) = 0$ and $\Lambda_V(v_i) = i$, and

   b) for all v with $f_{out}(v) > 0$, $\Lambda_V(v) \in N$;

Furthermore, for all $u_1, u_2 \in V$ with $\Lambda_V(u_1), \Lambda_V(u_2) \in N$, $u_1 \prec u_2$ iff $min(\{\Lambda_V(u) \mid u \in V \text{ and } \Lambda_V(u) \in \mathbb{N} \text{ and } \langle u_1, u\rangle \in E^+\}) < min(\{\Lambda_V(u) \mid u \in V \text{ and } \Lambda_V(u) \in \mathbb{N} \text{ and } \langle u_2, u\rangle \in E^+\})$.

Note that in both dependency and constituency structures, we order the leaves with respect to their numeric labeling. In constituency structures, we additionally order the internal nodes by the label of the resp. leftmost leaf they dominate. Our definition excludes ε-labeled leaves on purpose, since they are not needed to describe the treebank data we use (see next section). We now subsume both dependency and constituency structures under a single term.

DEFINITION 2.12 (Syntactic structure). $\mathcal{D} = (V, E, r)$ is a *syntactic structure* iff it is either a dependency structure or a constituency structure. $\mathcal{D}$ may optionally have an edge labeling over an alphabet L_E. Thereby, if $\mathcal{D}$ is a dependency structure, then L_E is a set of dependency labels. Otherwise, L_E is a set of grammatical function labels.

EXAMPLE 2.13 (Syntactic structures). Let us consider again (3), p. 4, repeated here as (5).

(5) Darüber muss nachgedacht werden
Thereof must thought be
"Thereof must be thought."

Figure 8 contains the graphical representation of two syntactic structures for (5), one constituency structure (above) and one dependency structure (below). Note the following points.

- As usual, constituency structures are represented as trees. For dependency structures, the graphical representation method of Kuhlmann (2007) will be adopted. The nodes are arranged in ascending order. Each node is depicted by a circle and connected to its label by a dotted line.
- In the graphical representation of constituency structures, all nodes are arranged such that for two different nodes N_1, N_2 where N_1 dominates N_2, N_2 is situated below N_1. For this reason, all node connections always point downwards and therefore do not need arrows, unlike in dependency structures, where arrows are necessary.

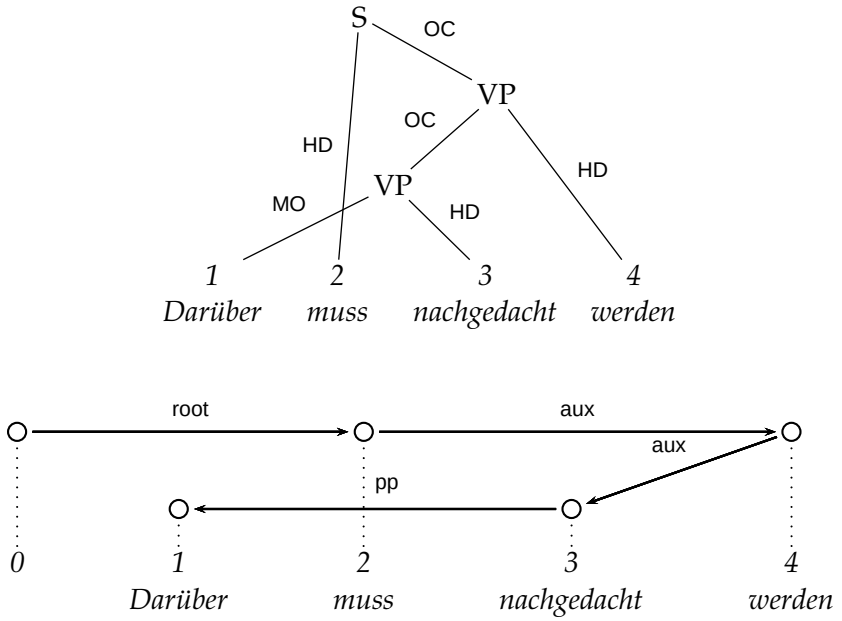


Figure 8: Syntactic structures

- In both syntactic structures, the node labeled i, $1 \leqslant i \leqslant 4$, corresponds to the ith word of (5). In our definition of dependency structures, we assume an additional root node which is mapped to 0, in order to ensure that the resulting graph is connected. For easier accessibility, in further graphical representations of syntactic structures, the words of the sentence are used for labeling (instead of their indices).
- In the case of constituency structures, non-terminal nodes are labeled with syntactic category labels (VP, S), and edges with grammatical function labels (OC, HD, MO). In the case of dependency structures, the edges carry dependency type labels (root, aux, pp). In further graphical representations, the edge labeling may be omitted, if of no importance.

The yield of a node is the set of labels of all integer-labeled nodes which it dominates. In other words, the yield of a node represents the part of the sentence which the node covers.

Yield and yield blocks

DEFINITION 2.14 (Yield). Let $\mathcal{D} = (V, E, r)$ be a syntactic structure. The *yield* function $\pi : V \to \mathfrak{P}_{fin}(\mathbb{N})$ is such that for all $v \in V$, $\pi(v) = \{i \in \mathbb{N} \mid \text{ there is a } v' \in V \text{ with } \langle v, v' \rangle \in E^* \text{ and } \Lambda_V(v') = i\}$.

We call a continuous sequence of integers in the yield of a node a *yield block*. The set of all yield blocks of a node is its yield block set.

DEFINITION 2.15 (Yield block set). Let $\mathcal{D} = (V, E, r)$ be a syntactic structure. For all $v \in V$, the *yield block set* Ω_v of v is a partition of $\pi(v)$ given by the equivalence relation $\mathfrak{O}$ on $\pi(v)$ which is such that for all $i, j \in \pi(v)$, $i \sim_{\mathfrak{O}} j$ iff

1. $i < j$ and $i + 1 \in \pi(v)$ and $i + 1 \sim_{\mathfrak{O}} j$, or
2. $i > j$ and $i - 1 \in \pi(v)$ and $i - 1 \sim_{\mathfrak{O}} j$, or
3. $i = j$.

Any $\omega \in \Omega_v$ is called a *yield block* of v.[5]

The subscript on the yield block set may be omitted if clear from the context. The yield block set is ordered by the respective smallest integer of the blocks it contains.

DEFINITION 2.16 (Yield block set ordering). Let $\mathcal{D} = (V, E, r)$ be a syntactic structure. For the block set Ω_v of any $v \in V$, the following holds.

1. Ω_v is equipped with a strict total order $\prec$ called *block ordering* which is such that for all $\omega_1, \omega_2 \in \Omega_v$, $\omega_1 \prec \omega_2$ iff $min(\omega_1) < min(\omega_2)$.
2. Any $\omega' \in \Omega_v$ is the i*th block of* v, notated $v^{(i)}$, iff $|\{\omega \in \Omega_v \mid \omega \prec \omega'\}| = i - 1$.

Block degree

The block degree of a node is the cardinality of its block set.

DEFINITION 2.17 (Block degree). Let $\mathcal{D} = (V, E, r)$ be a syntactic structure. For all $v \in V$, the *block degree* function $dim : V \to \mathbb{N}$ yields the cardinality of the block set of v.

EXAMPLE 2.18 (Yield, yield blocks and block degree). Consider again the two syntactic structures in figure 8.

- The node labeled S in the constituency structure and the node labeled 2 in the dependency structure have the yield $\{1, 2, 3, 4\}$, respectively. The corresponding yield block set is $\{\{1, 2, 3, 4\}\}$, and the block degree $|\{\{1, 2, 3, 4\}\}| = 1$.

5 We use the terms *yield block* and *yield component* interchangeably and omit *yield* when clear from the context.

- The upper node labeled VP in the constituency structure and the node labeled 4 in the dependency structure have the same yield $\{1,3,4\}$. The corresponding yield block set is $\{\{1\},\{3,4\}\}$ (where $\{1\}$ is the first and $\{3,4\}$ the second yield block), and the block degree is $|\{\{1\},\{3,4\}\}| = 2$.

2.1.3 *Treebanks*

A collection of texts is called a *corpus*. A corpus in which each sentence is annotated with a syntactic structure is called a *treebank*. The annotation in a treebank is typically created by human annotators. It is often split in modules, i. e., apart from a syntactic annotation level, a treebank can contain, e. g., Part-of-Speech (POS) annotation, morphological annotation, annotation of semantic frames, etc. The rules which human annotators follow during annotation are generally written down in a *stylebook*. The stylebook determines the *annotation schema*, i. e., it determines what kind of information is encoded and how the encoding is accomplished. Here, we focus on the syntactic annotation.

We now define treebanks. Recall the definitions of syntactic structures (defs. 2.11 and 2.12).

DEFINITION 2.19 (Treebank). *Treebanks*

1. A *lexicon* is an alphabet of atomic natural language tokens $\mathcal{W}$. A *POS tag set* is an alphabet of POS tags $\mathcal{P}$.

2. Let $\mathcal{W}$ be a lexicon and $\mathcal{P}$ be a POS tag set. A *POS-tagged token* over $\mathcal{W}$ and $\mathcal{P}$ is a tuple $\langle w, p\rangle \in \mathcal{W} \times \mathcal{P}$. For $\langle w, p\rangle$ we may write w/p, or just w, if the POS tag is of no importance. A *POS tagged sentence* over $\mathcal{W}$ and $\mathcal{P}$ is a non-empty word $s \in (\mathcal{W} \times \mathcal{P})^+$.

3. A *treebank* is a set Υ of tuples (i, υ_i, s_i), $1 \leqslant i \leqslant |\Upsilon|$, where i is an index which is unique to each tree, and υ_i is a syntactic structure for some POS tagged sentence[6] s_i. υ_i is also called the *(syntactic) annotation* of s_i.

4. Υ is called a *constituency treebank*, resp. a *dependency treebank*, iff for all $(i, \upsilon_i, s_i) \in \Upsilon$, $1 \leqslant i \leqslant |\Upsilon|$, it holds that υ_i is a constituency structure, resp. a dependency structure.

6 The nature of the treebanks we are investigating (cf. sections 4.2.1 and 4.3.1) allows us to assume that all words carry POS annotation.

Instead of *token*, we may also say *word*, if it is clear from the context that we not referring to a word in the sense of definition 2.2, but to a natural language token.

2.2 GRAMMATICAL DESCRIPTION

In order to formulate a linguistic theory in a mathematically explicit way, an appropriate mathematical framework is needed. Such a framework needs to provide a possibility for licensing, resp. ruling out sentences, and for specifying the appropriate structure of a licensed sentence. The two major classes of such frameworks are *Model-Theoretic Syntax (MTS)* and *Generative Enumerative Syntax (GES)* (Pullum and Scholz, 2001). The contributions of this thesis focus on the latter (see also section 1.2). For details on MTS, the reader is referred to Pullum and Scholz (2001) and Pullum (2007).

Frameworks belonging to GES are finite devices called *grammar formalism* or simply *formalism*. They typically allow for the specification of an axiom and substitution rules. Starting from the axiom, languages can be generated via a series of (recursive) rule applications. Pullum and Scholz (2001) present a list of major frameworks, completed here by several others:[7]

- Phrase Structure Grammar, including Context-Free Grammar, introduced by Noam Chomsky (Chomsky, 1956),
- Transformational Grammar, Chomsky's work on linguistics in its various versions (Chomsky, 1957, 1981, 1995),
- Tree-Adjoining Grammar, introduced by Aravind Joshi (Joshi et al., 1975),
- Multi-Component Tree-Adjoining Grammar, introduced by Joshi (1987),
- Different flavors of Categorial Grammar, such as Combinatory Categorial Grammar, introduced by Mark Steedman (Steedman, 2000),
- Generalized Phrase Structure Grammar, the work of Gerald Gazdar and colleagues (Gazdar et al., 1985),

7 This list is, of course, still far from exhaustive.

- Linear Context-Free Rewriting System (Vijay-Shanker et al., 1987) and the independently developed equivalent formalisms Multiple Context-Free Grammar (Seki et al., 1991) and Simple Range Concatenation Grammar (Boullier, 1998),
- Minimalist Grammar (Stabler, 1997), the formalization of Chomsky's minimalist program (Chomsky, 1995), and
- Range Concatenation Grammar, introduced by Pierre Boullier (Boullier, 1998).

Only some of these frameworks will be used in this thesis.

2.2.1 *Context-Free Grammar*

For the presentation of the definition of Context-Free Grammar and related definitions, we follow Hopcroft and Ullman (1979).

Context-Free Grammar

DEFINITION 2.20 (Context-Free Grammar). A *Context-Free Grammar (CFG)* is a tuple $G = (N, T, P, S)$, where

1. N and T are disjoint alphabets of non-terminal symbols and terminal symbols,
2. $S \in N$ is a distinguished *start symbol*, and
3. $P \subseteq N \times (N \cup T)^*$ is a finite set of *productions*.[8] A production $\langle A, \beta \rangle \in P$ is written as $A \to \beta$.

The derivational process on the basis of which we can derive a set of words using a CFG is driven by the relation *derives*.

Derivation and language

DEFINITION 2.21 (Derivation (CFG)). Let $G = (N, T, P, S)$ be a CFG.

1. $\Longrightarrow_G \subseteq (N \cup T)^+ \times (N \cup T)^*$ is a relation called *derives* which is defined as follows. $\gamma \Longrightarrow_G \gamma'$ iff there are $A \to \beta \in P$ and $\alpha, \alpha' \in (N \cup T)^*$ with $\gamma = \alpha A \alpha'$ and $\gamma' = \alpha \beta \alpha'$. We may write $\Longrightarrow_G^{A \to \beta}$ to make the production explicit. If G is not relevant or implicitly understood, we may omit the corresponding subscript.
2. $\overset{*}{\Longrightarrow}_G$ is the reflexive transitive closure of $\Longrightarrow_G$.

8 Productions are also called *rules*.

3. Let $\alpha_1, \ldots, \alpha_m \in (N \cup T)^*$, $m \in \mathbb{N}$. A sequence

$$\alpha_1 \Longrightarrow_G \cdots \Longrightarrow_G \alpha_m$$

is a *derivation* of length m, also written $\alpha_1 \overset{m}{\Longrightarrow}_G \alpha_m$. α_m is called *derivable* from α_1. Each $\alpha_i \Longrightarrow_G \alpha_j$, $1 \leqslant i, j \leqslant m$, $j = i + 1$, is a *derivation step*.

The *language* of a CFG consists of the terminal words that can be derived from its start symbol.

DEFINITION 2.22 (Language (CFG)). Let $G = (N, T, P, S)$ be a CFG. The *language* of G is $\mathcal{L}(G) = \{w \mid w \in T^* \text{ and } S \overset{*}{\Longrightarrow}_G w\}$.

The derivations of a grammar formalism can be represented as trees. Such trees are called *derivation trees*.

Derivation trees

DEFINITION 2.23 (Derivation tree (CFG)). Let $G = (N, T, P, S)$ be CFG. A *derivation tree*[9] for G is an labeled ordered tree $\mathcal{D} = (V, E, r)$ which is as follows.

1. $\prec$ is such that for all $u_1, u_2 \in V$ with $\langle u_1, u_2 \rangle \notin E^*$ and $\langle u_2, u_1 \rangle \notin E^*$,
   a) either $u_1 \prec u_2$ or $u_2 \prec u_1$,
   b) for all $u_1', u_2' \in V$, if
      i. $\langle u_1, u_1' \rangle \in E$ and $u_1 \prec u_2'$ or
      ii. $\langle u_2, u_2' \rangle \in E$ and $u_1' \prec u_2$,

      then it holds that $u_1' \prec u_2'$;
   c) $\prec$ contains nothing else.
2. It has a node labeling Λ over $N \cup T \cup \{\varepsilon\}$ such that
   a) for all $v \in V$, if $f_{out}(v) = 0$, then $\Lambda(v) \in T \cup \{\varepsilon\}$, otherwise $\Lambda(v) \in N$, and
   b) for all $v_0, v_1, \ldots, v_n \in V$, $n \in \mathbb{N}$, $1 \leqslant i \leqslant n$, $1 \leqslant j < n$, where
      i. $\langle v_0, v_i \rangle \in E$, and there is no $v' \in V \setminus \{v_1, \ldots, v_n\}$ with $\langle v_0, v' \rangle \in E$,
      ii. $v_j \prec v_{j+1}$, and
      iii. $\Lambda(v_0) = A$ and $\Lambda(v_i) = \beta_i$,

9 Derivation trees are also called *parse trees*.

there exists a $A \to \beta_1 \cdots \beta_n \in P$.

$\mathcal{D}$ is also called a $\Lambda(r)$-*tree*. If $\Lambda(r) \neq S$, then $\mathcal{D}$ is called a *partial derivation tree*.

Derivation trees are connected to derivations through their frontier, the word which results from reading all leaf nodes from left to right.

DEFINITION 2.24 (Frontier). Let $G = (N, T, P, S)$ be a CFG and let $\mathcal{D} = (V, E, r)$ be a derivation tree for G. Let $V_l = \{v \in V \mid f_{out}(v) = 0\}$. The *frontier* of $\mathcal{D}$ is the word $\Lambda(u_1) \cdots \Lambda(u_{|V_l|})$ where for all $1 \leqslant i \leqslant |V_l|$, $u_i \in V_l$ and $|\{u \in V_l \mid u \prec u_i\}| = i - 1$.

PROPOSITION 2.25. Let $G = (N, T, P, S)$ be a CFG, and let $w \in T^*$. $S \overset{*}{\Longrightarrow}_G w$ iff there is a derivation tree for G with frontier w (Hopcroft and Ullman, 1979).

DEFINITION 2.26 (Tree language (CFG)). Let $G = (N, T, P, S)$ be a CFG. The *tree language* of G is $\mathcal{T}(G) = \{\gamma \mid \gamma$ is a derivation tree (V, E, r) for G with $\Lambda(r) = S\}$.

To distinguish the tree language of a grammar G from its language, the latter is also referred to as its *string language*.

Normal forms which impose restrictions on the grammar can be useful for parsing. Among others, we define ε-free CFG and the Chomsky Normal Form. *Normal Forms*

DEFINITION 2.27 (ε-free CFG). Let $G = (N, T, P, S)$ be a CFG. G is *ε-free* if either

1. $\varepsilon \in \mathcal{L}(G)$, there is a production $S \to \varepsilon \in P$, S occurs in no other right-hand side (RHS) of a production in P and there is no other production of the form $A \to \varepsilon \in P$ for some $A \in N$; or
2. $\varepsilon \notin \mathcal{L}(G)$ and there is no production of the form $A \to \varepsilon \in P$ for some $A \in N$.

DEFINITION 2.28 (Useful symbol). Let $G = (N, T, P, S)$ be a CFG. A nonterminal $A \in N$ is *useful* if for $\alpha, \gamma \in (N \cup T)^*$ and $w \in T^*$, there is a derivation $S \overset{*}{\Longrightarrow}_G \alpha A \gamma \overset{*}{\Longrightarrow}_G w$. Otherwise, A is *useless*.

PROPOSITION 2.29. If $L = \mathcal{L}(G)$ for some CFG $G = (N, T, P, S)$, then $L \setminus \{\varepsilon\}$ is the language $\mathcal{L}(G')$ of a CFG G' without useless symbols and without productions of the form $A \to \varepsilon$ (Hopcroft and Ullman, 1979).

DEFINITION 2.30 (Chomsky Normal Form). Let $G = (N, T, P, S)$ be a CFG with $\varepsilon \notin \mathcal{L}(G)$. G is in *Chomsky Normal Form (CNF)* iff all $p \in P$

are either of the form $A \to a$ or of the form $A \to BC$, where $a \in T$ and $A, B, C \in N$.

PROPOSITION 2.31. For every ε-free CFG G, there is another CFG G' in CNF with $\mathcal{L}(G') = \mathcal{L}(G)$ (Hopcroft and Ullman, 1979).

Context-Free Grammar has been introduced within a hierarchy of formalisms with different expressivity, the *Chomsky Hierarchy* (Chomsky, 1956). However, CFG is the only formalism of the hierarchy which will be used in this thesis.

In order to be able to compare formalisms, we introduce weak and strong equivalence between formalisms.

Equivalence of formalisms

DEFINITION 2.32 (Equivalence of formalisms). Let $\mathfrak{F}_1$ and $\mathfrak{F}_2$ be two formalisms.

1. $\mathfrak{F}_1$ and $\mathfrak{F}_2$ are *weakly equivalent* if
   a) the concept of a language is defined for both, and
   b) for every grammar instance $G_{\mathfrak{F}_1}$ of $\mathfrak{F}_1$ generating the language $\mathcal{L}$, there is a grammar instance $G_{\mathfrak{F}_2}$ of $\mathfrak{F}_2$ which also generates $\mathcal{L}$ and vice versa.
2. $\mathfrak{F}_1$ and $\mathfrak{F}_2$ are *strongly equivalent* if
   a) the concept of a tree language is defined for both, and
   b) for every grammar instance $G_{\mathfrak{F}_1}$ of $\mathfrak{F}_1$ generating the tree language $\mathcal{T}$, there is a grammar instance $G_{\mathfrak{F}_2}$ of $\mathfrak{F}_2$ which also generates $\mathcal{T}$ and vice versa.

EXAMPLE 2.33 (Context-Free Grammar). Let $G = (\{S, A\}, \{a, b\}, P, S)$ be a CFG with $P = \{S \to AB, A \to aAb, A \to \varepsilon, B \to cBd, B \to \varepsilon\}$. It is easy to see that $\mathcal{L}(G) = \{a^n b^n c^m d^m \mid n, m \geqslant 0\}$. Figure 9 shows the graphical representation of a derivation tree for *aabb* $\in \mathcal{L}(G)$.

The example shows a characteristic property of Context-Free Languages (CFLs), namely *center embedding*. Center embedding leads to words with a nested structure. Cross-serial dependencies (see p. 3) can not be described. Consider again (2) and (3), repeated here as (6) and (7).

(6) ... mer em Hans_i es Huus_j hälfed_i aastriiche_j
... we Hans-DAT the house-ACC help paint
... we help Hans paint the house

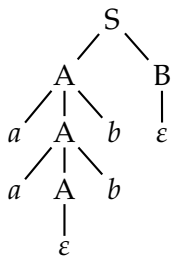


Figure 9: CFG derivation tree

(7) ...mer d'chind$_i$ em Hans$_j$ es Huus$_k$ lönd$_i$ hälfe$_j$ aastriiche$_k$
...we the kids Hans the house let help paint
"...we let the kids help Hans paint the house"

Introducing one symbol for each verb/complement combination, (6) can be abstracted to the word *abab*, (7) to *abcabc*. In fact, the abstraction can be generalized to the *copy language* $\mathcal{L}_{\text{copy}} = \{ww \mid w \in \{a, b\}^*\}$. However, this language is no CFL. This can be shown using the closure properties of CFLs, generalizing the proof idea of Shieber (1985) who shows non-context-freeness for a special case of this language.

2.2.2 *Mild Context-Sensitivity*

Mild Context-Sensitivity

Joshi (1985) introduces *Mild Context-Sensitivity (MCS)* as an attempt to characterize in a formal way the amount of context-sensitivity, i. e., the amount of expressivity beyond CFG which is necessary to model natural language. A set of languages is mildly context-sensitive if it obeys the following necessary conditions.[10]

1. It properly contains all CFLs.
2. It allows for a limited amount of cross-serial dependencies, i. e., it contains all string languages $\{w^n \mid w \in \Sigma^*\}$ for some $n \geqslant 2$.
3. All languages in the set can be recognized in polynomial time (see section 2.3.2).

10 See Kracht (2003), pp. 369, and Kallmeyer (2010a) for a critical discussion of the conditions from a formal point of view.

4. All languages in the set have the *constant growth property*.[11]

If the set of languages which can be generated by a formalism is mildly context-sensitive, then we speak of a *mildly context-sensitive formalism* or *MCS formalism*. Now, two such formalisms will be introduced, namely Tree-Adjoining Grammar and Linear Indexed Grammar. Formal definitions will not be presented, since none of the contributions of this thesis requires them.

Tree-Adjoining Grammar

The first version of *Tree-Adjoining Grammar (TAG)* has been introduced under the name of Tree Adjunct Grammar in Joshi et al. (1975). A comprehensive introduction can be found in Joshi and Schabes (1997). For a complete formal definition of TAG, consult, for instance, Kallmeyer (1999) or Kallmeyer (2010b). TAG is not a symbol-rewriting formalism such as CFG, but a *tree-rewriting formalism*. A TAG consists of a set of labeled ordered trees called *elementary trees*. For linguistic modeling, *Lexicalized Tree-Adjoining Grammar (LTAG)* is used. In LTAG, the trees are *lexicalized*, i. e., at least one of its leaves is labeled with a terminal. Generally, the linguistic principle of *elementary tree minimality* is respected: a tree must encode the subcategorization requirements of one lexical item (Frank, 2002). Elementary trees come in two variants. *Auxiliary trees* have one leaf node called *foot node* labeled with the same label as the root node and marked with a star. *Initial trees* are non-auxiliary elementary trees.

A derivation is built by combining elementary trees into new trees, using two different operations. The *substitution* operation allows to replace a non-terminal leaf of an elementary tree (*substitution node*) with another elementary tree the root of which carries the same label. The *adjunction* operation allows to replace the internal node of a tree (*adjunction node*) with an auxiliary tree the root and foot of which carry the same label; i. e., the part below the node to which the auxiliary tree is to be adjoined is excised, the node is replaced with the auxiliary tree, and the excised part of the original tree is appended to the foot node. Generally, by definition, only one adjunction happens at a time and one can only adjoin at a node once (Joshi and Schabes, 1997). In order to generate the copy language, TAG must be extended with *adjunction constraints*. These are given by two functions. The first one tells for

11 The lengths of the words generated by grammars of a formalism which has the constant growth property must grow linearly, i. e., not, e. g., exponentially as in the language $\{a^{2^n} \mid n \geqslant 1\}$. The constant growth property of a language can be shown by showing its *semilinearity* (Parikh, 1966).

each node if adjunction is obligatory (node is marked with *OA* for *obligatory adjunction*). The second one yields for each node the set of trees that can be adjoined. If this set is empty, the node is marked *NA* for *no adjunction*.

A tree which is the result of a tree combination is called a *derived tree*. In CFG, the derived tree, i. e., the result of the application of productions, is isomorphic to the derivation tree. This is not the case for TAG, since one cannot reconstruct from a derived tree alone how it has been built. A *TAG derivation tree* does encode this information. In such a tree, every node is labeled with the name of a tree an instance of which has been used in a derivation. An edge between two nodes exists if for the corresponding trees, it holds that one tree has been adjoined or substituted into the other. The edge is labeled with the Gorn address[12] of the substitution, resp. adjunction node.

Example 2.34 (Tree-Adjoining Grammar). Figure 10 shows a TAG for the copy language. Furthermore, it shows the derivation process, a derivation tree and a derived tree for the word *abab*. It is easy to see that the reason for the additional expressivity of TAG is the adjunction operation, which can be controlled through adjunction constraints.

Figure 2, p. 5, shows a linguistic example with the derivation process and the derived tree. It features both an adjunction and substitutions.

Feature Structure TAG

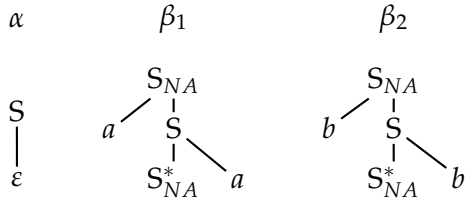
A popular variant of TAG is *Feature Structure Based Tree-Adjoining Grammar (FTAG)* (Vijay-Shanker and Joshi, 1988). A *feature structure*[13] is a data structure which consists of a set of attribute-value pairs. Attributes are names; values can be of different types (e. g., boolean, string, integer, or other feature structures). *Structure sharing* allows to express, resp. force value identities. In FTAG, every node in an elementary tree carries two feature structures (generally represented as *Attribute-Value Matrices (AVMs)*, see figure 11 for an example), a *top* and a *bottom* feature structure.[14] Given a node A, the idea is that the top structure gives information about the part of the tree above A and the bottom structure about the part below A. *Feature structure unifi-*

12 In this convention (Gorn, 1967), the root address is ε and the jth child of a node with address p has address $p \cdot j$. The node labeled ε in the derived tree in figure 10, e. g., has the address 22111.

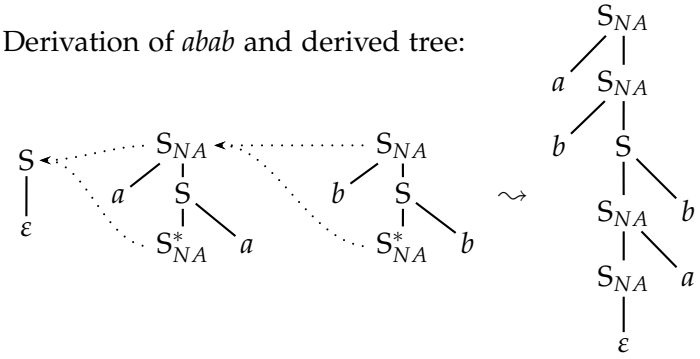
13 The TEI Consortium offers a good introduction to feature structures in their guidelines for feature structure representation (TEI Consortium, 2010).

14 Note that structure sharing is expressed via variables, written as boxed numbers.

TAG generating the copy language:



Derivation of *abab* and derived tree:



Derivation tree:

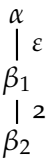


Figure 10: A TAG for the copy language

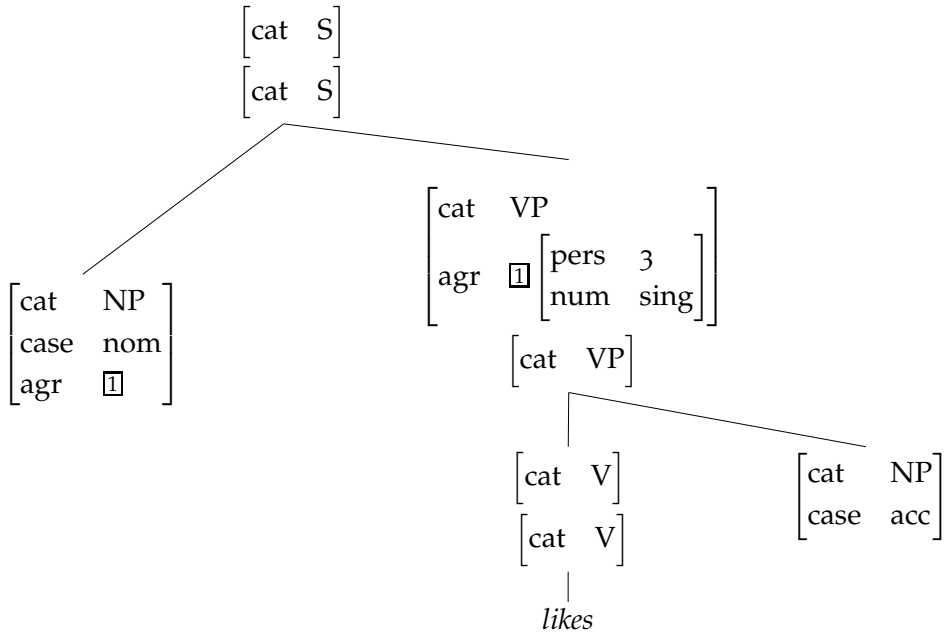


Figure 11: An elementary tree in FTAG

cation plays a central role. Intuitively, the unification of two feature structures is built by creating a new feature structure and adding all attribute-value pairs from both feature structures to it. The result of unification is undefined if there is a value clash for two features. When adjoining a tree β at A, the top structure of A is unified with the top structure of the root node of β and the bottom structure of A is unified with the bottom structure of the foot node of β. To complete a derivation, all top and bottom structures are unified. A derivation is only possible if all feature structure unifications are defined.

Linguistic principles, such as agreement, can be modeled easily with FTAG. Therefore, grammar implementations make extensive use of feature structures (XTAG Research Group, 2001; Kallmeyer et al., 2008a). As an example, figure 11 shows an elementary tree for the word *likes* with feature structures. The feature structures can take over the role of the adjunction constraints. Note how structure sharing is used to restrict the set of trees which can be substituted at the subject position.

Multi-Component Tree-Adjoining Grammar (MCTAG) is a variant of TAG. Instead of single elementary trees, an MCTAG contains sets of

Multi-Component TAG

trees (therefore *multi-component*). In a single adjunction step, all of the trees of a set must be used at once. The key idea is that by splitting the requirements, resp. the contributions of a lexical item on various trees, more flexibility is gained for linguistic modeling. In *non-local* MCTAG, no restriction is imposed on the adjunction operation (apart from the restriction that all trees must be adjoined at the same time). In *tree-local* and *set-local* MCTAG (Joshi, 1985), the trees of a tree set may be adjoined to a single tree, resp. to trees from a single tree set. In *Vector Tree-Adjoining Grammar (VTAG)* with dominance links (Rambow, 1994), the simultaneous adjunction condition is relaxed. Locality is achieved by imposing dominance links on the trees in the multi-component sets which must be respected in the final derivation. *Multi-Component Tree-Adjoining Grammar with Tree Tuples (TT-MCTAG)* (Lichte, 2007) takes yet another approach. Like in VTAG, the condition of simultaneity of adjunction is relaxed; still, all the trees of a multi-component set must be used once a single tree of it has been used. Furthermore, the number of sets which can be "open" at the same time is limited. For a formal definition of MCTAG and several of its variants, consult Kallmeyer (2009). Note that tree-local MCTAG is strongly equivalent to TAG and set-local MCTAG is weakly equivalent to Linear Context-Free Rewriting System (LCFRS), which will be introduced in the following. As far as parsing complexity (see section 2.3.2) is concerned, parsing non-local MCTAG without restrictions and parsing non-local MCTAG with dominance link is NP-complete, even in the lexicalized case (Champollion, 2011). As for TT-MCTAG, only the universal recognition problem is NP-complete (Søgaard et al., 2007; Kallmeyer and Satta, 2009).

Linear Indexed Grammar

Linear Indexed Grammar (LIG) (Gazdar, 1988) is an extension of CFG which is weakly equivalent to TAG (Vijayshanker, 1987). It is defined as a tuple $G = (N, T, I, P, S)$ where N, T, S are as for CFG and I is a finite set of indices. The productions in P look like the productions of CFG except that the non-terminals carry stacks of indices from I. Stacks are written as $[\ldots]$. There are three different types of productions, each with different operations on the stacks. In all productions, the stack is first copied from the non-terminal on the left-hand side (LHS) to a single non-terminal of the RHS. Then, either nothing else happens, or a stack symbol is pushed on the stack, or a stack symbol is deleted from the stack. The key idea is to use the stacks

for counting, i. e., as a kind of memory, and to overcome this way the limitations of center embedding.

Example 2.35 (LIG). As an example, consider the LIG

$$G = (\{S, T\}, \{a, b\}, \{\#, a, b\}, P, S)$$

where P contains the following productions.

$$\begin{array}{rcl@{\qquad}rcl}
S_0 & \to & S[\#] & & & \\
S[\ldots] & \to & aS_a[\ldots] & S_a[\ldots] & \to & S[a\ldots] \\
S[\ldots] & \to & bS_b[\ldots] & S_b[\ldots] & \to & S[b\ldots] \\
S & \to & T & & & \\
T[a\ldots] & \to & T[\ldots]a & T[b\ldots] & \to & T[\ldots]b \\
T[\#] & \to & \varepsilon & & &
\end{array}$$

G generates the copy language. The productions interact as follows. The S non-terminals are used to generate the first half of a word of the copy language, in other words, the word to be copied. While generating it, the symbols used are "recorded" on the stack. At some point, $S \to T$ is applied and the second part of the word, in other words the copy, is generated on the basis of the stack's contents.

LIG is a restricted form of Indexed Grammar (IG) (Aho, 1968). It is *linear* in the sense that the stack is copied from the LHS non-terminal to a single RHS non-terminal.

(Linear) Indexed Grammar is not as nice to use as a linguistic description device as TAG. Expressing linguistic abstractions in a tree rewriting system is more immediate than expressing them in a system of symbol rewriting (cf. p. 6). However, due to the proximity of LIG to CFG, the parsing problem for LIG is conceptually easier to solve than for TAG. Therefore, as already mentioned, LIG is used as a pivot formalism for parsing (Boullier, 1996).

2.2.3 *Generalized Rewriting*

Generalized rewriting

Several formalisms share the property that their derivation processes can be expressed by a CFG. Capturing this intuition is the idea behind Linear Context-Free Rewriting System (Vijay-Shanker et al., 1987). The backbone of the definition of Linear Context-Free Rewriting System is Generalized Context-Free Grammar (Pollard, 1984).[15]

15 Our definition of LCFRS follows Weir's (1988) definition.

DEFINITION 2.36 (Generalized Context-Free Grammar). A *Generalized Context-Free Grammar (GCFG)* is a tuple $G = (N, S, F, P)$ where

1. N is an alphabet of non-terminals,
2. F is an alphabet of function symbols disjoint from N with an arity function $dim : F \to \mathbb{N}_0$
3. $S \in N$ is a distinguished start symbol, and
4. P is a finite set of productions $A \to f(A_1, \ldots, A_n)$ where $n \in \mathbb{N}_0$,[16] $A, A_1, \ldots A_n \in N$, $f \in F$ and $dim(f) = n$.

DEFINITION 2.37 (Derivation (GCFG)). Let $G = (N, S, F, P)$ be a GCFG. $\Longrightarrow_G$ is a relation called *derives* which is such that

1. $A \Longrightarrow_G f()$ if $A \to f() \in P$, and
2. $A \overset{*}{\Longrightarrow}_G f(t_1, \ldots, t_n)$ if $A \to f(A_1, \ldots, A_n) \in P$ and $A_i \overset{*}{\Longrightarrow}_G t_i$ for $1 \leqslant i \leqslant n$.

A GCFG G derives a set of terms $\mathfrak{T}(G)$ such that for all $t \in \mathfrak{T}(G)$, $S \overset{*}{\Longrightarrow}_G t$. The idea is to interpret these terms as the derivation trees of various grammar formalisms. To do that, we need to give an interpretation for each $f \in F$, such that each $f \in F$ is mapped onto the composition operation of the formalism. The interpretation of some term $t \in \mathfrak{T}(G)$ in some formalism $\mathfrak{F}$ is written as $[\![t]\!]_{\mathfrak{F}}$ (Weir, 1988, pp. 92).

As for the interpretation for CFG, the co-domain of f are strings, since Context-Free Grammar is a string-rewriting formalism. As a concrete example, consider a CFG with the productions $S \to aSb$ and $S \to C$. A corresponding GCFG G' would contain the productions $S \to f(X_a, S, X_b)$, $S \to f(C)$, $X_a \to f_a()$, $X_b \to f_b()$. $[\![f_w()]\!]_{CFG} = w$ for all f_w with $dim(f_w) = 0$ and some string w, i. e., in our example, we define $[\![f_a()]\!] = a$ and $[\![f_b()]\!] = b$. Futhermore, and finally, we define

$$[\![f(t_1, \ldots, t_n)]\!]_{CFG} = [\![t_1]\!]_{CFG} \circ \cdots \circ [\![t_n]\!]_{CFG}.$$

However, the interpretation function is not restricted to a co-domain of strings. Weir (1988) gives more examples. For TAG, the co-domain of the interpretation function are trees. For Head Grammars (HGs)

16 As usual, if $n = 0$, then we have a production of the form $A \to f()$.

(Pollard, 1984), the co-domain are pairs of strings. For MCTAG, the co-domain are finite sequences of trees.

One can capture the common properties of different interpretation functions by defining a *yield function* on the interpretations of terms to capture their contribution of terminals to a final derivation (Weir, 1988, p. 94). We assume a function $\phi_{\mathfrak{F}}$ which returns the yield $\langle w_1, \ldots, w_k \rangle$ of intermediate structures $[\![t]\!]_{\mathfrak{F}}$ produced by a formalism $\mathfrak{F}$, where $\langle w_1, \ldots, w_k \rangle$, $k \in \mathbb{N}_0$, is a tuple of terminal strings; and we assume a function $ar_{\mathfrak{F}}$ which for an intermediate structure $[\![t]\!]_{\mathfrak{F}}$ returns the arity of the tuple returned by $\phi_{\mathfrak{F}}([\![t]\!]_{\mathfrak{F}})$. Note that ϕ_{CFG} is the identity.

Linear Context-Free Rewriting Systems

Definition 2.38 (Linear Context-Free Rewriting System). A *Linear Context-Free Rewriting System (LCFRS)* is a GCFG $G = (N, S, F, P)$ for which the following holds. A unique yield function $\bar{f}_{\mathfrak{F}}$ is associated with each $f \in F$ such that if

$$\phi_{\mathfrak{F}}([\![t_i]\!]_{\mathfrak{F}}) = \langle w_{i,1}, \ldots, w_{i,ar_{\mathfrak{F}}([\![t_i]\!]_{\mathfrak{F}})} \rangle$$

for all $1 \leqslant i \leqslant n$ and

$$\phi([\![f(t_1, \ldots, t_n)]\!]_{\mathfrak{F}}) = \langle w_1, \ldots, w_k \rangle$$

for $k \in \mathbb{N}_0$ then

$$\bar{f}_{\mathfrak{F}}(\langle w_{1,1}, \ldots, w_{1,ar_{\mathfrak{F}}([\![t_1]\!]_{\mathfrak{F}})} \rangle, \ldots, \langle w_{n,1}, \ldots, x_{n,ar_{\mathfrak{F}}([\![t_n]\!]_{\mathfrak{F}})} \rangle) = \langle w_1, \ldots, w_k \rangle$$

and every $\bar{f}_{\mathfrak{F}}$ can be defined by an equation of the form

$$\bar{f}_{\mathfrak{F}}(\langle x_{1,1}, \ldots, x_{1,m_1} \rangle, \ldots, \langle x_{n,1}, \ldots, x_{n,m_n} \rangle) = \langle \alpha_1, \ldots, \alpha_m \rangle$$

where $n \in \mathbb{N}_0$, $m, m_1, \ldots, m_n \in \mathbb{N}$, and for all $1 \leqslant p \leqslant n$, $1 \leqslant q \leqslant m_n$, $x_{p,q}$ is a variable, and each α_i, $1 \leqslant i \leqslant m$, is a finite string over variables occurring on the LHS of the equation and terminal symbols. Thereby, the m is called the *fan-out* of the function and n is called its *rank*. The rank and fan-out of G are the maximal fan-out of its equations. The equations must be *non-erasing* (all variables which appear on the LHS also appear on the RHS and vice versa) and *linear* (all variables appear only once on the LHS and only once on the RHS).

The string languages generated by formalisms which can be formulated as LCFRSs can be obtained by linking the equations from definition 2.38 to the productions of the underlying GCFG (Weir, 1988, p. 95).

Derivation and language

DEFINITION 2.39 (Derivation (LCFRS)). Let $G = (N, S, F, P)$ be a Linear Context-Free Rewriting System. $\overset{*}{\Longrightarrow}_G$ is a relation called *derives* which is such that for all $A \in N$, $A \overset{*}{\Longrightarrow}_G \langle w_1, \ldots, w_m \rangle$, $m \in \mathbb{N}_0$, if one the following conditions hold.

1. there is a $A \to f() \in P$ such that $\bar{f}_{\mathfrak{F}}() = \langle w_1, \ldots, w_m \rangle$.
2. there is a $A \to f(A_1, \ldots, A_n) \in P$, $n \in \mathbb{N}$, with yield function

   $$\bar{f}_{\mathfrak{F}}(\langle x_{1,1}, \ldots, x_{1,m_1} \rangle, \ldots, \langle x_{n,1}, \ldots, x_{n,m_n} \rangle) = \langle \alpha_1, \ldots, \alpha_m \rangle$$

   and there are $\langle w_{p,1} \ldots w_{p,m_p} \rangle$ for all $1 \leqslant p \leqslant n$ such that

   a) $A_p \overset{*}{\Longrightarrow}_G \langle w_{p,1} \ldots w_{p,m_p} \rangle$, $m_p \in \mathbb{N}$,

   b) $\langle w_1, \ldots, w_m \rangle$ can be obtained from $\langle \alpha_1, \ldots, \alpha_m \rangle$ in the following way. For all $1 \leqslant i \leqslant m$, w_i is α_i with all occurrences of variables $x_{p,q}$ in α_i substituted by $w_{p,q}$ for all $1 \leqslant q \leqslant m_p$.

DEFINITION 2.40 (Language (LCFRS)). Let $G = (N, S, F, P)$ be a Linear Context-Free Rewriting System. The *language* of G is $\mathcal{L}(G) = \{w \mid S \overset{*}{\Longrightarrow}_G \langle w \rangle\}$

Of course, if only the string languages are of concern, a much more direct definition of LCFRS is possible. One can interpret the productions of the GCFG as functions which state how to compute the yield of the LHS given the yields of the RHS. This view is adopted in more recent work. See, e. g., Gómez-Rodríguez et al. (2009a).

EXAMPLE 2.41 (Linear Context-Free Rewriting System). As an example, consider the LCFRS $G = (\{S, A\}, S, \{f_1, f_2, f_3, f_4\}, P)$ with the following productions in P and their associated equations.

$$\begin{array}{lllll}
S & \to & f_1(A), & \bar{f}_1(\langle x_{1,1}, x_{1,2} \rangle) & = & \langle x_{1,1} x_{1,2} \rangle \\
A & \to & f_2(A), & \bar{f}_2(\langle x_{1,1}, x_{1,2} \rangle) & = & \langle a x_{1,1}, a x_{1,2} \rangle \\
A & \to & f_3(A), & \bar{f}_3(\langle x_{1,1}, x_{1,2} \rangle) & = & \langle b x_{1,1}, b x_{1,2} \rangle \\
A & \to & f_4(), & \bar{f}_4() & = & \langle \varepsilon, \varepsilon \rangle
\end{array}$$

G generates the copy language $\{ww \mid w \in \{a, b\}^*\}$. The S production separates the two copies of the word, and the A productions generate the *a*s and *b*s, respectively. Note that the equations are given without a subscript indicating a formalism, because only the string-generating capacity of LCFRS is used; however, all equations adhere to the conditions given in definition 2.38.

Multiple Context-Free Grammar

Multiple Context-Free Grammar (MCFG), a formalism which is weakly equivalent to LCFRS, was independently introduced by Seki et al. (1991). The original definition of MCFG is not as general as the definition of LCFRS, MCFG has been defined as a string-rewriting formalism. The productions of an MCFG have the form $A_0 \to f[A_1, \ldots, A_k]$. Just as in the GCFG underlying an LCFRS, f is a function which is used to compute the yield of A_0 from the yields of $A_1, \ldots A_k$. The only difference between the MCFG yield functions and their LCFRS equivalents is that MCFG does not impose the bottom-up non-erasing condition (see def. 2.43). Nevertheless, MCFG is weakly equivalent to LCFRS, as Seki et al. (1991) show.

Another formalism which is equivalent to LCFRS is Simple Range Concatenation Grammar (Boullier, 1998). In the remainder of the thesis, we will adopt the Simple Range Concatenation Grammar terminology. This formalism will be introduced later in this chapter, together with a definition of its derivation trees.

Other formalisms equivalent to LCFRS

Further formalisms are equivalent to LCFRS. *Context-Free Grammatical Framework (cf-GF)*, a restricted variant of GF (a version of dependent type theory) (Ranta, 2004), is weakly equivalent to LCFRS. Both formalisms are conceptually very close (Ljunglöf, 2004). *Minimalist Grammar (MG)* (Stabler, 1997), the formalization of Chomsky's Minimalist program (Chomsky, 1995), is weakly equivalent to MCFG, as shown by Michaelis (2001a,c). Also, set-local MCTAG (see p. 35) is weakly equivalent to LCFRS (Weir, 1988).

2.2.4 *Range Concatenation Grammar*

Range Concatenation Grammar has been introduced by Pierre Boullier (Boullier, 1998).

Range Concatenation Grammar

DEFINITION 2.42 (Positive Range Concatenation Grammar). A *Positive Range Concatenation Grammar* is a tuple $G = (N, T, V, P, S)$ where the following holds.

1. N is an alphabet of predicate names with an arity function $dim : N \to \mathbb{N}$.
2. T and V are disjoint alphabets of terminals and variables.
3. P is a finite set of *clauses* of the form

$$\psi_0 \to \psi_1 \cdots \psi_m$$

with the *rank* $m \in \mathbb{N}_0$ and each of the ψ_i, $0 \leqslant i \leqslant m$, is a *predicate* of the form $A_i(\alpha_{i,1}, \ldots, \alpha_{i,dim(A_i)})$ where $A_i \in N$ and each of the $\alpha_{i,j} \in (T \cup V)^*$, $1 \leqslant j \leqslant dim(A)$ is an argument. $A_i(\alpha_{i,1}, \ldots, \alpha_{i,dim(A_i)})$ is abbreviated as $A_i(\vec{\alpha_i})$, $\psi_1 \cdots \psi_m$ is abbreviated as $\vec{\Psi}$.

4. $S \in N$ is the distinguished start predicate name with $dim(S) = 1$.

Since Negative Range Concatenation Grammar (NRCG) will not be used in this thesis (see Boullier (1998) for details), henceforth by Range Concatenation Grammar (RCG), Positive Range Concatenation Grammar is meant. Depending on the particular form of clauses, we distinguish several subtypes of RCGs.

DEFINITION 2.43 (Subtypes of RCGs). Let $G = (N, T, V, P, S)$ be an RCG.

1. G is a k-*RCG* iff its maximal predicate arity is k; G has the *rank* m iff its maximal clause rank is m.
2. G is *non-combinatorial* if for all $c \in P$, it holds that the length of all arguments of RHS predicates is 1.
3. G is *bottom-up non-erasing* if for all $c \in P$, it holds that every variable which occurs on the RHS also occurs on the LHS.
4. G is *top-down non-erasing* if for all $c \in P$, it holds that every variable which occurs on the LHS also occurs on the RHS.
5. G is *non-erasing* if it is bottom-up non-erasing and top-down non-erasing.
6. G is *bottom-up linear* if for all $c \in P$, no variable appears more than once on the LHS of c.
7. G is *top-down linear* if for all $c \in P$, no variable appears more than once on the RHS of c.
8. G is *linear* if it is bottom-up linear and top-down linear.
9. A clause $c \in P$ is an ε-*clause* if there is an argument of its predicates which is the empty word. G is ε-*free* if either P contains no ε-clauses or there is exactly one clause $S(\varepsilon) \rightarrow \varepsilon \in P$ and S does not appear in any of the RHSs of the clauses in P.

Clauses in an RCG can be *instantiated*. In an instantiated clause, all occurrences of terminals and ε, and the variables, are mapped to ranges of a string. For this, we first must define the ranges of a string.

Ranges and instantiation

DEFINITION 2.44 (Range). Let Σ be an alphabet. For every $w \in \Sigma^*$, such that $w = w_1 \cdots w_n$ with $n \in \mathbb{N}$ and $w_i \in \Sigma$ for $1 \leqslant i \leqslant n$, we define:

1. $Pos(w) = \{0, \ldots, n\}$.
2. A *range* in w is a pair $\langle l, r \rangle \in Pos(w) \times Pos(w)$ with $l \leqslant r$. Its *yield* $\langle l, r \rangle(w)$ is the substring $w_{l+1} \cdots w_r$.
3. For two ranges $\rho_1 = \langle l_1, r_1 \rangle, \rho_2 = \langle l_2, r_2 \rangle$: if $r_1 = l_2$, i. e., if the ranges are adjacent, then $\rho_1 \cdot \rho_2 = \langle l_1, r_2 \rangle$; otherwise $\rho_1 \cdot \rho_2$ is undefined.

A range vector is a vector of ranges in a certain string.

DEFINITION 2.45 (Range vector). Let Σ be an alphabet and let $w \in \Sigma^*$. A *range vector* of dimension $k \in \mathbb{N}$ in w is defined as $\vec{\phi} \in (Pos(w) \times Pos(w))^k$, where $\langle l_i, r_i \rangle \in \vec{\phi}$, $1 \leqslant i \leqslant k$, is a range in w. $\vec{\phi}(i)$ denotes $\langle l_i, r_i \rangle$ and $\vec{\phi}(i).l$ (resp. $\phi(i).r$) denotes then the first (resp. second) component of $\vec{\phi}(i)$, that is l_i (resp. r_i).

In order to instantiate a clause of the grammar, we need to find ranges for all variables in the clause, for all occurrences of terminals, and for all ε arguments. We first introduce a numbering of argument elements.

DEFINITION 2.46 (Argument numbering). Let $G = (N, T, V, P, S)$ be an RCG.

1. For all $c \in P$, we assume all occurrences of terminals, all occurrences of ε and the resp. first occurrences of variables in c to be consecutively numbered with a unique index w.r.t. their occurrence from left to right, starting with 1.
2. For any $c \in P$, the function ξ yields the maximal index of c. The function ξ_e yields the variable, occurrence of a terminal, or occurrence of ε in c at index i for all $c \in P$, $1 \leqslant i \leqslant \xi(c)$.

Now we can give an instantiation by a range vector, using the argument numbering we have just defined to map elements of the clause to ranges in the range vector.

DEFINITION 2.47 (Clause instantiation). Let $G = (N, T, V, P, S)$ be an RCG and let $w \in T^*$ be a string.

1. An *instantiation* of some clause $c \in P$ with $\xi(c) = k$ with respect to w is given by a range vector $\vec{\phi}$ of dimension k where $\vec{\phi}(j)$, $1 \leqslant j \leqslant k$ contains the range to which $\xi_e(c, j)$ is bound. Thereby,
   a) variables which are adjacent in c must be mapped on adjacent ranges, and
   b) for $0 \leqslant i < |w|$,
      i. if $\xi_e(c, j)$ is the occurrence of a terminal a, it must be mapped on a range $\langle i, i+1 \rangle$ with $w_{i+1} = a$, and
      ii. if $\xi_e(c, j)$ is an occurrence of ε, it must be mapped on a range $\langle i, i \rangle$.
2. Applying $\vec{\phi}$ to a predicate $A(\vec{\alpha})$ in c, notated $\vec{\phi}(A(\vec{\alpha}))$, is defined as a mapping of all occurrences of terminals and ε and all variables in $\vec{\alpha}$ to elements of $\vec{\phi}$ such that $\xi_e(c, i)$ is mapped to $\vec{\phi}(i)$.

   If the result is defined, i. e., if the images of adjacent variables can be concatenated, then it is called an *instantiated predicate*. If the result of applying $\vec{\phi}$ to all predicates in c is defined, then the result of the mapping is called an *instantiated clause*.
3. The notation $A(\vec{\psi})$ designates a predicate $A(\vec{\alpha})$, instantiated with $\vec{\phi}$.

An RCG derivation consists of rewriting instantiated predicates applying instantiated clauses.

Derivation and language

DEFINITION 2.48 (Derivation (RCG)). Let $G = (N, T, V, P, S)$ be an RCG and $w \in T^*$ a string.

1. $\Longrightarrow_{G,w}$ is a relation called *derives* on strings of instantiated predicates the following way. Let Γ, Γ' be strings of instantiated predicates. If $A_0(\vec{\psi_0}) \rightarrow A_1(\vec{\psi_1}) \ldots A_m(\vec{\psi_m})$, $m \in \mathbb{N}_0$, is the instantiation of some clause $c \in P$, then

$$\Gamma A_0(\vec{\psi_0})\Gamma' \Longrightarrow_{G,w} \Gamma A_1(\vec{\psi_1}) \ldots A_m(\vec{\psi_m})\Gamma'.$$

   We may write $\overset{A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})}{\Longrightarrow_{G,w}}$ to make the clause explicit. If G and w are not relevant or implicitly understood, then we may omit the corresponding subscript.

2. $\overset{*}{\Longrightarrow}_{G,w}$ is the reflexive transitive closure of $\Longrightarrow_{G,w}$.

3. Let $\Gamma_1, \ldots, \Gamma_n$ be strings of instantiated predicates, $n \in \mathbb{N}$. A sequence
$$\Gamma_1 \Longrightarrow_{G,w} \cdots \Longrightarrow_{G,w} \Gamma_n$$
is a *derivation* of length n, also written $\Gamma_1 \overset{n}{\Rightarrow}_{G,w}$. Γ_n is called *derivable* from Γ_1. Each $\Gamma_i \Longrightarrow_{G,w} \Gamma_j$, $1 \leqslant i, j \leqslant m$, $j = i + 1$, is a *derivation step*.

The *language* of an RCG G is the set of strings that can be reduced to the empty word.

Definition 2.49 (Language (RCG)). Let $G = (N, T, V, P, S)$ be an RCG. The *language* of G is $\mathcal{L}(G) = \{w \mid S(\langle 0, |w| \rangle) \overset{*}{\Longrightarrow}_{G,w} \varepsilon\}$.

In fact, for a w, the set of instantiated clause w.r.t. w is a CFG G' with start symbol $S(\langle 0, |w| \rangle)$ (this is the grammar which generates the shared forest, see p. 54). If the language is not empty ($\{\varepsilon\} = L(G')$), then $w \in L(G)$.

Example 2.50 (Range Concatenation Grammar). RCG can generate languages beyond mild context-sensitivty. An example for such a language is $L_{exp} = \{a^{2^n} \mid n \in \mathbb{N}_0\}$ (it lacks the constant growth property). L_{exp} can generated by an RCG $G = \langle \{S, eq\}, \{a\}, \{X, Y\}, P, S \rangle$ with P containing the following clauses.

$$\begin{aligned} S(XY) &\rightarrow S(X)eq(X,Y) \\ S(a) &\rightarrow \varepsilon \\ eq(aX, aY) &\rightarrow eq(X,Y) \\ eq(\varepsilon, \varepsilon) &\rightarrow \varepsilon \end{aligned}$$

The eq clauses check two strings of *a*s for equal length: Only a string with an even number of *a*s can be reduced to ε. The S clauses split the string and apply the eq predicate.

RCG can be used for linguistic modeling (see section 3.1.1). For an elaborate linguistic example, consult Sagot (2005), section 4.1.

The most important sub-class of RCGs in this thesis are Simple Range Concatenation Grammars.

Simple RCG

Definition 2.51 (Simple Range Concatenation Grammar). A *Simple Range Concatenation Grammar (SRCG)* is an RCG which is linear, non-erasing, and non-combinatorial.

Intuitively, an SRCG clause can be seen as a combined notation of an LCFRS production together with the corresponding yield function. See example 2.56.

PROPOSITION 2.52. For every Simple Range Concatenation Grammar, there is a weakly equivalent Linear Context-Free Rewriting System and vice versa (Boullier, 1998).

A way of encoding SRCG (and also RCG) derivations as trees is presented by Boullier (1998). He uses the derivation trees of the previously mentioned CFG consisting of the set of instantiated clauses w.r.t. a word w (following def. 2.23, p. 28). In contrast, we define the derivation trees for SRCG such that in a derivation tree for a derivation of a word $w \in \mathcal{L}(G)$, G being some SRCG, leaf nodes are introduced in the tree for all terminals and ε used in instantiations, along with edges from the corresponding predicates to these nodes. The leaves are ordered.

Derivation trees

DEFINITION 2.53 (Derivation tree (SRCG)). Let $G = (N, T, V, P, S)$ be an SRCG and let $w = w_1 \cdots w_n$, $n \in \mathbb{N}$, with $w_i \in T$ for all $1 \leqslant i \leqslant n$ be a string. A *derivation tree* for G for a derivation $S(\langle 0, n\rangle) \Longrightarrow_{G,w} \varepsilon$ is an ordered tree $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D}, r)$ with a node labeling Λ over $N \cup T \cup \{\varepsilon\}$ and a function $pos : \{\nu \in V_\mathcal{D} \mid f_{out}(\nu) = 0\} \to \mathbb{N}$ such that the following holds.

1. Λ is as follows.
   a) For all $\nu \in V_\mathcal{D}$, if $f_{out}(\nu) > 0$, then $\Lambda(\nu) \in N$, otherwise $\Lambda(\nu) \in T \cup \{\varepsilon\}$.
   b) For all w_i, there is exactly one $\nu \in V_\mathcal{D}$ with $\Lambda(\nu) = w_i$.
2. For all $\nu_0, \nu_1, \ldots, \nu_k \in V_\mathcal{D}$, $k \in \mathbb{N}$, with $\langle \nu_0, \nu_j\rangle \in E$, $1 \leqslant j \leqslant k$, such that there are no other $\nu_d \in V_\mathcal{D} \setminus \{\nu_1, \ldots, \nu_k\}$ with $\langle \nu_0, \nu_d\rangle \in E_\mathcal{D}$, there exists a clause $c = A_0(\vec{\alpha}_0) \to A_1(\vec{\alpha}_1) \cdots A_m(\vec{\alpha}_m) \in P$, $m \in \mathbb{N}_0$, which is instantiable with respect to w by range vector $\vec{\phi}$, such that
   a) $|\{p \mid 1 \leqslant p \leqslant |\vec{\phi}| \text{ and } \xi_e(c, p) \text{ is no variable}\}| = k - m$,
   b) $\Lambda(\nu_0) = A_0$, and for all $1 \leqslant q \leqslant m$, there is exactly one $\nu' \in \{\nu_1, \ldots, \nu_k\}$ with $\Lambda(\nu') = A_q$,
   c) for all $1 \leqslant p \leqslant |\vec{\phi}|$ where $\xi_e(c, p)$ is a occurrence of a terminal t or of ε, there is a exactly one $\nu'' \in \{\nu_1, \ldots, \nu_k\}$ such that

i. $\Lambda(\nu'') = \xi_e(c, p)$, and

ii. $pos(\nu'') = \vec{\phi}(p).l$.

3. $\prec$ is such that for all $u', u'' \in \{u \in V_{\mathcal{D}} \mid \Lambda(u) \in T \cup \{\varepsilon\}\}$, $u' \prec u''$ iff $pos(u') < pos(u'')$.

Again, $\mathcal{D}$ is also called a $\Lambda(r)$-*tree*. If $\Lambda(r) \neq S$, then $\mathcal{D}$ is called a *partial derivation tree*.

A comparable alternative definition to our definition 2.53 can be found in Kracht (2003), p. 409.

Imposing an ordering on variables can be useful for parsing (Villemonte de la Clergerie, 2002; Kallmeyer, 2010b).

DEFINITION 2.54 (Ordered Simple Range Concatenation Grammar). An *Ordered Simple Range Concatenation Grammar (OSRCG)* is an SRCG $G = (N, T, V, P, S)$ where for all $\psi_0 \to \psi_1 \cdots \psi_m \in P$, it holds that if a variable X_1 precedes a variable X_2 in a ψ_i, $1 \leqslant i \leqslant m$, then X_1 also precedes X_2 in ψ_0.

LEMMA 2.55. Every SRCG can be transformed into an equivalent Ordered SRCG (Kallmeyer, 2010b, p. 145).

Note, however, that the corresponding construction yields a grammar which is exponential in size. Other instances of the idea of variable ordering are *monotone LCFRS* (Kracht, 2003, p. 408) and *MCFG in Monotone Function Form (MFF)* (Michaelis, 2001b, p. 21).

In *well-nested* SRCG, certain "crossing" variable configurations are disallowed (Kanazawa, 2009b). This variant of SRCG will be introduced later (def. 5.13, p. 140).

EXAMPLE 2.56 (Simple Range Concatenation Grammar). As an example, consider the following SRCG $G = (\{S, A\}, \{a, b\}, \{X, Y\}, P, S)$ where P contains the following clauses.

$$\begin{aligned} S(XY) &\to A(X, Y) \\ A(aX, aY) &\to A(X, Y) \\ A(bX, bY) &\to A(X, Y) \\ A(\varepsilon, \varepsilon) &\to \varepsilon \end{aligned}$$

Again, $\mathcal{L}(G)$ is the copy language. Note that the grammar is ordered. It works in parallel to the corresponding LCFRS in example 2.41. Figure 12 shows the derivation tree for G over *abaaba*.

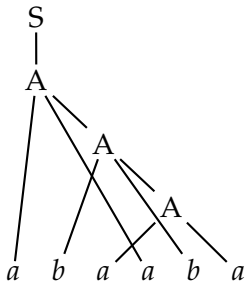


Figure 12: An SRCG derivation tree

RCG and other formalisms

Boullier (1998), pp. 18, presents the relation between RCG and other formalisms.

RCG has been introduced as a variant of *Literal Movement Grammar (LMG)* (Groenink, 1996). The definitions are very similar. The crucial difference is that while in RCG, variables are bound to ranges, in LMG they are bound to strings. While unrestricted LMG is not computationally tractable, a restricted version called *simple LMG* (Groenink, 1997) is, in fact, weakly equivalent to RCG.

Simple 1-RCG is strongly equivalent to CFG. Full 1-RCG can still be parsed in cubic time (see section 2.3.2), but offers more expressivity through copying (Boullier, 2000a).[17]

SRCG is equivalent to LCFRS, MCFG, MG and *cf*-GF. Well-nested SRCG (def. 5.13, p. 140) is equivalent to *Coupled Context-Free Grammar (CCFG)* (Hotz and Pitsch, 1994).

Boullier furthermore presents conversion algorithms from TAG and MCTAG to RCG. Kallmeyer and Parmentier (2008) present a conversion algorithm from TT-MCTAG to RCG. It has found an application in the TuLiPA parser (Kallmeyer et al., 2008b).

2.2.5 *Beyond RCG*

Before Shieber's proof of the non-context-freeness of natural language and before the introduction of the concept of Mild Context-Sensitivity, two extensions of CFG appeared with the particular aim of providing means for the handling of free word order, resp. discontinuous constituents.

17 This only holds if one discards the cost of finding instantiations. See the discussion in section 3.4.

Generalized Phrase Structure Grammar

The framework of *Generalized Phrase Structure Grammar (GPSG)* extends CFG with additional mechanisms that facilitate the expression of linguistic abstractions (Gazdar et al., 1985). In fact, these mechanisms are a means of specifying an underlying CFG: Every GPSG can be *expanded* into a weakly equivalent CFG called *object grammar*. In the context of this work, the *Immediate Dominance/Linear Precedence (ID/LP)* formalism, an extension of GPSG, is particularly interesting (Shieber, 1984). It separates the specification of dominance relations from the specification of precedence relations. This way, constituency structures can be modeled independently from word order. This is particularly useful for free word order languages. One specifies an ID/LP grammar in terms of *ID rules* and *LP rules*. ID rules (specifying immediate dominance) take the form of context-free rules the RHSs of which are specified as a set. LP rules specify linear precedence across all ID rules of the grammar. E. g., in order to state that a category A precedes a category B in all rules, one would write $A \prec B$. Note that in CFG it is not possible to generalize over linearization in a grammar-wide manner.

ID/LP

The disadvantage of ID/LP is its parsing complexity. The parsing algorithm of Shieber (1984) offered greatly improves over previous work which used the object grammar for parsing by refraining from compiling the entire object grammar. Using *Unordered Context-Free Grammar (UCFG)* (basically ID/LP without any LP rules), Barton Jr. (1985) finally proved that parsing ID/LP is NP complete (cf. section 2.3.2).

ID/LP triggered a number of linguistic works on free word order and discontinuous constituents. A complete review of this area is beyond the scope of this work; some of it will be covered in section 5.4.

Discontinuous Phrase Structure Grammar

Discontinuous Phrase Structure Grammar (DPSG) is a formalism inspired by the ID/LP tradition. The original definition of DPSG (Bunt et al., 1987; Bunt, 1991, 1996; Plaehn, 1999, 2004) starts from the idea of defining trees with discontinuous constituents and then develops grammar rules which derive them. The crucial difference between DPSG rules and SRCG rules is that DPSG rules explicitly specify the material which can occur in gaps while SRCG does not.

DPSG

Discotrees

The definition of DPSG is based on discontinuous trees, nicknamed *discotrees*. They are in turn composed of *subdiscotrees*. Intuitively speaking, a subdiscotree is a labeled tree in which the dominance relation is not completely defined, i. e., a node in a subdiscotree can have "loose" daughters, called *context daughters*. A discotree is a subdiscotree in which every node is dominated by some other node in the tree. In order to obtain crossing branches, node precedence must be defined accordingly. The children of all nodes of subdiscotrees are ordered; additionally, if one node v_1 precedes another node v_2 according to the ordering, the leftmost daughter of v_1 precedes the leftmost daughter of v_2. Two nodes v, v' form an *adjacency pair* if v precedes v' and the leftmost daughter of v' is the first node on the right of the leftmost daughter of v which is not dominated by v. They are *adjacent* if they are an *adjacency pair* or if they connected by a sequence of adjacency pairs.

On the basis of the definitions of discotrees, subdiscotrees and adjacency pairs, productions are specified which derive[18] discotrees. Those productions look like CFG productions (i. e., they are supposed to be subtrees of height one in a derivation tree), with the difference that elements of the RHS can be marked as context daughters (by enclosing them in squared brackets), and are interpreted to end up as adjacent nodes in the derived discotree.

Example 2.57 (DPSG). Here is an example, borrowed from Bunt (1996). The discotree in figure 13 can be generated by the following grammar:

VP	→	V NP			
V	→	VS [DET] [N] PART			
NP	→	DET N			
VS	→	wake	N	→	friend
DET	→	your	PART	→	up

The parsing complexity of DPSG is exponential (Plaehn, 2004) (see section 2.3.2). Vogel and Bunt (1992), resp. Vogel and Erjavec (1994) introduce a restricted version of DPSG, called *DPSG*^{r}. The latter present arguments which suggest that DPSG^r is less expressive than TAG, but more expressive than CFG. They remark that this would make it parsable in no worse than $\mathcal{O}(n^7)$, due to the applicability of a recog-

18 There does not seem to be an explicit definition of a derivation relation anywhere in the literature on DPSG.

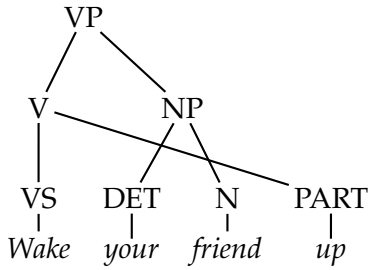


Figure 13: A discotree

nition algorithm for MCS languages of Pollard (1984). While DPSG^r fulfills the goal of being ideally suited for modeling certain types of discontinuity while remaining tractable, it cannot model cross-serial dependencies (i. e., the copy language). The work on probabilistic parsing by Plaehn will be reviewed in chapter 4.

2.2.6 *The Big Picture*

Figure 14 shows the inclusion hierarchy of some of the formalisms which have been presented so far. MCS is indicated by an arrow because the class is not clearly delimited, due to the fact that its membership conditions are necessary but not sufficient. Note that RCG generates exactly the class of languages recognizable in PTIME (cf. section 2.3.2).

2.3 SYMBOLIC PARSING

In this section, I introduce basic concepts of parsing. They are the foundation of the work which is presented in the following chapters.

2.3.1 *Parsing as Deduction*

The *recognition problem* is the problem of deciding for a grammar G and a word w over some alphabet Σ if $w \in \mathcal{L}(G)$. If the grammar is fixed, then we speak of the *fixed recognition problem*, otherwise about the *universal recognition problem*. The difference is that in the case of the latter, the complexity of the problem depends on both the size of the input string and the size of the grammar, while in the case

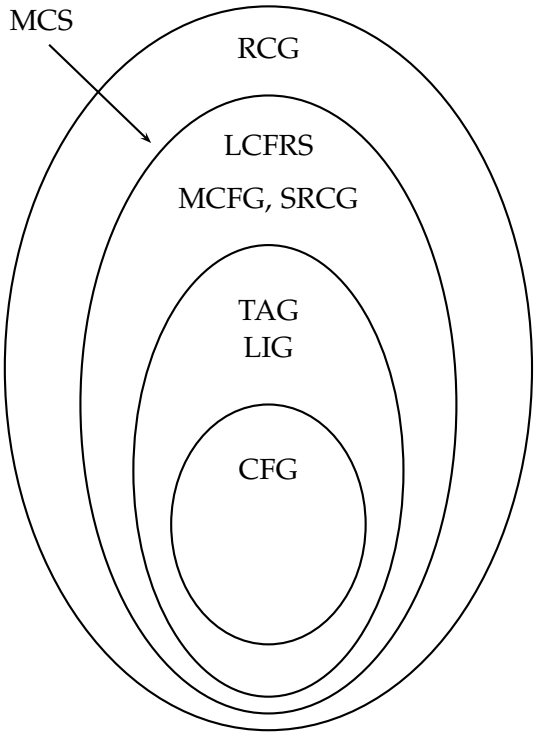


Figure 14: Inclusion hierarchy of formalisms

of the former, it depends on the length of the input string alone. A *recognition algorithm*, or just *recognizer*, is an algorithm which solves the recognition problem. A *parsing algorithm* or *parser* essentially is a recognition algorithm which records the steps that lead to a solution and outputs them in case of success. Since parsers can sometimes be implemented as an extension of the corresponding recognizer, there is often no strict distinction between the terms parser and recognizer.

Parsing algorithms can be specified in pseudo-code. Such specifications can be implemented fairly directly. However, their disadvantage is that they mix up the algorithm itself, i. e., the strategy for the solution of the recognition problem, with the control structures used for its implementation. An alternative to pseudo-code is the specification of parsing algorithms as deduction systems. This idea goes back to Lambek (1958). It has been picked up by Pereira and Warren (1983) and Shieber et al. (1995), and has been further formalized by Sikkel (1997). Deduction systems allow for parsing algorithms to be specified in a declarative way instead of in a procedural way. One can concentrate on the properties of an algorithm instead of concentrating on its implementation. Intuitively speaking, the formulation of a parsing algorithm as deduction system requires

1. a characterization of partial parsing results, or, in other words, a way of stating the grammatical status of strings, as so-called *items*,[19]

2. *inference rules* which determine how new items can be built from existing ones, and

3. *goal items*, which represent complete parses.

Deduction systems

Following Shieber et al. (1995) and Nederhof (2003), we define here a *deduction system* as a finite set of *inference rules*. The general form of an inference rule is

$$\frac{A_1, \ldots, A_k}{B} \; c_1, \ldots, c_m$$

where $k, m \in \mathbb{N}_0$, the *antecedents* $A_1, \ldots, A_k$ and the *consequent* B are *items*, and $c_1, \ldots, c_m$ are *side conditions* with which a link is established

19 Also often referred to as *edges*, especially in the context of a parsing process.

between the inference rule, and the grammar and the input string. We allow null antecedents, therefore, unlike Shieber et al. (1995), we do not need axioms.[20]

Following Nederhof (2003), we interpret a deduction system which refers to productions of a grammar G and to an input string w in its side conditions as a construction of a CFG G_ℓ out of G and w. Let $\mathbb{I}$ be the set of all possible instantiations of inference rules with productions from G and input positions in w. The productions P_ℓ of G_ℓ are obtained from $\mathbb{I}$: In each production $p \in P_\ell$, the LHS is the consequent and the RHS are the antecedents of an instantiated inference rule $I \in \mathbb{I}$. Parsing amounts to deriving ε from a goal item.

If G is ambiguous, i. e., if there is more than one way to derive w or substrings of w, then G_ℓ also contains an ambiguity. Another view is that given the input string w, G_ℓ is a compact encoding of all possible derivations of w: If two different derivations of w or of parts of w contain a common sub-derivation, then the sub-derivation is *shared* among them through the ambiguity of G_ℓ. In this context, we speak of a *shared forest* (Billott and Lang, 1989).

Parsing with a formalism $\mathfrak{F}$ beyond CFG can also be seen as the construction of a grammar G_ℓ out of G and w, where G is a grammar in formalism $\mathfrak{F}$ and w is an input string. This has been shown, e. g., for TAG (Kallmeyer and Satta, 2009) and RCG (Boullier, 1998), and will also be the basis for the presentation of the parsers for SRCG and RCG in chapter 3.

CYK and Earley algorithms

As a first example, consider the CYK algorithm for CFG in Chomsky Normal Form (Cocke and Schwartz, 1970; Younger, 1967; Kasami, 1965). CYK is a non-directional bottom up parsing technique.

Example 2.58 (CYK algorithm for CFG). Let Σ be an alphabet. Given a CFG $G = (N, T, P, S)$ in CNF and an input string $w = w_1 \cdots w_n$, $n \in \mathbb{N}$, with $w_i \in \Sigma$ for $1 \leqslant i \leqslant n$, the CYK algorithm recognizes items of the form $[A, i, j]$, where $A \overset{*}{\Longrightarrow} w_{i+1} \cdots w_j$, $0 \leqslant i < j$ and

20 Note that the formalization of parsing as deduction of Sikkel (1997) is closely related. He calls deduction systems *parsing systems*, parsing systems which are instantiated to a grammar and an input string are called *parsing schemata* and inference rules are called *deduction steps*. The latter are notated as sets of all possible instances of inferences. In principle, Sikkel's formalization could have been used for this thesis. However, especially the novel algorithms in chapter 3 would potentially have been more unintuitive to notate due to the particular form of items which is used there.

$A \in N$. A first deduction rule for terminal *scanning* introduces an item for each terminal of the input string:

$$\textsc{scan} \quad \frac{}{[A, i-1, i]} \; A \to w_i \in P$$

A second deduction rule *completes* two items which represent non-terminals from which adjacent parts of the input string can be derived into a new item, in case a matching production is present in the grammar.

$$\textsc{complete} \quad \frac{[B, i, j], [C, j, k]}{[A, i, k]} \; A \to BC \in P$$

The input string is recognized if the *goal item* can be derived.

$$\textsc{goal} \quad [S, 0, n]$$

The disadvantage of the CYK algorithm is that in some cases, many items are produced which do not lead to the goal item. As a second example, we look at the *Earley algorithm* for CFG (Earley, 1970), which remedies this problem to a certain extent.

Example 2.59 (Earley algorithm for CFG). Let $G = (N, T, P, S)$ now be a CFG which is not necessarily in CNF, and let w be the input string as in example 2.58. The Earley algorithm produces items of the form $[A \to \alpha \bullet \beta, i, j]$ where $0 \leqslant i \leqslant j$, $A \to \alpha\beta \in P$, and $\alpha \overset{*}{\Longrightarrow} w_{i+1} \cdots w_j$ as well as $S \overset{*}{\Longrightarrow} w_1 \cdots w_i A\gamma$ for some $\gamma \in (N \cup T)^*$. The Earley algorithm processes the input string strictly from left to right. The dot marks the position up to which a production has been recognized, i. e., from the part of the production on the right of the dot, a continuous part of the input string can be derived.

The initialize rule predicts unrecognized productions which have the start symbol on their LHS.

$$\textsc{initialize} \quad \frac{}{[S \to \bullet\alpha, 0, 0]} \; S \to \alpha \in P$$

If in an item, the dot is in front of some non-terminal B on the RHS of the production, we can predict new items with productions which have B on their LHS and cover a part of the input string which is adjacent to the part recognized so far.

$$\textsc{predict} \quad \frac{[A \to \alpha \bullet B\beta, i, j]}{[B \to \bullet\gamma, j, j]} \; B \to \gamma \in P$$

We can move the dot over a terminal a if in the input this terminal is present at the indices of the current item.

$$\textsc{scan} \quad \frac{[A \to \alpha \bullet a\beta, i, j]}{[A \to \alpha a \bullet \beta, i, j+1]} \; w_{j+1} = a$$

If in an item, the dot is in front of some non-terminal B on the RHS of the production, and we have recognized already a non-terminal B from which a part of the input string can be derived which is adjacent to the right, we can move the dot over the non-terminal and update the indices accordingly.

$$\textsc{complete} \quad \frac{[A \to \alpha \bullet B\beta, i, j], [B \to \gamma\bullet, j, k]}{[A \to \alpha B \bullet \beta, i, k]}$$

The goal is to completely recognize a production with the start symbol on its LHS.

$$\textsc{goal} \quad [S \to \alpha\bullet, 0, n] \text{ with } S \to \alpha \in P$$

Chart parsing

For the actual parser implementation, normally, *dynamic programming* (Bellman, 1957) is used. Roughly, this means that we keep a set of partial results which can be reused (*computation sharing*). The canonical storage structure is called *chart*. Parsing with a chart is consequently often called *chart parsing*. The advantage of parsing as deduction, as described above, is that the items give a natural formalization of partial results. In order to store the items of the Earley algorithm for CFG, e. g., one can use a four-dimensional boolean matrix $\mathcal{C}$. Assuming that we have uniquely numbered all productions $A \to \alpha$, we can store some item $[A \to \alpha \bullet \beta, i, j]$ in $\mathcal{C}$ by setting (i, j, k, l) to TRUE, where k is the index of $A \to \alpha$ and l is the position of the dot in α. The canonical way of filling $\mathcal{C}$ is as depicted in algorithm 1. It involves an additional data structure $\mathcal{A}$ called *agenda*.

If we just want to recognize the input string, it is enough to check if $\mathcal{C}$ contains a goal item. In order to obtain a real parser, we must construct G_f. In order to achieve this, for each item which was the consequent in an instantiated inference rule, we must keep track of its antecedents. These references to antecedents are called *backpointers*. In

Let G be a grammar in formalism $\mathfrak{F}$, and let w be the input string
$\mathcal{C} = \emptyset$
$\mathcal{A} = \emptyset$
for all items i which are consequents of inference rules with empty antecedent instantiated with prod. from G and positions in w **do**
  add i to $\mathcal{A}$ and $\mathcal{C}$
end for
while $\mathcal{A} \neq \emptyset$ **do**
  remove an item i from $\mathcal{A}$
  for all items i' which are consequents of inference rules with i and eventually other items from $\mathcal{C}$ in their antecedent, instantiated with prod. from G and positions in w **do**
    if $i' \notin \mathcal{C}$ **then**
      add i' to $\mathcal{C}$ and $\mathcal{A}$
    end if
  end for
end while

Algorithm 1: Filling the chart

order to reflect ambiguity, obviously, every item in the chart needs a list of them. Assuming that we have exactly one goal item, a parse of w is then obtained by following the backpointers in the chart starting at the goal item.

Example 2.60 (Parsing trace). Let $G = (\{S, A\}, \{a, b\}, P, S)$ be a CFG with $P = \{S \to aSb, S \to ab\}$. $\mathcal{L}(G) = \{a^n b^n \mid n \geqslant 1\}$. Table 1 shows all items which are generated (a *trace*) when parsing the input string *aabb*. Note that the items which lead to the goal items are only 1, 3, 6, 8, 11, 12 and 13.

Figure 15 shows the chart for the trace in table 1. The arrows symbolize the backpointers. Note that a single backpointer points to possibly more than one item if it denotes the application of an inference rule which has more than one item in its antecedent.

2.3.2 *Parsing and Complexity*

Parsing complexity

The *complexity* of a parsing algorithm is characterized in relation to the size of the input string and, only in case of the universal recognition problem, of the grammar. The following *complexity classes* are important for this work (Hopcroft and Ullman, 1979).

NO.	ITEM	RULE	ANTECEDENTS
1	$[S \to \bullet aSb, 0, 0]$	INITIALIZE	none
2	$[S \to \bullet ab, 0, 0]$	INITIALIZE	none
3	$[S \to a \bullet Sb, 0, 1]$	SCAN	1
4	$[S \to a \bullet b, 0, 1]$	SCAN	2
5	$[S \to \bullet aSb, 1, 1]$	PREDICT	3
6	$[S \to \bullet ab, 1, 1]$	PREDICT	3
7	$[S \to a \bullet Sb, 1, 2]$	SCAN	5
8	$[S \to a \bullet b, 1, 2]$	SCAN	6
9	$[S \to \bullet aSb, 2, 2]$	PREDICT	7
10	$[S \to \bullet ab, 2, 2]$	PREDICT	7
11	$[S \to ab \bullet, 1, 3]$	SCAN	8
12	$[S \to aS \bullet b, 0, 3]$	COMPLETE	3 and 11
13	$[S \to aSb \bullet, 0, 4]$	SCAN	12: GOAL

Table 1: Trace for CFG Earley example

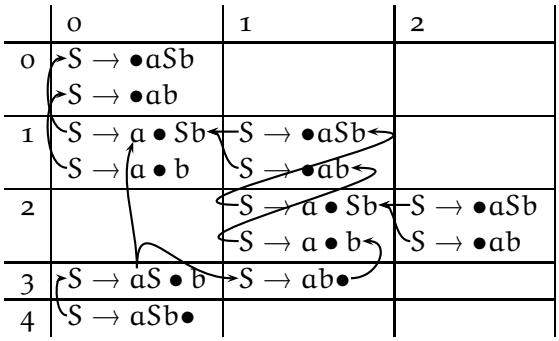


Figure 15: Chart for CFG Earley example

PTIME contains problems the solutions of which can be found in an amount of time which is polynomial with respect to the size of the input. The complexity of an algorithm is $\mathcal{O}(n^k)$ if there is a constant c and a $k \in \mathbb{N}$ such that the parsing of a string of length n takes an amount of time $\leqslant cn^k$. $\mathcal{O}(n)$ is called *linear time*, $\mathcal{O}(n^2)$ is called *quadratic time*, $\mathcal{O}(n^3)$ is called *cubic time*. PTIME is often equated with the class of "tractable" problems.

NP contains problems for which a given solution can be verified in deterministic polynomial time. Alternatively, NP can be defined as the problems the solutions of which can be found in polynomial time when proceeding non-deterministically.

NP-COMPLETE contains problems into which any problem in NP can be transformed in polynomial time.

The complexity of a parsing algorithm which is notated as deduction system can be determined easily. Given an input string of length n and an inference rule which manipulates k mutually independent indices, the complexity of the inference rule is usually $\mathcal{O}(n^k)$. This is the case since the number of independent indices determine the number of instantiations of inference rules, and the number of instantiations of inference rules determines the time complexity since it specifies the possible number of different operations. For instance, the complexity of the CYK algorithm for CFG, e. g., is $\mathcal{O}(n^3)$, since the COMPLETE rule involves 3 independent indices i, j and k.

Since this thesis does not contain contributions to complexity theory, the reader is referred to Hopcroft and Ullman (1979) for more details.

3

SYMBOLIC PARSING BEYOND CONTEXT-FREE GRAMMAR

This chapter takes up the problem of symbolic Range Concatenation Grammar (RCG) parsing and symbolic Simple Range Concatenation Grammar (SRCG) parsing.

In section 3.1, I introduce an Earley-style parsing algorithm for RCG. RCG has interesting computational properties. Inter alia, it generates exactly the class of languages recognizable in polynomial time. It is therefore a candidate for modeling linguistic phenomena which lie beyond mild context-sensitivity. In section 3.2, I present an incremental parsing strategy for Simple Range Concatenation Grammar. SRCG, which is equivalent to Linear Context-Free Rewriting System (LCFRS) and Multiple Context-Free Grammar (MCFG), is a mildly context-sensitive formalism. Apart from the applications of its probabilistic variant which are described in later chapters of this thesis, it has been used as a pivot formalism for parsing other mildly context-sensitive formalisms, such as Tree-Adjoining Grammar (TAG) variants. Related work is discussed in section 3.3. Section 3.4 concludes this chapter.

Material in this chapter has been previously published in Kallmeyer et al. (2009), Kallmeyer et al. (2009), Parmentier and Maier (2008) and Kallmeyer and Maier (2009).

3.1 PARSING RANGE CONCATENATION GRAMMAR

In this section, I present top-down algorithms, CYK-style algorithms and an Earley-style algorithm for RCG, formulated as deduction systems.

3.1.1 *Introduction*

Over the years, the parsing community has produced extensive work on symbolic parsing algorithms for formalisms beyond Context-Free Grammar (CFG). For TAG (p. 32), there exist CYK (Vijay-Shanker and

Joshi, 1985; Kallmeyer and Satta, 2009), Earley-style (Schabes and Joshi, 1988; Joshi and Schabes, 1997; Nederhof, 1997, 1999) and LR strategies (Nederhof, 1998; Prolo, 2000, 2003). Linear Indexed Grammar (LIG) (p. 36) has also found attention, see, e. g., Boullier (1996). For SRCG and formalisms which are equivalent to it, we also find different parsing strategies in the literature. To this respect, see section 3.2.

Work on RCG parsing (def. 2.42, p. 41) is scarce. See section 3.3 for a detailed overview. The relatively small amount of work might be explained with the fact that RCG is expensive to recognize. However, the fact that it is also highly expressive (beyond Mild Context-Sensitivity (MCS)) entails several advantages.

EXPRESSIVITY RCGs can be used for modeling linguistic phenomena which arguably lie beyond mild context-sensitivity. The phenomena which have been put forward in the literature include scrambling in German (Becker et al., 1992), case stacking in Old Georgian (Michaelis and Kracht, 1997) and Chinese number names (Radzinski, 1991). For the latter, Boullier (1999) presents an RCG modeling.

In RCG derivations, multiple parts of the input string can be handled at once, through copying. However, parsing complexity is not necessarily higher. Just as CFG, we can parse 1-RCG in $\mathcal{O}(n^3)$, while obtaining additional expressivity which can be used to model complex linguistic phenomena (Boullier, 2000a). In syntax-directed machine translation, this additional expressivity of RCG is useful, too (Søgaard, 2011).

Grammars of MCS formalisms can be translated into equivalent Range Concatenation Grammars (cf. p. 48). This way RCG can act as a pivot-formalism at no additional cost (modulo the cost of the conversion), i. e., the parsing complexity remains unchanged (Boullier, 1998, pp. 18).

MODULARITY Boullier (1998) explicitly proposes RCG as a *modular* formalism, in the sense of a formalism which, through its closure properties, constitutes a device for elegant linguistic modeling (Boullier, 1998, p. 3). Boullier (2000b) shows that Range Concatenation Languages (RCLs) are closed under intersection, unlike Context-Free Languages (CFLs). This is interesting, since it allows for a modularization of the description of language properties. Even though it has been ar-

gued that this property is less useful than it seems (Chiang, 2004), it is still a feature which few formalisms offer.

Range Concatenation Grammar predicates can be viewed as describing certain (linguistic) properties of its ranges in a modular way. Copying, i. e., the use of the same variable in more than one predicate, makes it possible to describe more than one property of a certain range within the same clause. Sagot (2005) takes advantage of this view and proposes a formalism called *Meta-RCG*, which is strongly equivalent to RCG and offers a notation which facilitates the description of linguistic concepts, such as lexical heads.

TRACTABLE PARSING RCG is still relatively tractable. Due to the structure of its clauses, RCGs have a context-free backbone, i. e., the choice of the operation performed at each step only depends on the object to be derived from. This way, RCG derivations can be packed into polynomially sized parse forests (see p. 54). Furthermore, as mentioned before, it has been shown by Bertsch and Nederhof (2001) that RCLs, i. e., the class of languages generated by RCG, are equal to the class of languages recognizable in PTIME (see p. 51). Even though in practice, the high cost of computing clause instantiations must be taken into account, especially those languages of which some properties are already known before parsing can be parsed very efficiently. To this respect, Boullier (2000b) presents parsing times for certain variants of 1-RCGs which lie below linear time.

The attractive properties combined with the fact that so far, there is no parsing algorithm for RCG in the literature which has been formulated as a deduction system provide the motivation for the presentation of several different parsing strategies for RCG in the following subsections.

3.1.2 *Definitions*

For ease of presentation, it is assumed without loss of generality for all grammars that empty arguments (ε) only appear in clauses with an empty right-hand side (RHS). Every RCG $G = (N, T, V, P, S)$ can easily be transformed in an RCG G' which fulfills this condition. A new unary predicate Eps is introduced together with a clause $\text{Eps}(\varepsilon) \rightarrow \varepsilon$. Then, for every clause $c \in P$ with a RHS that is not ε, replace every argument ε that occurs in c with a variable $X \in V$ (picking a different

variable with each replacement which does not occur in c) and add the predicate $\mathrm{Eps}(X)$ to the RHS.

With range vectors (def. 2.45, p. 43), we can only express complete instantiations: All instances of terminals as well as all variables in a clause must be mapped to some range. However, during parsing, we may incrementally gain more and more information about range boundaries which we would like to accumulate before determining a complete instantiation. In other words, we would like to be able to express a *partial* instantiation. This is what we will use the constraints in the range constraint vector for.

DEFINITION 3.1 (Range constraint vectors). Let $V_r = \{r_1, r_2, \dots\}$ be a set of range boundary variables.

1. A *range constraint vector* of dimension $k \in \mathbb{N}$ is a pair $\langle \vec{\rho}, C \rangle$ for which the following holds.
   a) $\vec{\rho} \in (V_r \times V_r)^k$, furthermore all range boundary variables in $\vec{\rho}$ are pairwise different. We define $V_r(\vec{\rho})$ as the set of range boundary variables occurring in $\vec{\rho}$; since those variables must be pairwise different, $|V_r(\vec{\rho})| = 2k$.
   b) Again for all $1 \leqslant i \leqslant k$, $\vec{\rho}(i)$ denotes $\langle x_i, y_i \rangle \in \vec{\rho}$ and $\vec{\rho}(i).l$ (resp. $\vec{\rho}(i).r$) denotes then the first (resp. second) component of $\vec{\rho}(i)$, that is x_i (resp. y_i).
   c) C is a set of constraints c_r that have one of the following forms: $r_1 = r_2$, $\kappa = r_1$, $r_1 + \kappa = r_2$, $\kappa \leqslant r_1$, $r_1 \leqslant \kappa$, $r_1 \leqslant r_2$ or $r_1 + \kappa \leqslant r_2$ for $r_1, r_2 \in V_r(\vec{\rho})$ and $\kappa \in \mathbb{N}_0$.
2. We say that a range vector $\vec{\phi}$ *satisfies* a range constraint vector $\langle \vec{\rho}, C \rangle$ iff $\vec{\phi}$ and $\vec{\rho}$ are of the same dimension $k \in \mathbb{N}$ and there is a function $f : V_r \to \mathbb{N}$ that maps $\vec{\rho}(i).l$ to $\vec{\phi}(i).l$ and $\vec{\rho}(i).r$ to $\vec{\phi}(i).r$ for all $1 \leqslant i \leqslant k$ such that all constraints in C are satisfied.
3. We say that a range constraint vector $\langle \vec{\rho}, C \rangle$ is *satisfiable* iff there exists a range vector $\vec{\phi}$ that satisfies it.

Intuitively, a range constraint vector of some clause captures all information about boundaries forming a range, ranges containing only a single terminal, and about adjacent variables and terminal occurrences in the clause. Recall the definition of argument numbering (def. 2.46, p. 43).

DEFINITION 3.2 (Range constraint vector of a clause). Let G be an RCG with $G = (N, T, V, P, S)$; assume the numbering from definition 2.46. For every clause $c \in P$, we define its *range constraint vector* $\langle \vec{\rho}, C \rangle$ with respect to to a string w as follows:

1. $\vec{\rho}$ has dimension $\xi(c)$ and all range boundary variables in $\vec{\rho}$ are pairwise different.
2. The constraints in C are as follows.
   a) For all $\langle r_1, r_2 \rangle \in \vec{\rho}$: $0 \leqslant r_1$, $r_1 \leqslant r_2$, $r_2 \leqslant |w| \in C$.
   b) For all occurrences t of terminals in c with $i = \xi_e(c, t)$: $\vec{\rho}(i).l + 1 = \vec{\rho}(i).r \in C$.
   c) For all x, y that are variables or occurrences of terminals in c such that xy is a substring of one of the arguments in c: $\vec{\rho}(\xi_e(c, x)).r = \vec{\rho}(\xi_e(c, y)).l \in C$.
   d) These are all constraints in C.

3.1.3 *Directional Top-Down Parsing*

The directional top-down parsing algorithm evaluates RHS predicates from left to right and stops further evaluation once a predicate fails.

For the directional top-down parsing algorithm, we need to distinguish between *passive items* and *active items*. Passive items have the form $[A, \vec{\phi}, \mathit{flag}]$, where A is a predicate, $\vec{\phi}$ is a range vector of dimension $\mathit{dim}(A)$ (containing the ranges that the arguments of A are instantiated with) and $\mathit{flag} \in \{c, p\}$ indicates if the item has been completed or predicted. Since, unlike in standard top-down parsing for context-free grammar, we already start off with the entire string at initialization time, we need a way to propagate information about successful predicates. This is achieved by the p/c-flag, which is set by the scan and the complete operations.

Active items allow us to move a dot through the RHS of a clause. The item form is $[A(\vec{x}) \to \Phi \bullet \Psi, \vec{\phi}]$ where $A(\vec{x}) \to \Phi\Psi$ is a clause and $\vec{\phi}$ is a range vector of dimension $h = \xi(A(\vec{x}) \to \Phi\Psi)$ that gives an instantiation of the clause.

The axiom is the prediction of the start predicate ranging over the entire input. The INITIALIZE rule is as follows.

$$\frac{}{[S, (\langle 0, n\rangle), p]}$$

We have two predict operations. The first one, PREDICT-RULE, predicts active items with the dot on the left of the RHS, for a given predicted passive item.

$$\frac{[A, \psi, p]}{[A(\vec{x}) \to \bullet\Psi, \vec{\phi}]}$$

with the side condition $\vec{\phi}(A(\vec{x})) = A(\psi)$, i. e., $\vec{\phi}$ instantiates $A(\vec{x})$ with $A(\psi)$.

PREDICT-PRED predicts a passive item for the predicate following the dot in an active item:

$$\frac{[A(\vec{x}) \to \Phi \bullet B(\vec{y})\Psi, \vec{\phi}]}{[B, \psi, p]}$$

with the side condition $\vec{\phi}(B(\vec{y})) = B(\psi)$.

The SCAN operation scans a terminal in the input string:

$$\frac{[A, (\langle l, r\rangle), p]}{[A, (\langle l, r\rangle), c]}$$

with the side conditions $A(x) \to \varepsilon, \langle l, r\rangle(w) = x$.

COMPLETE moves the dot over a predicate in the RHS of an active item if the corresponding passive item has been completed.

$$\frac{[B, \vec{\phi}_B, c], [A(\vec{x}) \to \Phi \bullet B(\vec{y})\Psi, \vec{\phi}]}{[A(\vec{x}) \to \Phi B(\vec{y}) \bullet \Psi, \vec{\phi}]}$$

with the side condition $\vec{\phi}(B(\vec{y})) = B(\vec{\phi}_B)$.

Once the dot has reached the right end of a clause, we can CONVERT the active item into a *completed* passive item:

$$\frac{[A(\vec{x}) \to \Phi\bullet, \vec{\phi}]}{[A, \psi, c]}$$

with the side condition $\vec{\phi}(A(\vec{x})) = A(\psi)$.

The goal is to find a completed passive item denoting the start predicate ranging over the entire input.

$$\textsc{goal} \quad [S, (\langle 0, n\rangle), c]$$

An obvious problem of this algorithm is that PREDICT-RULE has to compute all possible instantiations of A-clauses, given an instantiated A-predicate. Take for example the RCG for $\{a^{2^n} \mid n \geqslant 0\}$ from example 2.50. If $w = aaaa$, starting from $[S, (\langle 0, 4\rangle), p]$ PREDICT-RULE would predict (among others) the active items

$$[S(X_1Y_2) \to \bullet S(X_1)eq(X_1, Y_2), (\langle 0, r\rangle, \langle r, 4\rangle)]$$

for all $0 \leqslant r \leqslant 4$.

The computation of all these possible instantiations is very costly and will be avoided in the Earley algorithm that is presented in 3.1.5. The latter will use range constraint vectors (instead of range vectors) and predict only one active item $[S(X_1Y_2) \to \bullet S(X_1)eq(X_1, Y_2), \langle\langle(r_1, r_2), (r_3, r_4)\rangle, \{0 = r_1, r_1 \leqslant r_2, r_2 = r_3, r_3 \leqslant r_4, 4 = r_4\}\rangle]$.

3.1.4 *Bottom-Up Chart Parsing*

Basic CYK Parsing

CYK (p. 54) is a non-directional bottom-up parsing technique.

The items have the form $[A, \vec{\phi}]$ where A is a predicate and $\vec{\phi}$ a range vector of dimension $dim(A)$. The SCAN operation is as follows.

$$\frac{}{[A, \vec{\phi}]}$$

As a side condition, it must hold that there is a clause $c = A(\vec{x}) \to \varepsilon$ with an instantiation ψ such that $\psi(A(\vec{x})) = A(\vec{\phi})$.

COMPLETE works in parallel to the CFG variant of the algorithm.

$$\frac{[A_1, \vec{\phi}_1] \ldots [A_k, \vec{\phi}_k]}{[A, \vec{\phi}]}$$

with the side condition that $A(\vec{\phi}) \to A_1(\vec{\phi}_1) \ldots A_k(\vec{\phi}_k)$ is an instantiated clause.

The goal is again to find an item denoting the start predicate ranging over the whole input.

$$\textsc{goal} \quad [S, (\langle 0, n\rangle)]$$

Directional Bottom-Up Parsing

An obvious disadvantage of the basic CYK algorithm is that, in order to perform a COMPLETE step, all $A_1, \ldots, A_k$ in the RHS must be checked for appropriate items. This leads to a lot of indices that need to be checked at the same time.

In order to avoid this, we can again move a dot through the RHS of a clause. As in the case of the directional top-down algorithm, in addition to the items used above which we call *passive* items now, we also need *active* items. In the active items, while traversing the RHS of the clause, we keep a record of the positions already found for the left and right boundaries of variables and terminal occurrences. This is achieved by subsequently enriching the range constraint vector of the clause.

Active items have the form $[A(\vec{x}) \to \Phi \bullet \Psi, \langle \vec{\rho}, C \rangle]$ with $A(\vec{x}) \to \Phi\Psi$ a clause, $\Phi\Psi \neq \varepsilon$, $\xi(A(\vec{x} \to \Phi\Psi)) = h$ and $\langle \vec{\rho}, C \rangle$ a range constraint vector of dimension h. We require that $\langle \vec{\rho}, C \rangle$ be satisfiable.

Items that are distinguished from each other only by a bijection of the range variables are considered equivalent. I. e., if the application of a rule yields a new item such that an equivalent one has already been generated, this new one is not added to the set of partial results.

The SCAN rule is the same as in the basic algorithm. In addition, we have an INITIALIZE rule that introduces clauses with the dot on the left of the RHS.

$$\frac{}{[A(\vec{x}) \to \bullet\Phi, \langle \vec{\rho}, C \rangle]}$$

with the side condition that $A(\vec{x}) \to \Phi$ is a clause with range constraint vector $\langle \vec{\rho}, C \rangle$, $\Phi \neq \varepsilon$.

COMPLETE moves the dot over a predicate in the RHS of an active item provided the corresponding passive item has been completed.

$$\frac{[B, \vec{\phi}_B], \quad [A(\vec{x}) \to \Phi \bullet B(x_1 \ldots y_1, \ldots, x_k \ldots y_k)\Psi, \langle \vec{\rho}, C \rangle]}{[A(\vec{x}) \to \Phi B(x_1 \ldots y_1, \ldots, x_k \ldots y_k) \bullet \Psi, \langle \vec{\rho}, C' \rangle]}$$

with the side conditions $C' = C \cup \{\vec{\phi}_B(j).l = \vec{\rho}(\xi(x_j)).l,\ \vec{\phi}_B(j).r = \vec{\rho}(\xi(y_j)).r \mid 1 \leqslant j \leqslant k\}$. Note that the conditions on the items require the new constraint set for $\vec{\rho}$ to be satisfiable.

CONVERT turns an active item with the dot at the end of the RHS into a completed passive item.

$$\frac{[A(\vec{x}) \to \Psi\bullet, \langle\vec{\rho}, C\rangle]}{[A, \vec{\phi}]}$$

with the side condition that there is an instantiation ψ of $A(\vec{x}) \to \Psi$ that satisfies $\langle\vec{\rho}, C\rangle$ such that $\psi(A(\vec{x})) = A(\vec{\phi})$.

The goal is the same as in the basic CYK algorithm.

$$\text{GOAL} \quad [S, (\langle 0, n\rangle)]$$

A sample parse trace is shown in table 2. For the sake of readability, instead of the range boundary variables, we use X.l and X.r respectively for the left and right range boundary of the range associated with X.

3.1.5 *The Earley Algorithm*

We now add a prediction operation to the CYK algorithm with active items which leads to an Earley-style algorithm. As before, the passive items are enriched with an additional flag that can have values p or c depending on whether the item is only predicted or already completed. Furthermore, they contain range constraint vectors since when predicting a category, the left and right boundaries of its arguments might not be known.

Passive items either have the form $[A, \langle\vec{\rho}, C\rangle, p]$ for a predicted item, where $\langle\vec{\rho}, C\rangle$ is a range constraint vector of dimension $dim(A)$, or the form $[A, \vec{\phi}, c]$ for completed items where $\vec{\phi}$ is a range vector of dimension $dim(A)$. The active items are the same as in the CYK case.

Deduction Rules

The axiom is the prediction of an S ranging over the entire input, i. e., the INITIALIZE rule is as follows:

$$\frac{}{[S, \langle(\langle r_1, r_2\rangle), \{0 = r_1, n = r_2\}\rangle, p]}$$

We have two predict operations. The first one, PREDICT-RULE, predicts active items with the dot on the left of the RHS, for a given predicted passive item:

Grammar for $\{a^{2^n} \mid n \geqslant 0\}$: $S(XY) \to S(X)eq(X,Y)$, $S(a_1) \to \varepsilon$, $eq(a_1X, a_2Y) \to eq(X,Y)$, $eq(a_1,a_2) \to \varepsilon$. Items generated for input $w = aa$ (the constraints $0 \leqslant r_1, r_2 \leqslant n$ for a range $\langle r_1, r_2\rangle$ are omitted):

NO.	ITEM	OPERATION
1	$[S, (\langle 0,1\rangle)]$	SCAN $S(a_1) \to \varepsilon$
2	$[S, (\langle 1,2\rangle)]$	SCAN $S(a_1) \to \varepsilon$
3	$[eq, (\langle 0,1\rangle, \langle 0,1\rangle)]$	SCAN $eq(a_1,a_2) \to \varepsilon$
4	$[eq, (\langle 0,1\rangle, \langle 1,2\rangle)]$	SCAN $eq(a_1,a_2) \to \varepsilon$
5	$[eq, (\langle 1,2\rangle, \langle 0,1\rangle)]$	SCAN $eq(a_1,a_2) \to \varepsilon$
6	$[eq, (\langle 1,2\rangle, \langle 1,2\rangle)]$	SCAN $eq(a_1,a_2) \to \varepsilon$
7	$[S(XY) \to \bullet S(X)eq(X,Y), \{X.l \leqslant X.r, X.r = Y.l, Y.l \leqslant Y.r\}]$	INITIALIZE
8	$[eq(a_1X, a_2Y) \to \bullet eq(X,Y), \{a_1.l + 1 = a_1.r, a_1.r = X.l, X.l \leqslant X.r, a_2.l + 1 = a_2.r, a_2.r = Y.l, Y.l \leqslant Y.r\}]$	INITIALIZE
9	$[S(XY) \to S(X) \bullet eq(X,Y), \{\ldots, 0 = X.l, 1 = X.r\}]$	COMPLETE 7 with 1
10	$[S(XY) \to S(X) \bullet eq(X,Y), \{\ldots, 1 = X.l, 2 = X.r\}]$	COMPLETE 7 with 2
11	$[eq(a_1X, a_2Y) \to eq(X,Y)\bullet, \{\ldots, 1 = X.l, 2 = X.r, 1 = Y.l, 2 = Y.r\}]$	COMPLETE 8 with 6
12	$[S(XY) \to S(X)eq(X,Y)\bullet, \{\ldots, 0 = X.l, 1 = X.r, 1 = Y.l, 2 = Y.r\}]$	COMPLETE 9 with 4
13	$[eq, (\langle 0,2\rangle, \langle 0,2\rangle)]$	CONVERT 11
14	$[S, (\langle 0,2\rangle)]$	CONVERT 12

Table 2: Trace of an RCG CYK parse

$$\frac{[A, \langle \vec{\rho}, C \rangle, p]}{[A(x_1 \dots y_1, \dots, x_k \dots y_k) \to \bullet\Psi, \langle \vec{\rho'}, C' \rangle]}$$

with the side condition that $\langle \vec{\rho'}, C' \rangle$ is obtained from the range constraint vector of the clause $A(x_1 \dots y_1, \dots, x_k \dots y_k) \to \Psi$ by taking all constraints from C, mapping all $\vec{\rho}(i).l$ to $\vec{\rho'}(\xi(x_i)).l$ and all $\vec{\rho}(i).r$ to $\vec{\rho'}(\xi(y_i)).r$, and then adding the resulting constraints to the range constraint vector of the clause.

The second predict operation, PREDICT-PRED, predicts a passive item for the predicate following the dot in an active item:

$$\frac{[A(\dots) \to \Phi \bullet B(x_1 \dots y_1, \dots, x_k \dots y_k)\Psi, \langle \vec{\rho}, C \rangle]}{[B, \langle \vec{\rho'}, C' \rangle, p]}$$

with the side conditions $\vec{\rho'}(i).l = \vec{\rho}(\xi(x_i)).l$, $\vec{\rho'}(i).r = \vec{\rho}(\xi(y_i)).r$ for all $1 \leqslant i \leqslant k$ and $C' = \{c \mid c \in C, c \text{ contains only range variables from } \vec{\rho'}\}$.

SCAN can be applied if a predicted predicate can be derived by an ε-clause:

$$\frac{[A, \langle \vec{\rho}, C \rangle, p]}{[A, \vec{\phi}, c]}$$

with the side condition that there is a clause $A(\vec{x}) \to \varepsilon$ with a possible instantiation ψ that satisfies $\langle \vec{\rho}, C \rangle$ such that $\psi(A(\vec{x})) = A(\vec{\phi})$.

Finally, COMPLETE and CONVERT are the corresponding deduction rules from the CYK algorithm with active items except that we add flags c to the passive items occurring in these rules.

Again, the goal item is as follows.

$$\text{GOAL} \quad [S, (\langle 0, n \rangle), c]$$

To understand how this algorithm works, consider the example in table 3, which uses the RCG and the input word from table 2.

Note that the algorithm shows a great similarity to the directional top-down algorithm. The crucial difference is that while in the top-down algorithm, we are using range vectors to record the variable bindings, in the Earley-style algorithm, we use range constraint vectors. Due to the fact that range constraint vectors allow us to leave range boundaries unspecified, we can compute the value of range boundaries in a more incremental fashion since we do not have to guess all values of all boundary variables of a clause at once as in

Grammar for $\{a^{2^n} \mid n \geqslant 0\}$: $S(XY) \to S(X)eq(X,Y)$, $S(a_1) \to \varepsilon$, $eq(a_1X, a_2Y) \to eq(X,Y)$, $eq(a_1,a_2) \to \varepsilon$ Items generated for input $w = aa$ (the constraints $0 \leqslant r_1, r_2 \leqslant n$ for a range $\langle r_1, r_2\rangle$ are omitted):

NO.	ITEM	OPERATION
1	$[S, \langle(\langle r_1, r_2\rangle), \{0 = r_1, r_1 \leqslant r_2, 2 = r_2\}\rangle, p]$	INITIALIZE
2	$[S(XY) \to \bullet S(X)eq(X,Y), \{X.l \leqslant X.r, X.r = Y.l, Y.l \leqslant Y.r, 0 = X.l, 2 = Y.r\}]$	PREDICT-RULE from 1
3	$[S, \langle(\langle r_1, r_2\rangle), \{0 = r_1, r_1 \leqslant r_2\}\rangle, p]$	PREDICT-PRED from 2
4	$[S, (\langle 0, 1\rangle), c]$	SCAN from 3
5	$[S(XY) \to \bullet S(X)eq(X,Y), \{X.l \leqslant X.r, X.r = Y.l, Y.l \leqslant Y.r, 0 = X.l, \}]$	PREDICT-RULE from 3
6	$[S(XY) \to S(X) \bullet eq(X,Y), \{\ldots, 0 = X.l, 2 = Y.r, 1 = X.r\}]$	COMPLETE 2 with 4
7	$[S(XY) \to S(X) \bullet eq(X,Y), \{X.l \leqslant X.r, X.r = Y.l, Y.l \leqslant Y.r, 0 = X.l, 1 = X.r\}]$	COMPLETE 5 with 4
8	$[eq, \langle(\langle r_1, r_2\rangle, \langle r_3, r_4\rangle), \{r_1 \leqslant r_2, r_2 = r_3, r_3 \leqslant r_4, 0 = r_1, 2 = r_4, 1 = r_2\}\rangle]$	PREDICT-PRED from 6
9	$[eq(a_1X, a_2Y) \to \bullet eq(X,Y), \{a_1.l + 1 = a_1.r, a_1.r = X.l, X.l \leqslant X.r, a_2.l + 1 = a_2.r, a_2.r = Y.l, Y.l \leqslant Y.r, X.r = a_2.l, 0 = a_1.l, 1 = X.r, 2 = Y.r\}]$	PREDICT-RULE from 8
	...	
10	$[eq, (\langle 0, 1\rangle, \langle 1, 2\rangle), c]$	SCAN 8
11	$[S(XY) \to S(X)eq(X,Y) \bullet, \{\ldots, 0 = X.l, 2 = Y.r, 1 = X.r, 1 = Y.l\}]$	COMPLETE 6 with 10
12	$[S, (\langle 0, 2\rangle), c]$	convert 11

Table 3: Trace of an RCG Earley parse

the top-down algorithm. In other words, the instantiations are computed "lazily". This becomes particularly apparent when comparing the COMPLETE rules of the non-directional top-down algorithm and the Earley-style algorithm. In the former, we check the compatibility of the range vector of the completed item with the range vector of the item which is to be completed as a side condition. In the latter, we add the information contributed by the range vector of the completed item dynamically to the range constraint vector of the item to be completed. Of course, the use of constraints makes comparisons between items more expensive, which means that for an efficient implementation, an efficient representation of the constraints and adequate techniques for constraint solving are required, such as they are presented, e. g., in Schulte (2002).

The directional bottom-up parser from section 3.1.4, in contrast to the Earley algorithm, lacks the top-down predictions. However, it uses the same technique of dynamic updating of a set of constraints on range boundaries, therefore the active items are the same for the two algorithms.

Soundness and Completeness

It is easy to see that the Earley-style algorithm is both sound and complete. More precisely, if a completed item is generated, then the corresponding predicate can be derived (soundness): $[A, \psi, c] \Rightarrow A(\psi)$. Furthermore, if we can derive a constituent $A(\psi)$, we also generate the corresponding item (completeness). Let Γ be a string of instantiated predicates. Then

$$S(\langle 0, n\rangle) \stackrel{*}{\Rightarrow}_l A(\psi)\Gamma \stackrel{*}{\Rightarrow}_l \Gamma \text{ iff } [A, \psi, c]$$

where $\stackrel{*}{\Rightarrow}_l$ denotes the leftmost derivation (def. 6.4, p. 167). In particular, $[S, (\langle 0, n\rangle), c]$ iff $S(\langle 0, n\rangle) \stackrel{*}{\Rightarrow} \varepsilon$.

Obtaining a Parse Forest

So far, we have described recognizers, not parsers. The way to obtain a parse forest from the item set resulting from the Earley recognizer with range boundary constraints is rather obvious. Whenever a convert is done, a fully instantiated clause has been found. By collecting

these clauses, we obtain a compact representation of our parse forest.[1] Starting from an S predicate ranging over the entire input and following the clauses for the instantiated predicates in the RHSs, we can read off the single parse trees from this forest.

Complexity

It is clear that all the algorithms presented here are polynomial in the input size.

When dealing with RCG, one huge factor of complexity, as shown by Boullier (1998), is the maximal number of range variables occurring inside an argument of a predicate. In the Earley-style approach, the computation, resp. resolution of the values of the range variables is delayed as much as possible in order to narrow the search space. The result of this is

1. a reduction of the number of items produced by the parser (thus a reduction in space complexity) and consequently
2. a reduction of time complexity, compared with the other approaches.

We do not give here a theoretical definition of the time complexity of our algorithm, but rather give a practical evaluation of the relative cost of directional top-down and Earley-style parsing (see section 3.1.6).

The Valid Prefix Property

The Earley-style algorithm lacks the *Valid Prefix Property (VPP)*. A parsing algorithm has the VPP if for some grammar G, it only recognizes strings which are valid prefixes of some $w \in L(G)$.

As an example, consider the following grammar which generates the language $\mathcal{L} = \{aa\}$.

$$
\begin{array}{rcl@{\qquad}rcl}
S(X) & \rightarrow & A(X)B(X) & & & \\
A(cX) & \rightarrow & A(X) & A(a) & \rightarrow & \varepsilon \\
B(Xc) & \rightarrow & B(X) & B(a) & \rightarrow & \varepsilon
\end{array}
$$

1 Note that, strictly speaking, this structure is not the parse forest as it contains some clause instantiations that are not part of the actual parse forest. This is not a problem for these useless instantiations are ignored when reading the parses starting from the instantiated S predicates.

Let the input string be $w = ca$. Using the A predicates, the Earley-style algorithm would recognize the entire string before realizing that *ca* is not a prefix of *aa*.

We briefly sketch the idea behind a prefix-valid version of the Earley-style algorithm. All deduction rules would have to be adapted in order to ensure additional side conditions. For a passive items $[A, \psi, c]$, we would have to ensure that there is a $v = w_1 \dots w_i v'$ such that $S(\langle 0, |v| \rangle) \stackrel{*}{\Rightarrow} A(\psi) \stackrel{*}{\Rightarrow} \varepsilon$ with respect to v, i being the rightmost range boundary in ψ. For an active item $[A(\vec{x}) \to \Phi \bullet \Psi, \psi]$, we would have to ensure that there is a $v = w_1 \dots w_i v'$ such that $S(\langle 0, |v| \rangle) \stackrel{*}{\Rightarrow} A(\Psi) \Rightarrow \Phi\Psi' \stackrel{*}{\Rightarrow} \Psi'$ with respect to v, i being the rightmost range boundary in arguments of Ψ. Note that the additional conditions highly increase the complexity of the algorithm.

3.1.6 *Experimental Evaluation*

For experimental evaluation, the directional top-down algorithm and the new Earley-style algorithm have been implemented within the TuLiPA system[2] (Kallmeyer et al., 2008b). The algorithms are tested on different words of the language $\mathcal{L} = \{a^{2^n} | n \leqslant 0\}$, given the grammar from example 2.50. Figure 16 shows a plot of the numbers of items generated with the top-down and the Earley algorithm. We can clearly see that range boundary constraint propagation increases the amount of information transported in single items and thereby considerably reduces the number of generated items.

3.1.7 *Efficient Instantiation Computation Without Incrementality*

The incremental algorithm has the advantage that the lazy computation of instantiations is easier than the "brute-force" approach of guessing complete clause instantiations at once. However, given the right formulation of the instantiation problem, it can also be feasible to guess complete clause instantiations. A convenient technique for this purpose is the *constraint satisfaction* paradigm.[3] In the following, I present a formulation of clause instantiation within this paradigm.

2 Available at `http://sourcesup.cru.fr/tulipa/`.

3 For a good introduction to constraint solving, see Schulte (2002).

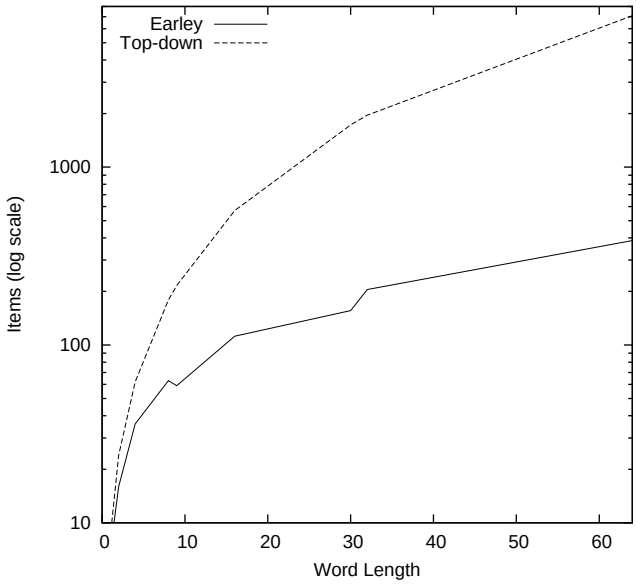


Figure 16: Items generated by RCG parsers

In the constraint satisfaction paradigm, a problem is described with a set of variables, each taking its value in a given domain. Constraints are then applied on the values these variables can take in order to narrow their respective domains. Finally, we search for one (or all) solution(s) to the problem, that is to say we search for some (or all) assignment(s) of values to variables respecting the constraints. One particularly interesting sub-class of Constraint Satisfaction Problems (CSPs) are those that can be stated in terms of constraints on variables ranging over finite sets of non-negative integers. For such CSPs, there exist several implementations offering a wide range of constraints (arithmetic, boolean and linear constraints), and efficient solvers. One example of such an implementations is the `Gecode` library.[4]

As mentioned above, an argument of a predicate to be instantiated contains range variables and/or constants, the latter acting like constraints on the boundaries between ranges. To illustrate this, consider the instantiations of the predicate $A(aXYdZ)$ with respect to the input string *abcdef*. For this example, we only have three solutions, depending on where to put the boundary between ranges X and Y. How-

4 See `http://www.gecode.org`.

ever, an instantiation of a predicate $A(XaYbZ)$ with the string *aaaabaa* would have a lot more solutions.

The idea underlying the interpretation of this instantiation task in terms of a CSP is to use the natural order of integers to represent the linear order imposed on ranges, and to define additional constraints reflecting the fact that constants (if any) are anchors for ranges of the input string. We do the following:[5]

1. we define a model which associates *boundary constants* with non-negative integers, and *boundary variables* with finite domains over non-negative integers,
2. we define constraints on these boundary variables,
3. we search for all assignments of values (i. e., non-negative integer) to these boundary variables.

STEP 1 The input string w is defined as follows:

$$w = b_0 s_1 b_1 s_2 \dots b_{n-1} s_n b_n$$

where $n \in \mathbb{N}$, s_i, $1 \leqslant i \leqslant n$, is a constant symbol of the input string, and b_j, $0 \leqslant j \leqslant n$, is a *boundary constant*. For convenience, we note $w[i] = s_i$. Every boundary constant is associated with an integer referring to its position in the string (boundary constants are ordered by the relation $\leqslant$ on $\mathbb{N}$). Thus $b_0 = 0$, $b_1 = 1$, etc.

In the same way, we define an argument to instantiate, *arg*, as follows:

$$arg = B'_0 s'_1 B'_1 s'_2 \dots B'_{m-1} s'_m B'_m$$

where $m \in \mathbb{N}$, s'_i, $1 \leqslant i \leqslant m$ is a symbol (range variable or constant), and B'_j, $0 \leqslant j \leqslant m$ is a *boundary variable*. We write $\mathbb{V}$ for the set of all boundary variables. As before, we note $arg[i] = s'_i$. Furthermore, each boundary variable is associated with the finite domain $[0..n]$ (i.e., a boundary variable must match a boundary constant defined over the input string).[6]

5 A formal proof of the correctness of this formulation is omitted.

6 Note that here we only consider the case where all constants appearing in the input string and in the argument to instantiate occur only once.

STEP 2 Once our model has been defined, we compute a constraint matrix M_C, which maps boundary variables to boundary constants. Thus M_C is a $(m+1) \times (n+1)$ matrix

$$M_C[i,j] = \begin{cases} 1 & \text{if } arg[i] = w[j] \text{ or } arg[i-1] = w[j-1] \\ & \quad (2 \leqslant i \leqslant m, 2 \leqslant j \leqslant n) \\ 1 & \text{if } (i,j) = (1,1) \qquad (*) \\ 1 & \text{if } (i,j) = (m+1, n+1) \qquad (*) \\ 0 & \text{otherwise} \end{cases}$$

The 1 in M_C represent boundary positions that are constrained by the input string. The lines marked (*) represent the fact the lower and upper bounds of the argument to instantiate must be respectively the lower and upper bounds of the input string. If we consider the previous example of the predicate $A(aXYdZ)$ to be instantiated with *abcdef*, we obtain the following constraint matrix:

$$M_C = \begin{bmatrix} & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \\ B'_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ B'_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ B'_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B'_3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ B'_4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ B'_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that without the assumption made about the uniqueness of constants in the input string, this matrix would not represent all possible constraints on the boundaries.

STEP 3 We finally search for all assignments of values in $[0..n]$ to the boundary variables B'_j, $0 \leqslant j \leqslant m$. In other terms, we search for all functions $f : \mathbb{V} \to [0..n]$ such that $B'j, 0 \leqslant j \leqslant m$ maps to $b_i, 0 \leqslant i \leqslant n$.

This search uses generic constraints reflecting the ordering of the boundary variables:

$$\text{for all } 0 \leqslant i,j \leqslant m : (i \leqslant j) \Rightarrow 0 \leqslant (f(B'_i) \leqslant f(B'_j)) \leqslant n$$

and the specific constraints encoded in the matrix M_C:

$$\text{for all } 1 \leqslant i \leqslant m+1, 1 \leqslant j \leqslant n+1 : (M_C[i,j] = 1) \Rightarrow (f(B'_{i-1}) = b_{j-1})$$

In the latter formula, the indexes of the boundaries are shifted with respect to the matrix indexes (i, j) because M_C's rows and columns indexes start from 1 while the indexes of the boundaries start from 0. Considering our previous example, all B'_i are constrained by M_C, except B'_2, which can take 3 values: 1 (b_1), 2 (b_2) or 3 (b_3), these are the 3 expected range boundaries.

This constraint-based computation of clause instantiations has been implemented within the context of an implementation of the directional top-down parsing algorithm (resp. Boullier's algorithm) in the TuLiPA system (using the `Gecode` library), where in comparison with a first naive implementation of clause instantiation, an enormous speed-up is achieved.

3.2 PARSING SIMPLE RANGE CONCATENATION GRAMMAR

This section is dedicated to the problem of the symbolic parsing of SRCG.

3.2.1 *Introduction*

Simple Range Concatenation Grammar and its equivalent formalisms (p. 41) are commonly considered to be the most expressive formalisms which are still mildly context-sensitive. Several parsing algorithms for them have appeared. See section 3.3 for details on these works and other related literature.

In the following, I present an incremental Earley parsing algorithm which is a modification of the "incremental algorithm" of Burden and Ljunglöf (2005) for LCFRS with a strategy very similar to the strategy adopted by Thread Automata. Then, in section 3.2.3, item filters are presented which limit the search space, as well as an experimental evaluation of the algorithm in section 3.2.4.

3.2.2 *Incremental Earley Parsing*

The Incremental Earley algorithm assumes the grammar to be ordered and ε-free. Recall that Ordered Simple Range Concatenation Grammar (OSRCG) is equivalent to SRCG (lemma 2.55, p. 47). We refrain from supporting non-ε-free grammars since the grammars used for the ex-

perimental implementation are all ε-free. However, note that only minor modifications would be necessary in order to support non-ε-free grammars (see below). Furthermore, without loss of generality, we assume that for every clause, there is a $k \in \mathbb{N}_0$ such that the variables occurring in the clause are exactly $X_1, \ldots, X_k$.

We process the arguments of the left-hand sides (LHSs) of clauses incrementally, starting from an S-clause. Whenever we reach a variable, we move into the clause of the corresponding RHS predicate (PREDICT or RESUME). Whenever we reach the end of an argument, we SUSPEND this clause and move into the parent clause that has called the current one. In addition, we treat the case where we reach the end of the last argument and move into the parent as a special case. Here, we first CONVERT the item into a passive one and then COMPLETE the parent item with this passive item. This allows for some additional factorization.

The item form for passive items is

$$[A, \vec{\rho}]$$

where A a predicate of some arity k, $\vec{\rho}$ is a range vector of arity k. The item form for active items is

$$[A(\vec{\phi}) \to A_1(\vec{\phi}_1) \ldots A_m(\vec{\phi}_m), \mathit{pos}, \langle i, j\rangle, \vec{\rho}]$$

where

1. $A(\vec{\phi}) \to A_1(\vec{\phi}_1) \ldots A_m(\vec{\phi}_m) \in P$;
2. $\mathit{pos} \in \{0, \ldots, n\}$ is the position up to which we have processed the input;
3. $\langle i, j\rangle \in \mathbb{N} \times \mathbb{N}$ marks the position of our dot in the arguments of the predicate A: $\langle i, j\rangle$ indicates that we have processed the arguments up to the jth element of the ith argument;
4. $\vec{\rho}$ is a range vector containing the bindings of the variables and terminals occurring in the left-hand side of the clause ($\vec{\rho}(i)$ is the range the ith element is bound to). We distinguish between different occurrences of the same terminal.

When first predicting a clause, it is initialized with a vector containing only symbols "?" for "unknown". We call such a vector (of appropriate

arity) $\vec{\rho}_{\text{init}}$. We introduce an additional piece of notation. We write $\vec{\rho}(X)$ for the range bound to the variable X in $\vec{\rho}$. Furthermore, we write $\vec{\rho}(\langle i, j\rangle)$ for the range bound to the jth element in the ith argument of the LHS of the clause.

The deduction rules are as follows.

INITIALIZE: We predict the S-predicate.

$$\frac{}{[S(\vec{\phi}) \to \vec{\Phi}, 0, \langle 1, 0\rangle, \vec{\rho}_{\text{init}}]}$$

with the side condition $S(\vec{\phi}) \to \vec{\Phi} \in P$.

SCAN: Whenever the next symbol after the dot is the next terminal in the input, we can scan it.

$$\frac{[A(\vec{\phi}) \to \vec{\Phi}, pos, \langle i, j\rangle, \vec{\rho}]}{[A(\vec{\phi}) \to \vec{\Phi}, pos + 1, \langle i, j + 1\rangle, \vec{\rho'}]}$$

with the side conditions $\vec{\phi}(i, j + 1) = w_{pos+1}$ and $\vec{\rho'}$ is $\vec{\rho}$ updated with $\vec{\rho}(i, j + 1) = \langle pos, pos + 1\rangle$.

In order to support non-ε-free grammars, one would need to store the pair of indices a ε is mapped to in the range vector, along with the mappings of terminals and variables. The indices could be obtained through a SCAN-ε operation, parallel to the SCAN operation.

PREDICT: Whenever our dot is left of a variable that is the first argument of some RHS predicate B, we predict new B-clauses.

$$\frac{[A(\vec{\phi}) \to \dots B(X, \dots) \dots, pos, \langle i, j\rangle, \vec{\rho}_A]}{[B(\vec{\psi}) \to \vec{\Psi}, pos, \langle 1, 0\rangle, \vec{\rho}_{\text{init}}]}$$

with the side condition $\vec{\phi}(i, j + 1) = X, B(\vec{\psi}) \to \vec{\Psi} \in P$.

SUSPEND: Whenever we arrive at the end of an argument that is not the last argument, we suspend the processing of this clause and we go back to the item that was used to predict it.

$$\frac{\begin{array}{c}[B(\vec{\psi}) \to \vec{\Psi}, pos', \langle i, j\rangle, \vec{\rho}_B], \\ [A(\vec{\phi}) \to \dots B(\vec{\eta}) \dots, pos, \langle k, l\rangle, \vec{\rho}_A]\end{array}}{[A(\vec{\phi}) \to \dots B(\vec{\eta}) \dots, pos', \langle k, l + 1\rangle, \vec{\rho}]}$$

with the side conditions that the dot in the antecedent A-item precedes the variable $\vec{\eta}(i)$, $|\vec{\psi}(i)| = j$ (the ith argument has length j and has

therefore been completely processed), $|\vec{\psi}| < i$ (the ith argument is not the last argument of B), $\vec{\rho}_B(\vec{\psi}(i)) = \langle pos, pos' \rangle$ and for all $1 \leqslant m < i$: $\vec{\rho}_B(\vec{\psi}(m)) = \vec{\rho}_A(\vec{\eta}(m))$. $\vec{\rho}$ is $\vec{\rho}_A$ updated with $\vec{\rho}_A(\vec{\eta}(i)) = \langle pos, pos' \rangle$.

CONVERT: Whenever we arrive at the end of the last argument, we convert the item into a passive one.

$$\frac{[B(\vec{\psi}) \to \vec{\Psi}, pos, \langle i, j \rangle, \vec{\rho}_B]}{[B, \vec{\rho}]}$$

with the side conditions $|\vec{\psi}(i)| = j$, $|\vec{\psi}| = i$ and $\vec{\rho}_B(\vec{\psi}) = \vec{\rho}$.

COMPLETE: Whenever we have a passive B item we can use it to move the dot over the variable of the last argument of B in a parent A-clause that was used to predict it.

$$\frac{[B, \vec{\rho}_B], [A(\vec{\phi}) \to \ldots B(\vec{\eta}) \ldots, pos, \langle k, l \rangle, \vec{\rho}_A]}{[A(\vec{\phi}) \to \ldots B(\vec{\eta}) \ldots, pos', \langle k, l+1 \rangle, \vec{\rho}]}$$

where the dot in the antecedent A-item precedes the variable $\vec{\eta}(|\vec{\rho}_B|)$, the last range in $\vec{\rho}_B$ is $\langle pos, pos' \rangle$, and for all $1 \leqslant m < |\vec{\rho}_B|$: $\vec{\rho}_B(m) = \vec{\rho}_A(\vec{\eta}(m))$. $\vec{\rho}$ is $\vec{\rho}_A$ updated with $\vec{\rho}_A(\vec{\eta}(|\vec{\rho}_B|)) = \langle pos, pos' \rangle$.

RESUME: Whenever we are left of a variable that is not the first argument of one of the RHS predicates, we resume the clause of the RHS predicate.

$$\frac{\begin{array}{c}[A(\vec{\phi}) \to \ldots B(\vec{\eta}) \ldots, pos, \langle i, j \rangle, \vec{\rho}_A], \\ [B(\vec{\psi}) \to \vec{\Psi}, pos', \langle k-1, l \rangle, \vec{\rho}_B]\end{array}}{[B(\vec{\psi}) \to \vec{\Psi}, pos, \langle k, 0 \rangle, \vec{\rho}_B]}$$

with the side conditions $\vec{\phi}(i)(j+1) = \vec{\eta}(k), k > 1$ (the next element is a variable that is the kth element in $\vec{\eta}$, i.e., the kth argument of B), $|\vec{\psi}(k-1)| = l$, and $\vec{\rho}_A(\vec{\eta}(m)) = \vec{\rho}_B(\vec{\psi})(m)$ for all $1 \leqslant m \leqslant k-1$.

The goal is again as follows.

$$\text{GOAL} \quad [S, \langle 0, n \rangle]$$

Note that, in contrast to a purely bottom-up CYK algorithm, the Earley algorithm presented here is prefix valid, provided that the grammar does not contain useless symbols.

3.2.3 *Filters*

During parsing, various optimizations known from (P)CFG parsing can be applied. More concretely, because of the particular form of our simple RCGs, we can use several filters to reject items very early that cannot lead to a valid parse tree for a given input string w with $|w| = n$.

For the grammars we extract from treebanks,[7] the following holds. They are ε-free, we know that each variable or occurrence of a terminal in the clause must cover at least one terminal in the input. Furthermore, since separations between arguments are generated only in cases where between two terminals belonging to the yield of a nonterminal, there is at least one other terminals that is not part of the yield, we know that between different arguments of a predicate, there must be at least one terminal in the input. Consequently, we obtain as a filtering condition on the validity of an active item that the length of the remaining input must be greater or equal to the number of variables and terminal occurrences plus the number of argument separations to the right of the dot in the LHS of the clause. More formally, an active item $[A(\vec{\phi}) \rightarrow A_1(\vec{\phi}_1) \dots A_m(\vec{\phi}_m), pos, \langle i, j\rangle, \vec{\rho}]$ satisfies the *length filter* iff

$$(n - pos) \geqslant (|\vec{\phi}(i)| - j) + \Sigma_{k=i+1}^{dim(A)} |\vec{\phi}(k)| + (dim(A) - i)$$

The length filter is applied to results of PREDICT, RESUME, SUSPEND and COMPLETE.

A second filter checks for the presence of required preterminals. This corresponds to the F estimate from Klein and Manning (2003a) (the same idea was in fact already proposed by Langer (1998)). In our case, we assume the preterminals to be treated as terminals, so this filter amounts to checking for the presence of all terminals in the predicted part of a clause (the part to the right of the dot) in the remaining input. Furthermore, we check that the terminals appear in the predicted order and that the distance between two of them is at least the number of variables/terminals and argument separations in between. In other words, an active item

$$[A(\vec{\phi}) \rightarrow A_1(\vec{\phi}_1) \cdots A_m(\vec{\phi}_m), pos, \langle i, j\rangle, \vec{\rho}]$$

7 Treebank grammar extraction is described in section 5.2.2.

satisfies the *terminal filter* iff we can find an injective mapping f_T : $Term = \{\langle k, l\rangle \mid \vec{\phi}(k)(l) \in T$ and either $k > i$ or $(k = i$ and $l > j)\} \rightarrow \{pos + 1, \ldots, n\}$ such that

1. $w_{f_T(\langle k,l\rangle)} = \vec{\phi}(k)(l)$ for all $\langle k, l\rangle \in Term$;
2. for all $\langle k_1, l_1\rangle, \langle k_2, l_2\rangle \in Term$ with $k_1 = k_2$ and $l_1 < l_2$:
$$f_T(\langle k_2, l_2\rangle) \geqslant f_T(\langle k_1, l_1\rangle) + (l_2 - l_1);$$
3. for all $\langle k_1, l_1\rangle, \langle k_2, l_2\rangle \in Term$ with $k_1 < k_2$:
$$f_T(\langle k_2, l_2\rangle) \geqslant f_T(\langle k_1, l_1\rangle) + (|\vec{\phi}(k_1)| - l_1) + \Sigma_{k=k_1+1}^{k_2-1}|\vec{\phi}(k)| + l_2 + (k_2 - k_1).$$

Checking this filter amounts to a linear traversal of the part of the LHS of the clause that is to the right of the dot. We start with index $i = pos + 1$, for every variable or gap we increment i by 1. For every terminal a, we search the next a in the input, starting at position i. If it occurs at position j, then we set $i = j$ and continue our traversal of the remaining parts of the LHS of the clause.

The preterminal filter is applied to results of the PREDICT and RESUME operations.

3.2.4 *Evaluation*

We have implemented the incremental Earley parser with the filtering conditions on items. In order to test it, we have extracted an SRCG from the first 1,000 sentences of the German NeGra and TIGER[8] (with removed punctuation) and parsed the sentences 1,001-1,100 with it. The grammars contained 2,474 clauses (NeGra) and 2,554 clauses (TIGER). Table 4 contains the total number of sentences for different lengths, the number of sentences for which a parse was found, and the average parsing times of the sentences for which a parse was found.

This shows that using the filters, symbolic parsing of SRCG is feasible in practice.

8 The treebanks are presented at length in section 4.2.1.

LENGTH	NEGRA PARSE/TOT/AV.T.	TIGER PARSE/TOT/AV.T.
$\|w\| \leqslant 20$	73/84/0.40 sec.	50/79/0.32
$20 \leqslant \|w\| \leqslant 35$	14/16/2.14 sec.	10/19/2.16

Table 4: Evaluation of the SRCG Parser

3.3 RELATED WORK

In this section, I present relevant related work for both RCG and SRCG.

3.3.1 *Range Concatenation Grammar*

A directional top-down recognizer has been presented by Pierre Boullier, using pseudocode (Boullier, 1998, 2000b). It is roughly equivalent to the directional top-down parsing algorithm. Boullier's RCG parser implementation, SYNTAX (Boullier and Deschamp, 1988), is (to my knowledge) the only implementation of a parser for arbitrary RCGs apart from the TuLiPA system (Kallmeyer et al., 2008b). In SYNTAX, a highly optimized and very fast system, (unpublished) heuristics on the input strings are used to avoid having to compute complete instantiations at once. The SYNTAX approach thereby seems to be conceptually comparable to the formulation using range constraint vectors presented in the previous sections and implemented in the TuLiPA system (Pierre Boullier and Benoît Sagot, personal communication).

Barthélemy et al. (2001) present an extension of the top-down algorithm. Roughly, their approach is to generate a 1-RCG from the grammar, by introducing a separate predicate for each argument of a clause. For example, from a clause $S(XYZ) \rightarrow A(X, Y, Z)$, the clause $S^1(XYZ) \rightarrow A^1(X)A^2(Y)A^3(Z)$ would be generated, the superscripts representing the respective argument of the original predicate. The 1-RCG then acts as an oracle which limits the search space by excluding early partial results which do not lead to a parse.

I am not aware of any further work on parsing RCG. However, RCG has found some applications. Some of them have already been mentioned in section 3.1.1. Another one has been presented by Boullier and Sagot (2009). They introduce an extension of Boullier's top-down algorithm which is able to handle Directed Acyclic Graphs (DAGs) as

input (instead of a simple sequence of words). Handling DAGs is relevant since they are used in several NLP applications, such as in speech recognition. While the complexity of parsing DAGs was known before (polynomial when using SRCG and NP-complete when using arbitrary RCG, cf. Bertsch and Nederhof (2001)), Boullier and Sagot's work is the first concerning implementation-related issues. Their parser produces parse forests for input DAGs using both SRCG and arbitrary RCG in PTIME. However, the parse forests for arbitrary RCG need to be filtered in order to be correct. The filtering algorithm takes exponential time (which reflects again Bertsch and Nederhof's result).

Søgaard (2011) argues that a variant of RCG, $(2,2)$-BRCG, is useful for Syntax-Directed Machine Translation. A $(2,2)$-BRCG is a bottom-up non-erasing 2-RCG, where an argument of the LHS predicate contains at most two variables. Such grammars can be used for the bilingual modeling of sentence pairs such at is offered by Inversion Transduction Grammar (ITG) (Wu, 1997) and Synchronous CFG used by Chiang (2007) (he calls them Syntax-Directed Transduction Grammars (SDTG)). In $(2,2)$-BRCG, intuitively speaking, the first argument of an LHS predicate models the source language and the second argument models the target language. The copying capacity of arbitrary RCG is used. Søgaard shows that the formalism is more expressive than SDTG and shows the relevance of his result with an empirical investigation of aligned corpora that shows that alignments which can be modeled by RCG, but cannot be modeled by SDTG, occur relatively frequently.

3.3.2 *Parsing SRCG and Equivalent Formalisms*

Satta (1992) investigates the formal properties of LCFRS and its recognition problem. While it had been known before that the fixed recognition problem is solvable in PTIME (Vijay-Shanker et al., 1987), Satta shows among other things that the universal recognition problem is NP-complete. Burden and Ljunglöf (2005) present several parsing algorithms for LCFRS, formulated as deduction systems, together with suggestions for optimizations. Their algorithms are used in Grammatical Framework (Ranta, 2004) and have been the starting point for the SRCG parsing algorithm presented in the previous section. The formal properties and the complexity of Grammatical Framework are explored in Ljunglöf (2004). Ljunglöf also establishes the relation be-

tween Grammatical Framework (GF) and Parallel Mulitple Context-Free Grammar (PMCFG) (Seki et al., 1991), a variant of MCFG which has recently found further attention with respect to parsing (Angelov, 2009).

Seki et al. (1991) introduce the formalism of MCFG. They present several important computational properties of Multiple Context-Free Languages (MCFLs), such as the relation to the languages of other formalisms such as TAGs. Furthermore, their article contains a CYK recognizer for their formalism, formulated as pseudo-code. Kanazawa (2009a) presents a prefix-valid Earley parser for MCFG, based on the idea of representing an MCFG by a Datalog program and applying generalized supplementary magic-sets rewriting (Beeri and Ramakrishnan, 1991).

Another approach to parsing MCFG is presented by Nakanishi et al. (1997). They reduce the parsing problem to boolean matrix multiplication, something which has been done before for CFG (Valiant, 1975) and TAG (Satta, 1994). The complexity of the algorithm directly depends on the complexity of the algorithm used for boolean matrix multiplication. However, since the algorithms for this task which have a low complexity are impractical (Lee, 2002), Nakanishi et al.'s work is of rather theoretical interest.

Work on Minimalist Grammar (MG) parsing has been presented by Michaelis (2001b), who investigates the formal properties of MG, and Harkema (2001), who delivers bottom-up, top-down and Earley strategies for MG.

Villemonte de la Clergerie (2002) presents an automaton model, called Thread Automata (TA). TA is not equivalent to a particular formalism, it can be parametrized to work as an automaton model for different mildly context-sensitive formalisms. Villemonte de la Clergerie's parametrization for SRCG, as already mentioned, delivers an incremental parser with a strategy similar to the parser presented before.

For SRCG and the equivalent formalisms, probabilistic variants have been created. The first probabilistic application can be found in Kato et al. (2006). The authors define Probabilistic MCFG (PMCFG)[9] and use a restricted version of it for the modeling of RNA pseudoknotted structures. Pseudoknot is a typical substructure in secondary struc-

9 See remark on acronym on p. 230.

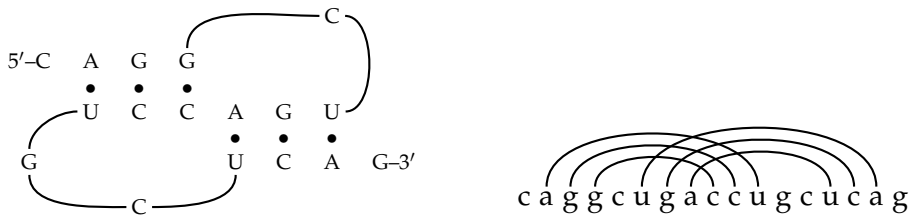


Figure 17: RNA pseudoknotted structure

tures of several RNAs. It poses a particular difficulty for *RNA structure prediction*, the task of determining nucleic acid secondary or tertiary structure from its sequence.[10] The reason for the difficulty is that, intuitively speaking, long-distance resp. crossing dependencies are involved. Figure 17 shows such a structure together (left) with an alternative representation called *arc depiction* (right). The example is taken from Kato et al. (2006).

A further application of probabilistic SRCG is data-driven parsing. This is what the remainder of this thesis is dedicated to.

3.4 CONCLUSION

This chapter was concerned with the problem of symbolic parsing of Range Concatenation Grammar and Simple Range Concatenation Grammar. Several algorithms have been presented which pursue different strategies.

A top-down algorithm, two CYK-style algorithms, and an Earley-style parsing algorithm for the full class of RCG have been presented, formulated in terms of deduction systems. While the directional top-down algorithm corresponds the algorithm of Boullier, all other algorithms are novel. The crucial difference between the directional top-down algorithm and the Earley-style algorithm is that while the former computes all clause instantiations during PREDICT operations, the latter avoids this using a technique of dynamic updating of a set of constraints on range boundaries. Experiments show that the Earley-style algorithm generates significantly less items than the directional top-down algorithm, which confirms that range boundary constraint propagation is a viable method for a lazy computation of ranges.

10 See Gardner and Giegerich (2004) for a survey of work on RNA structure prediction.

Furthermore, a Earley-style algorithm for SRCG has been presented, also formulated as deduction system. Filters on the chart have been presented which reduce the number of items produced during parsing. An implementation and experiments with grammars extracted from treebanks showed that reasonable parsing times can be achieved. Unfortunately, experiments with the Earley algorithm have shown that with grammars of a reasonable size for data-driven parsing (more than 15,000 clauses), exhaustive parsing is no longer efficient, due to the high ambiguity of treebank grammars. Algorithms using only passive items (pure CYK) are more suitable in this context since they do not "hard-code" a binarization strategy and therefore more easily allow to reduce the number of produced items through markovization. This subject will be treated in the following chapters.

With respect to parsing arbitrary RCG, the experience from implementing the algorithms presented in this chapter shows that in practice, most of the computational resources are taken up by the process of finding instantiations. Formulating the instantiation problem in an appropriate way is the key to efficient parsing; however, what is the best strategy seems to depend on the properties of the grammar which is used. In other words, it is very hard to build a parser for completely unrestricted RCG, i. e., for all possible grammar instances of the formalism. Nevertheless, by letting the parser know in advance certain details of the words in the language of the grammar, one can achieve excellent parsing performance on certain practically relevant Range Concatenation Languages. In our case, e. g., this information could be given to the parser in the form of constraints. This is also the approach taken in other work on parsing RCG such as Boullier (2000a), where very low "practical" parsing complexities are reported that cannot be achieved in the general case.

4

DATA-DRIVEN PARSING USING CONTEXT-FREE GRAMMAR

Linguistic grammars can not only be created manually by linguists. Another way to obtain grammars is to interpret the syntactic structures contained in a treebank as the derivations of a latent grammar and to use an appropriate algorithm for grammar extraction. One can not only obtain the grammar, but also estimate occurrence probabilities of its rules. These can be used for disambiguation, i. e., to determine the best parse, resp. parses of a sentence; furthermore, they can be used for speeding up the parsing process. Parsing with a probabilistic grammar obtained from a treebank is called *data-driven parsing*.

Data-driven constituency parsing looks back on a long history. In this field, many developments have been made which are relevant for the work presented in the following chapters of this thesis. Therefore, after a formal introduction to probabilistic parsing and data-driven parsing in section 4.1, I present an overview of major works in section 4.2. Since much of the development in constituency parsing is connected to the development of treebanks and treebank annotation schemes, a part of the section is dedicated to constituency treebanks. Correspondingly, in section 4.3 I present an overview of developments in data-driven dependency parsing. The same section introduces dependency annotation. Finally, in section 4.4, out of the preceding two sections, I motivate the main contributions of this thesis, namely the development of a parser for data-driven constituency parsing and grammar-based non-projective dependency parsing with Simple Range Concatenation Grammar (SRCG).

4.1 PROBABILISTIC PARSING AND DATA-DRIVEN PARSING

Probabilistic data-driven parsing is strongly associated with a single formalism, namely Probabilistic Context Free Grammar.

4.1.1 *Probabilistic Generative Syntax*

A Probabilistic Context-Free Grammar is a Context-Free Grammar augmented with rule probabilities. In the form they are presented in here, they have appeared first in Booth and Thomson (1973), albeit with a different notation.

DEFINITION 4.1 (Probabilistic Context-Free Grammar). A *Probabilistic Context-Free Grammar (PCFG)*[1] is a tuple $G = (N, T, P, S, p)$ where (N, T, P, S) is a Context-Free Grammar (CFG) and $p : P \to [0, 1]$[2] is a function such that for all $A \in N$,

$$\sum_{A \to \alpha \in P} p(A \to \alpha) = 1$$

$p(A \to \alpha)$ denotes the conditional probability $p(A \to \alpha \mid A)$. Since we are interested in probabilities of derivations, we must make sure that equivalent derivations are only counted once. This is achieved by always referring to the *leftmost derivation*, in which the leftmost non-terminal is always substituted first. To this end, in definition 4.2, we revise 1. of definition 2.21, p. 27.

DEFINITION 4.2 (Leftmost derivation (CFG)). Let $G = (N, T, P, S)$ be a CFG. The relation $\Longrightarrow_G \subseteq (N \cup T)^+ \times (N \cup T)^*$ called *leftmost derives* is defined as follows. $\gamma \Longrightarrow_G \gamma'$ iff there are $A \to \beta \in P$, $\alpha \in T^*$ and $\alpha' \in (N \cup T)^*$ such that $\gamma = \alpha A \alpha'$ and $\gamma' = \alpha \beta \alpha'$.

It is easy to see that *leftmost derives* is properly contained in *derives*.

PROPOSITION 4.3. Let $G = (V, T, P, S)$ be a CFG, let $w \in T^*$. If $w \in \mathcal{L}(G)$, then there exists at least one derivation tree $\mathcal{G}$ for w, and with respect to a certain derivation tree, w has a unique leftmost derivation (Hopcroft and Ullman, 1979).

DEFINITION 4.4 (Derivation probability (PCFG)). Let $G = (N, T, P, S, p)$ be a PCFG, and let $\alpha, \gamma \in (N \cup T)^*$.

1. Let $A \to \beta \in P$. The probability of a derivation step $\alpha \Longrightarrow_G^{A \to \beta} \gamma$ is defined as follows.

$$p(\alpha \Longrightarrow_G^{A \to \beta} \gamma) = p(A \to \beta)$$

1 Sometimes, especially in literature not about computational linguistics, PCFG is called Stochastic Context-Free Grammar (SCFG).

2 As usual, $[0, 1]$ denotes the interval $\{i \in \mathbb{R} \mid 0 \leqslant i \leqslant 1\}$.

2. Let $A_1 \to \beta_1, \ldots, A_m \to \beta_m \in P$, $m \in \mathbb{N}$. The probability of a derivation $\alpha \Longrightarrow_G^{A_1 \to \beta_1} \cdots \Longrightarrow_G^{A_m \to \beta_m} \gamma$ is defined as follows.

$$p(\alpha \Longrightarrow_G^{A_1 \to \beta_1} \cdots \Longrightarrow_G^{A_m \to \beta_m} \gamma) = \prod_{i=1}^{m} p(A_i \to \beta_i)$$

3. The probability of $\alpha \overset{*}{\Longrightarrow}_G \gamma$ is defined as the sum over the probabilities of all leftmost derivations of γ from α:

$$p(\alpha \overset{*}{\Longrightarrow}_G \gamma) = \sum_{i=0}^{k} \prod_{j=0}^{m_i} p(A_j^i \to \beta_j^i)$$

   where $k \in \mathbb{N}$ is the number of leftmost derivations of γ from α and $m_i \in \mathbb{N}$ is the derivation length of the ith derivation and $A_j^i \to \beta_j^i$ is the production used in the jth derivation step of the ith leftmost derivation.

DEFINITION 4.5 (Inside and Outside probabilities (PCFG)). Let $G = (N, T, P, S, p)$ be a PCFG and let $w = w_1 \cdots w_n$, $n \in \mathbb{N}$, $w_i \in \Sigma$ for some alphabet Σ, $1 \leqslant i \leqslant n$, be an input string. Let $1 \leqslant i \leqslant j \leqslant n$.

1. The probability of deriving a part $w_i \cdots w_j$ of the input string from a certain non-terminal $A \in N$ is called *inside probability* of A, i, j and defined as

$$p(A \overset{*}{\Longrightarrow}_G w_i \cdots w_j)$$

2. The probability of a starting with S and generating non-terminal $A \in N$ and all terminals outside of w_i to w_j is called *outside probability* of A, i, j and defined as

$$p(S \overset{*}{\Longrightarrow}_G w_1 \cdots w_{i-1} A w_{j+1} \cdots w_n)$$

An illustration for outside and inside probabilities is in figure 18. The part surrounded by the dashed line refers to the outside probability, and the part surrounded by the solid line refers to the inside probability. The *inside algorithm* and the *outside algorithm* can be used to compute the resp. probabilities for a given grammar and input string (Baker, 1979).

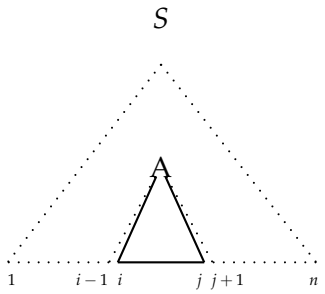


Figure 18: Inside and outside probabilities

4.1.2 *Finding the Most Probable Derivation*

A computational problem of predominant interest is how to find the most probable derivation for a given input string. One of the oldest approaches for efficiently searching the most probable derivation of a PCFG for a given string is the *Viterbi algorithm* (Viterbi, 1967).[3] The basic idea behind the algorithm is to keep track for all subderivations of which one has the highest probability. This can be achieved with a CYK-like algorithm (Goodman, 1998, p. 7), in which at all times, each chart cell contains the highest probability of all corresponding sub-derivations found so far. With dynamic programming, the algorithm runs in $\mathcal{O}(n^3)$ (Manning and Schütze, 1999, p. 396). The Viterbi algorithm is exhaustive, i. e., all possible sub-derivations are built.

An alternative to the Viterbi algorithm is *Weighted Deductive Parsing (WDP)* (Nederhof, 2003). Nederhof uses *weighted deduction systems* together with Knuth's algorithm (Knuth, 1977) for finding the derivation with the lowest weight. As weights, one commonly uses the absolute value of the natural logarithm of the corresponding probabilities. A production $A \to \alpha$ with $p(A \to \alpha) = y$ is notated as $y : A \to \alpha$; an item A with weight x is notated as $x : A$. Here and throughout, we will write $log(y)$ to denote $|ln(y)|$.

A weighted deduction system as presented by Nederhof (2003) is a finite set of inference rules extended with weights. The inference rules have the form

3 Some authors call any algorithm which computes the best derivation of a PCFG *Viterbi algorithm*.

$$\frac{x_1 : A_1, \ldots, x_k : A_k}{f(x_1, \ldots, x_k) : B} \; c_1, \ldots, c_m$$

where again, $k, m \in \mathbb{N}_0$, the *antecedents* $A_1, \ldots, A_k$ and the *consequent* B are items, and $c_1, \ldots, c_m$ are side conditions which link the inference rule to the grammar and the input string. $x_1, \ldots, x_k$ are unique weight variables. f is the *weight function*.

The actual algorithm is shown as algorithm 2. It uses two data structures, a chart $\mathcal{C}$ and an agenda $\mathcal{A}$. The agenda is organized as a *priority queue* of items, ordered by item weights. In each step, the item $x : I$ with the lowest weight is removed from the agenda and put in the chart. Next, all possible items are deduced from $x : I$ using items from the chart. The resulting items are added to the agenda. In case the agenda already contains such an item, the corresponding weight is updated to the maximum of both. If the weight functions $f(x_1, \ldots, x_m)$ in the deduction system are *superior* (in Knuth's terms), which means that they are monotone non-decreasing in each variable, and if $f(x_1, \ldots, x_m) \geqslant max(x_1, \ldots, x_m)$ for all possible values of $x_1, \ldots x_m$, then it is guaranteed that when an item is removed from the agenda, no competing analysis with a higher probability can be found (Nederhof, 2003). That means that when the goal item is removed from the agenda for the first time, it is guaranteed to be the most probable analysis of the input string.

The CYK algorithm for PCFG can be formulated using a weighted deduction system as shown in figure 19. in_A, in_B and in_C denote the log probabilities of the best inside parses of the respective items, i. e., their *Viterbi inside scores*. It is easy to see that for all possible values of in_B, in_C and p, it holds that $f(in_B, in_C, log(p)) \geqslant max(in_B, in_C, log(p))$, i. e., the superiority condition is fulfilled.

By estimating the Viterbi outside score[4] of items, one can get an idea of which items lead to a complete parse more quickly than others. In *A* parsing* (Klein and Manning, 2003a) one adds an estimate of the Viterbi outside score of an item to its inside score, in order to favorably influence the ordering of the priority agenda. If the estimate fulfills two conditions, then we are guaranteed to find the best parse.

4 Note that just as Klein and Manning (2003a), we always use *inside score* and *outside score* to denote the Viterbi inside and outside scores, i. e., the resp. maximum probabilities. They are not to be confused with the actual inside and outside probabilities (def. 4.5, p. 93).

Let G be a probabilistic grammar, and let w be the input string
$\mathcal{C} = \emptyset$
$\mathcal{A} = \emptyset$
add items resulting from deduction rules with no antecedent to $\mathcal{A}$
while $\mathcal{A} \neq \emptyset$ **do**
  remove best item $x : I$ from $\mathcal{A}$
  add $x : I$ to $\mathcal{C}$
  if I goal item **then**
    build parse tree from I and exit
  else
    for all $y : I'$ deduced from $x : I$ and items in $\mathcal{C}$ **do**
      if there is no z with $z : I' \in \mathcal{C} \cup \mathcal{A}$ **then**
        add $y : I'$ to $\mathcal{A}$
      else
        if $z : I' \in \mathcal{A}$ for some z **then**
          update weight of I' in $\mathcal{A}$ to $max(y, z)$
          if $y > z$ **then**
            update backpointers to the antecedents of I'
          end if
        end if
      end if
    end for
  end if
end while

Algorithm 2: Weighted deductive parsing

$$\textsc{scan} \quad \frac{}{log(p) : [A, i-1, i]} \; p : A \to w_i \in P$$

$$\textsc{complete} \quad \frac{in_B : [B, i, j], in_C : [C, j, k]}{in_B + in_C + log(p) : [A, i, k]} \; p : A \to BC \in P$$

$$\textsc{goal} \quad x : [S, 0, n]$$

Figure 19: Weighted deduction system for CYK

The first condition is *admissibility*: The actual outside score of an item must never be underestimated. The second condition is *monotonicity*: When deducing an item I_2 from an item I_1, the weight of I_2 must not be greater than the weight of I_1. In order to become A^*-enabled, the COMPLETE rule of the deduction system in figure 19 must be modified as follows (out_A, out_B and out_C are the respective outside estimates):

$$\frac{in_B + out_B : [B, i, j], in_C + out_C : [C, j, k]}{in_B + in_C + log(p) + out_A : [A, i, k]} \; p : A \to BC \in P$$

Klein and Manning build *context summary estimates* and *grammar projection estimates*. Intuitively, a context summary is a generalization of the outside part of the constituent represented by the item. For instance, given an input string of length n, an item $[A, i, j]$ can be summarized with $(A, i, n-j)$, i. e., by the length of the spans to the left and to the right of the span represented by the item.[5] The more informative a context summary item is, the closer its estimate is to the actual outside score. However, more informative estimates are also more expensive to compute. The idea is therefore to choose the summary general enough to be computable, but specific enough to provide enough information for discriminating items. Note that the least informative summary is the one which provides no information at all (which summarizes all items), and the most informative summary is the actual outside context (which summarizes only a single item). A grammar projection estimate projects the actual grammar to a simpler grammar. One parses then first with the simpler grammar, and uses the output as a guide for the actual parser.

One has to keep in mind that managing the agenda as priority queue implies an administrative overhead in an implementation. If one implements the priority queue as a binary heap, one faces a cost of $\mathcal{O}(log(n))$ for a queue of length n for every operation (access, deletion, addition) (Cormen et al., 2003, p. 164). When using a Fibonacci heap (Fredman and Tarjan, 1987), one can get better performance, e. g., updating the weight of an item can be done in constant amortized

5 This is the *SX* estimate of Klein and Manning.

time[6] (Cormen et al., 2003, pp. 505). To this respect, see also section 6.3.2.

WDP is closely related to parsing with *hypergraphs* (Klein and Manning, 2001). Instead of filling a chart as in deductive parsing, Klein and Manning build a hypergraph[7] which serves the same purpose. In WDP, Knuth's algorithm (a generalization of Dijkstra's algorithm) is used to find the best parse using the CFG which encodes the parse forest, the CFG being given by the chart. Klein and Manning use another extension of Dijkstra's algorithm to achieve the same thing on their hypergraph. Thereby, Knuth's superiority condition (see above) is replaced by an equivalent condition on the hypergraphs. See Klein and Manning (2001) for details, particularly sections 2.3 and 2.5. In this thesis, the deduction-based approach is given preference over the hypergraph approach due to the easier accessibility of deduction systems. However, a formulation in terms of hypergraphs would be possible, too.

4.1.3 *Obtaining a Probability Model*

Another important computational problem with regard to PCFG is the estimation of production probabilities given a collection of example derivations. Given a treebank all trees of which are CFG derivation trees (def. 2.23, p. 28), a CFG G which derives at least all sentences can be obtained by interpreting all subtrees of height one as a production of the grammar and collecting all productions in a grammar $G = (N, T, P, S)$. The simplest method for obtaining probabilities (for *training* the grammar) is to use a Maximum Likelihood Estimator. Let

$$\Upsilon = \{(1, \upsilon_1, s_1), \ldots, (n, \upsilon_n, s_n)\},$$

$n \in \mathbb{N}$, be a treebank (def. 2.19, p. 25). For all $A \to \alpha \in P$, the estimated probability $p(A \to \alpha)$ is

$$p(A \to \alpha) = \frac{\sum_{i=1}^{n} f(A \to \alpha, \upsilon_i)}{\sum_{\gamma \text{ s.t. } A \to \gamma \in P} \sum_{i=1}^{n} f(A \to \gamma, \upsilon_i)}$$

6 In an algorithm, the higher cost of an expensive operation might be outweighed by a higher number of times a cheap operation is called. Amortized analysis takes this into account by judging the run-time of an algorithm by looking at the interplay of all of its operations.

7 A directed hypergraph is a tuple (V, H) where V is a set of vertices and $H \subseteq \mathfrak{P}(V) \times \mathfrak{P}(V)$. For an introduction to hypergraphs, see Gallo et al. (1993).

where f is the function which yields the occurrence count of a production in a treebank tree. The counts necessary for the estimation can easily be obtained from the treebank.

In order to be *tight*, the probability model defined by a PCFG must be such that the sum of probabilities of all finite parse tree generated by G is 1 (Booth and Thomson, 1973). This is not always the case, since probability mass can get lost to infinite derivations. It is easy to see that this happens with the PCFG with the productions $0.8 : S \rightarrow SS$ and $0.2 : S \rightarrow a$. Fortunately, Chi and Geman (1998) showed[8] that the estimation of production probabilities from data guarantees tightness.

4.2 DATA-DRIVEN CONSTITUENCY PARSING

In this section, I introduce some relevant aspects of constituency treebank annotation. Furthermore, I present an overview of work on data-driven constituency parsing.

4.2.1 *Constituency Treebank Annotation*

As mentioned in the previous section, for PCFG extraction, we need a treebank which provides CFG parse trees. CFG parse trees cannot encode non-local dependencies because they only allow for constituents with a continuous yield; therefore, a common approach for annotation encoding is to use an annotation backbone based on CFG, enriched with an additional mechanism that encodes information beyond CFG. For PCFG parsing, this additional mechanism is then simply discarded and the pure context-free derivation backbone is used. Let us consider some examples.

The 24 sections of Wall-Street Journal part of the Penn Treebank (PTB) (Marcus et al., 1994) have been the data-source for most new developments in data-driven parsing. Figure 20 shows an example from the annotation guidelines of the PTB (Bies et al., 1995), namely the annotation of (8).

(8) But our outlook has been, and continues to be, defensive.

8 Nederhof and Satta (2006) present a more general proof which, unlike Chi and Geman's proof, also works for grammars with empty rules and unary rules.

```
( (S But
      (NP-SBJ-2 our outlook)
      (VP (VP has
              (VP been
                  (ADJP *RNR*-1)))
          ,
          and
          (VP continues
              (S (NP-SBJ *-2)
                 (VP to
                     (VP be
                         (ADJP *RNR*-1)))))
          ,
          (ADJP-1 defensive))) .)
```

Figure 20: Penn Treebank annotation example

The bracketed structure provides an annotation backbone which is based on CFG. Empty nodes together with labeling conventions establish additional relations in the tree. In the example, this is the case for the annotation of right node raising, where the relation between the raised constituent and its original sites is established by the label `*RNR*` and the coindexation (`-1`). For data-driven parsing, the empty nodes and the special labelings are generally discarded. PCFGs are then extracted from the remaining trees.

A comparable approach is taken in the German Tübingen Treebank of Written German (TüBa-D/Z) (Telljohann et al., 2006). Its annotation backbone is based on CFG. Punctuation and inserted phrases are not included in the annotation. Instead, they are attached to a "virtual" root node, together with the actual root node of the sentence. As an additional mechanism for long-distance dependencies, a combination of topological field annotation (Höhle, 1986) and edge labels is used. As an example, consider the annotation of (9), a case of fronting, in figure 21. The edge label `V-MOD` marks the NP *Ohne Bomben* as modifier of the verb.

(9) Ohne Bomben wären die Vertreibungen auch weitergegangen
Without bombs would have the expulsions also continued

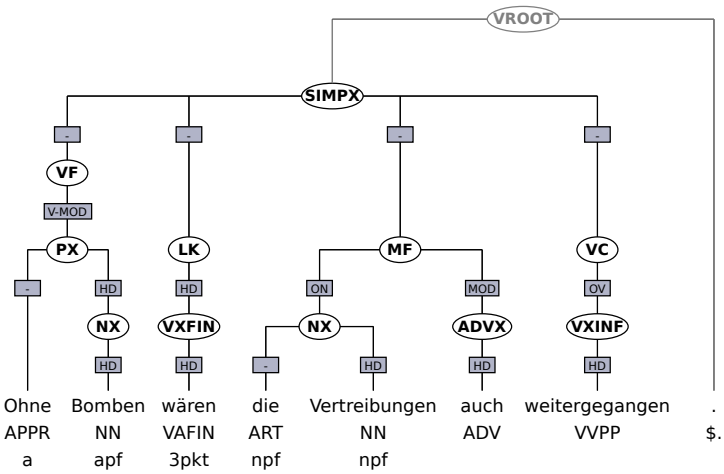


Figure 21: TüBa-D/Z annotation example

> "Without the bombs, the expulsions would have continued as well."

For data-driven PCFG parsing, (at least) all those elements directly below the virtual root node which cause crossing branches, namely punctuation and certain insertions, have to be attached to the actual tree. Since the topological field level and the edge labels do not interfere with context-freeness, there is no need to modify them. A PCFG can be extracted from the resulting trees.

As for the German NeGra (Skut et al., 1997) and TIGER (Brants et al., 2002) treebanks,[9] the situation is different. Crossing branches are allowed, in other words, the annotation backbone based on CFG is given up. This way, long-distance dependencies can be annotated directly by grouping all parts of a discontinuous constituent under a single node. As in TüBa-D/Z, punctuation and certain constituents are not attached to the trees but to a virtual root node, together with the root node of the sentence. As an example, figure 22 shows the annotation of (10). It is also a case of fronting; however, other than in the TüBa-D/Z example, the fronted constituent is attached directly to the VP.

9 The difference between both treebanks is that TIGER has a refined POS tag set and that it is about double the size of NeGra.

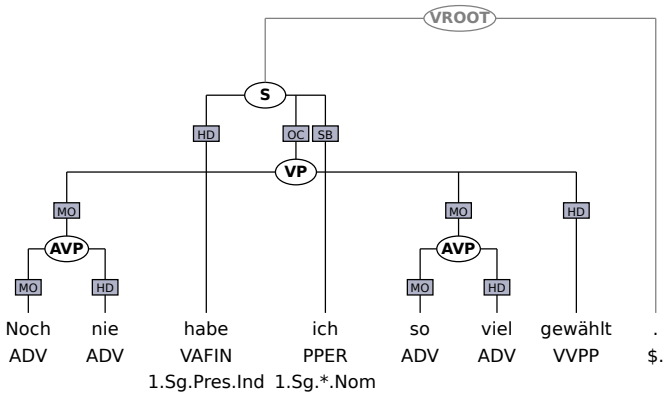


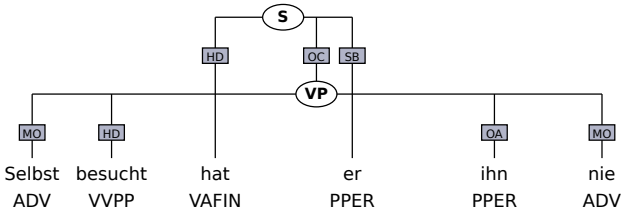
Figure 22: NeGra annotation example

(10) Noch nie habe ich so viel gewählt
Yet never have I so much elected
"Never have I made so many choices."

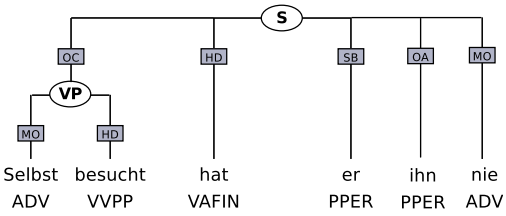
For data-driven parsing using PCFG, the elements directly dominated by the virtual root node must be attached to the actual tree. Furthermore, the crossing branches must be resolved somehow. For this purpose, most of the works using TIGER or NeGra for parsing (Kübler and Penn, 2008, among others) make use of an algorithm which proceeds bottom-up, moving certain constituents in the tree to higher positions, in order to remove the crossing branches. Head words (identified by their edge labels) are excluded from movement, and as few other material as possible is moved. An undocumented implementation of this algorithm by Thorsten Brants is shipped with the NeGra release. Boyd (2007) proposes another algorithm which, for nonterminals with a block degree greater than one, injects additional nodes in the tree that capture this nodes' yield blocks. Figure 23 shows the annotation of (11), another fronting example from NeGra, without and with resolved crossing branches, using the Brants method and the Boyd method.

(11) Selbst besucht hat er ihr nie
Self visited has he him never
"He has never visited him personally."

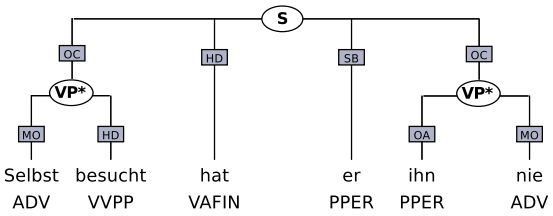
See section 7.2 for a more detailed description of the Brants algorithm.



Unresolved crossing branches



Brants algorithm



Boyd algorithm

Figure 23: Removal of crossing branches

Both NeGra and TIGER (and also TüBa-D/Z) are distributed in the `export` format (Brants, 1997). In this format, a sentence w annotated with a constituency structure that has m non-terminals, $m \in \mathbb{N}_0$, is represented by a list of $|w| + m + 2$ lines. The first line is a begin-of-sentence marker with the sentence number and metadata information about the annotator of the sentence. The following $|w|$ lines, implicitly numbered from 1 to $|w|$, represent the terminals of w. Each of the following m lines, carrying numbers greater than 500, represents a non-terminal node of the constituency structure, except the last line, which is an end-of-sentence marker. The node numbered 0 is the implicit *virtual* root node. Every node line contains tab-separated fields. In version 3 of the `export` format, the first field contains the terminal token, resp. the non-terminal node number, the second field contains the Part-of-Speech (POS) tag, resp. the non-terminal syntactic node label, the third field contains morphological information, the fourth field contains the label of the edge from the node to its parent, the fifth field contains the node number of the parent node, the sixth field can contain the number of an arbitrary node in the tree to which the current one has a so-called *second edge*, the seventh field can contain comments on the node. In version 4 an additional field between the fields one and two contains lemmatization information. Since starting from each node, only the resp. parent node is referenced, child ordering is not specified explicitly. This means that trees with crossing branches can be represented as well. In the graphical representation, as defined in definition 2.11, p. 21, non-terminals are ordered by the leftmost leaf they dominate, i. e., by the smallest element of their respective yields. Figure 24 shows the annotation of (10) in the `export` format, version 3.

4.2.2 *Context-Free Data-Driven Constituency Parsing*

In the following, I present an overview of work on data-driven parsing with the aim of producing context-free derivations.

Evaluating Parser Performance

In order to judge the performance of a parser, one must be able to assess the quality of its output (the parsed *test data*) with respect to the desired output (the *gold data*). The most widely used technique

```
#BOS 90 3 867232142 2
Noch                    ADV     --              MO      500
nie                     ADV     --              HD      500
habe                    VAFIN   1.Sg.Pres.Ind   HD      503
ich                     PPER    1.Sg.*.Nom      SB      503
so                      ADV     --              MO      501
viel                    ADV     --              HD      501
gewählt                 VVPP    --              HD      502
.                       $.      --              --      0
#500                    AVP     --              MO      502
#501                    AVP     --              MO      502
#502                    VP      --              OC      503
#503                    S       --              --      0
#EOS 90
```

Figure 24: `export` format example

for this task consists of comparing for each parsed sentence the set of bracketings produced by the parser with the set of gold bracketings from the manual treebank annotation. A bracketing is thereby a pair of indices on the input string denoting the start and the end of the span dominated by a certain non-terminal (i. e., its yield block). The bracketing is called *labeled* if the label is included; if it is just the index pair, it is called *unlabeled*.

Commonly, bracket scoring is defined as follows (Black et al., 1991; Lin, 1995; Collins, 1997). Let $\mathcal{O}$ be the set of bracketings from the parser output, and let the set of bracketings from the treebank annotation be $\mathcal{G}$. *Precision* is then computed as $\frac{|\mathcal{O} \cap \mathcal{G}|}{|\mathcal{O}|}$, *recall* as $\frac{|\mathcal{O} \cap \mathcal{G}|}{|\mathcal{G}|}$, and F_1 as $\frac{2 * precision * recall}{precision + recall}$. A literal interpretation of this formulation can have an undesired effect, namely, that unary edges are counted only once. This holds especially for the unlabeled case. For instance, if the unlabeled edge $[0, 2]$ occurs $\geqslant 1$ times in the gold data, any number of $[0, 2]$s $\geqslant 1$ in the parser output would lead to the same score. *Evalb* (Sekine and Collins, 1997), the quasi-standard software for bracket scoring, solves this problem by computing the intersection in the formulae of precision and recall such that a bracket is marked as used after it matches for the first time. Black et al. (1991), however, mention the convention of excluding unary constituents. Another difference between Black et al.'s work and Evalb is how statistics are computed over a

set of more than one sentence. While the former uses *micro-averaging* (the aggregate score is computed by averaging the scores of single sentences), the latter uses *macro-averaging* (all counts are summed before the division).[10]

Evalb suffers from certain biases (Sampson and Babarczy, 2003; Rehbein and van Genabith, 2007a; Emms, 2008; Kübler et al., 2008). There is no grade of severeness of errors. From a linguist's point of view, some errors are worse than others, however, spans are either correct or wrong. This makes it impossible to honor a partially correct recognized span which, e. g., has just one boundary right, or which has the boundaries right, but not the label. Last, attachment errors are punished too severely, since a wrong attachment position can cause many (otherwise correct spans) to be disrupted. Other metrics do not suffer from those problems and provide a more balanced evaluation.

- The *leaf-ancestor metric (LA)* (Sampson and Babarczy, 2003) extracts for each tree the paths between the root and leafs in both test and gold data and computes their Levenshtein edit distances (Levenshtein, 1966).

- In *dependency evaluation* (Lin, 1995) the constituency structures in both test and gold data are transformed into dependency structures according to the lexical heads in the trees. The dependency structures are then evaluated against each other.

- The *tree-distance metric (TDIST)* (Emms, 2008) uses a tree edit distance measure (Zhang and Shasha, 1989) between the trees from the parser output and gold trees to determine tree similarity. TDIST will be used later, it will be explained in more detail in section 7.1.

Unlexicalized and Lexicalized Parsing

Charniak (1996) investigates the performance of a PCFG obtained directly from the PTB and trained using a plain Maximum Likelihood estimator, such as described above. With this grammar, he parses unseen sequences of POS tags (not words). Charniak's key finding is that without any smoothing or other methods for improving the probability

10 In order to allow for the computation of the score over more than one sentence, in the case of macro-averaging, a bracketing also includes a sentence number.

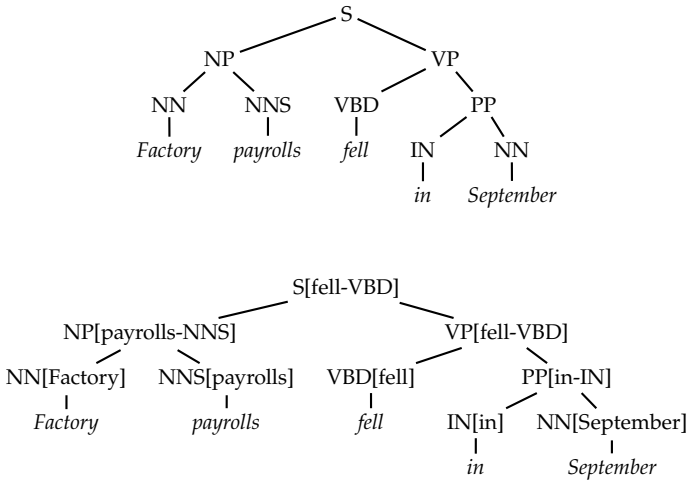


Figure 25: PCFG lexicalization

model, and without incorporating any lexical information, treebank grammars perform much better than expected. He furthermore shows that the extracted grammars place virtually no restriction on the sequence of POS tags which can be parsed. This means that in spite of the flat annotation of the PTB, which produces many rules with a low frequency, low coverage is the least concern for such a grammar (even though at the time, this was thought to be a problem). Amongst other things, though, Charniak suggests that incorporating lexical information offers possibilities for improvement. He supports his claim citing the successful decision-tree-based model of Magerman (1995).

Many high-performance parsers are based on the *lexicalization* of PCFGs (Collins, 1999; Charniak, 1997, 2000; Dubey and Keller, 2003; Klein and Manning, 2003c). The basic idea of lexicalization is to make use of the lexical heads of phrases to augment the grammar. As an illustration, figure 25 shows a constituency structure without and with head annotation (example borrowed from Klein and Manning (2003c)). Productions of a lexicalized grammar can be seen as having the form $A[a] \rightarrow B[b]C[a]$, where A, B, C are non-terminals and a, b are lexical elements or a combination of lexical elements and POS tags (Eisner and Satta, 1999). The terminal, resp. terminal/POS tag combination on the left-hand side (LHS) (a) is thereby inherited from a single right-hand side (RHS) non-terminal. This kind of grammars can en-

code, e. g., selectional preferences. Consider the following lexicalized productions extracted from the annotated example tree.

$$\begin{aligned}
\text{VP[fell-VBD]} &\rightarrow \text{VBD[fell] PP[in-IN]} \\
\text{PP[in-IN]} &\rightarrow \text{IN[in] NN[September]} \\
\text{VBD[fell]} &\rightarrow \text{fell} \\
\text{IN[in]} &\rightarrow \text{in} \\
\text{NN[September]} &\rightarrow \text{September}
\end{aligned}$$

The productions derive the VP *fell in September*. By virtue of not adding certain other lexicalized productions to the grammar, one can avoid generating, e. g., *fell in Monday*.

With a lexicalized grammar, the use of a Maximum Likelihood Estimator is no longer feasible due to the data sparseness caused by the highly increased number of symbols. Collins (1999) therefore proposes to refrain from generating the right-hand sides of rules in a single step, and to generate them instead step by step, since the corresponding probabilities are easier to estimate. In his three parsing models, the lexicalized PCFG rules have the form

$$P[h] \rightarrow L_n[l_n] \cdots L_1[l_1] H[h] R_1[r_1] \cdots R_m[r_m]$$

where H is the head child and $L_n[l_n] \cdots L_1[l_1]$, resp. $R_1[r_1] \cdots R_m[r_m]$ are left, resp. right modifiers of H, with $n, m \in \mathbb{N}_0$. The generation of the RHS of a rule is decomposed into three steps. First, the head constituent label is generated with the probability $p_h(H \mid P, h)$. Then the modifiers on the left of the head are generated using a Markov process, with the probability $\prod_{i=1}^{n+1} p_l(L_i[l_i] \mid P, h, H)$ where $L_{n+1}[l_{n+1}] = \texttt{STOP}$ (The symbol `STOP` is added to the non-terminal alphabet). Then, the modifiers right of the lexical head are generated, also using a Markov process with the probability $\prod_{i=1}^{m+1} p_r(R_i[r_i] \mid P, h, H)$ which generates `STOP` on finishing, i. e., we define $R_{m+1}[r_{m+1}]$ as `STOP`. Collins' models incorporate many additional features which model linguistic facts ("everything is conditioned on everything"). For example, the generation of modifiers is conditioned on their distance to the head, since the probability of occurrence of a certain modifier can indeed depend on its distance to the head. For the left modifiers, one would include such a feature as follows.

$$p_l(L_i[l_i] \mid H, P, h, L_1[l_1] \ldots L_{i-1}[l_{i-1}]) = p_l(L_i[l_i] \mid H, P, h, \Delta_l(i-1))$$

where Δ is a function yielding the distance. In order to account for sparse data, elaborate smoothing, resp. back-off techniques are necessary which ensure that if a certain fine-grained class of training data is not available, a more coarse-grained class can be recurred to.

Dubey and Keller (2003) are one of the first to present non-English work on parsing.[11] They present a lexicalized model for German. NeGra, the treebank they are using, has a much flatter annotation than the PTB (see p. 101); e. g., other than in the PTB, PP-internal NPs are not annotated. To compensate for the flatness, amongst other things, they propose a *sister-head* model as an alternative to Collins' *head-head* model. Instead of conditioning modifiers on the LHS and the head, they condition them on the LHS and their previous sister. Dubey and Keller find that while an unlexicalized baseline model outperforms the lexicalized model using head-head dependencies, the sister-head dependencies give better performance than the baseline model. This suggests at least that the benefit of lexicalization does not generalize straight-forward from English to German (and possibly other languages). This concurs with the findings of Kübler et al. (2006, 2008).

Lexicalization comes with a price. Estimating the model parameters of a lexicalized PCFG is highly complex; furthermore, lexicalized PCFG have a higher parsing complexity than PCFG, namely $\mathcal{O}(n^5)$, resp. $\mathcal{O}(n^4)$ using the "hook trick" (Eisner and Satta, 1999). The higher parsing complexity requires a reconciliation of the goal of finding a good parse with the goal of finding it as quickly as possible. Generally, the idea is to refrain from searching the parse which is globally optimal. *Beam search* restricts the number of edges tracked at any moment during parsing to a certain maximum. With a well chosen threshold, beam search can provide enormous speed-up while doing few harm to the parsing results. In *Best-First Parsing* (Caraballo and Charniak, 1998), edges are rated in function of their ability to lead to a complete parse. The value which is computed for a single item is called its *Figure-of-Merit (FOM)*. During parsing, always the item with the best FOM is processed first.

Klein and Manning (2003b) revisit the possibilities of unlexicalized parsing. They show that without relying on lexicalized grammars, good results can be achieved, and estimating the probability model is much easier. To improve their unlexicalized grammars, the authors

11 Previously, Collins et al. (1999) had applied the Collins parser to Czech and Bikel and Chiang (2000) had applied two lexicalized models to Chinese.

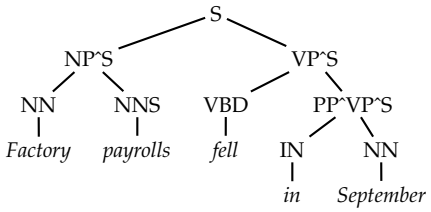


Figure 26: Grand-parent annotation

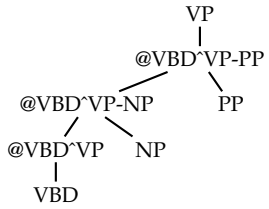


Figure 27: $v = 1, h = 1$ markovization of VP $\rightarrow$ VBD NP PP

augment unlexicalized PCFG productions in three ways. Firstly, rules are equipped with *parent annotation* (Johnson, 1998). This means that before grammar extraction, every non-terminal in the tree is annotated with its parent label, and possibly also with its grand-parent label. As an example, figure 26 shows the upper tree from figure 25 with grand-parent annotation.[12] Secondly, the rules are horizontally *markovized*. Horizontal markovization mimics the head-outward generation of the RHS of the rule: First, the head is generated, then the modifiers. In fact, parent annotation and horizontal markovization are two instances of the same idea, and can be called *vertical* and *horizontal markovization*. They break down the strong independence assumptions of PCFG by reducing the horizontal and by augmenting the vertical context of a rule. As an example, figure 27 shows the markovization ($v = 1$, $h = 1$) of a production VP $\rightarrow$ VBD NP PP (VBD being the lexical head). Thirdly, Klein and Manning exploit the fact that in treebank trees, certain symbols have a characteristic distribution, i. e., they occur only in certain contexts. Klein and Manning find that annotating grammar symbols (*splitting* them) w.r.t. their context is beneficial. An example for such an annotation is that the symbol for auxiliary verbs is split into two symbols, one for *be* and one for *have*.

12 The example follows Johnson (1998) in that the POS tags are not annotated – he parses POS tags and not natural language tokens.

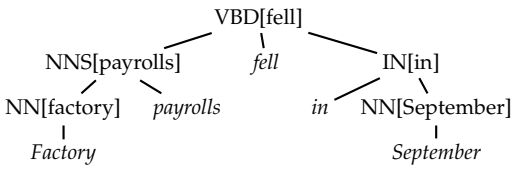


Figure 28: Factoring out lexicalization

Klein and Manning (2003c) take advantage of the fact that bilexical dependencies can be encoded by dependency structures. This is easy to see in the example above: Every lexicalized (CNF) production involves exactly two lexical elements. Figure 28 shows the factored dependency tree for the example in figure 25.

The Stanford Parser, which implements their work, consists of a combination (Klein and Manning, 2003c) of the unlexicalized parser described in Klein and Manning (2003b,a), and a dependency parser.

Latent Variable Grammars and Coarse-To-Fine Parsing

Klein and Manning improve unlexicalized parsing by splitting grammar symbols according to the linguistic properties of their data. However, splits (and merges) can also be done automatically.

The work of Slav Petrov and the Berkeley NLP group (Petrov et al., 2006; Petrov and Klein, 2007; Petrov, 2010) has established the current state-of-the art in statistical parsing. Before Petrov, experiments had been done with a fixed number of category splits (e. g., an NP split into NP-1 to NP-8) (Matsuzaki et al., 2005; Prescher, 2005). However, the problem with those methods is that some categories are oversplit while others are undersplit. Petrov et al. (2006) introduce an improved system which performs splits (and merges) based on the benefit they bring for parsing, i. e., it splits only when needed. For this task, it uses the *EM algorithm* (Dempster et al., 1977). In the expectation step, the probability of each rule with split labels and position in each training set tree is computed (using the corresponding inside and outside probabilities). In the Maximization step, those probabilities are used to update the rule probabilities. The grammar is then modified by repeatedly splitting and retraining it. Overfitting is avoided by remerging splits (splits are undone if the loss of likelihood for removing them falls below a certain threshold) and by smoothing.

An alternative approach to splitting, resp. merging symbols has been introduced by Ule (2003, 2006). Ule's work, which unfortunately has largely been overlooked, is based on Bockhorst and Craven (2001). The crucial difference between his and Petrov's approach is that he computes the benefit of splits on the basis of a more local context, i. e., instead of relying on inside/outside probabilities, he looks at changes of likelihood of split labels occurring in a certain context, the context being just the corresponding parent label. Since no experimental data are available on how this method performs on the PTB, no direct comparison to Petrov's work is possible. It is to be suspected that the method performs worse; in exchange, it should also be computationally less demanding by an order of magnitude.

A related line of work is called *coarse-to-fine parsing* (Charniak et al., 2006; Petrov, 2009). The idea of this technique is to split the task of learning a probability model from a treebank grammar into different steps. Instead of using a single treebank grammar G, one uses grammars G_1 to G_n. Thereby, G_{i+1} is a *refinement* of G_i, in the sense that the symbols in G_{i+1} refine the symbols of G_i, such that there is a surjective mapping which maps every symbol A' in G_{i+1} to a unique symbol A in G_i. In other words, in G_{i+1}, the symbols of G_i are split into equivalence classes. During parsing, the coarser grammars can be used to prune the search space of the finer grammars.

Exploiting Higher Symbolic Expressivity

The problem of PCFG is that its independence assumptions are too strong: The context within a derivation which is covered by one CFG production, i. e., by a subtree of height 1, is not big enough. As we have seen, this limitation can be overcome to a certain extent with a more expressive probabilistic model. Alternatively, one can use formalisms which provide more expressivity on the symbolic side, such as Tree-Adjoining Grammar (TAG) (p. 32). Given their extended domain of locality, TAG elementary trees capture a bigger context than CFG rules and allow for the estimation of simple but meaningful probability models. The disadvantage of such approaches w.r.t. CFG is that grammar extraction from treebanks is more difficult, probabilities are more difficult to estimate and the parsing complexity is higher.

DATA-ORIENTED PARSING *Data-Oriented Parsing (DOP)* (Bod and Scha, 1996), resp. its first incarnation called *DOP-1* (Bod, 1995), is based on *Stochastic Tree Substitution Grammar (TSG)*. TSG can be seen as TAG lacking the adjunction operation. It works as follows. Take a collection of syntactic structures and extract every subtree of every syntactic structure. Take these trees to be the elementary trees of a TSG and assign a probability to each tree which is the count of occurrences of the tree as a subtree in the collection of syntactic structures divided by the sum of the counts of trees which have the same root label. Figure 29 shows an example with a single tree and the STSG extracted from it, including the probabilities.

For DOP, interesting linguistic arguments have been given (Scha, 1990). Apart from that, excellent parsing results (accuracy far over 90%) have been reported (Bod, 1995). Just as with Probabilistic Tree-Adjoining Grammar (PTAG) (see below), the good results can be attributed to the fact that the subtrees capture a larger context than PCFG rules. The disadvantage of DOP is that parsing with DOP-1 is very expensive. Unlike PTAG, which settles for the most probable derivation (i. e., derivation tree), DOP-1 searches for the most probable parse (i. e., the most probable *derived* tree, where the probability of a derived tree is the sum of the probabilities of all underlying derivations). This problem is NP-complete (Sima'an, 1996). Various methods exist which make DOP parsing (more) tractable. Bod (1993) uses a *Monte Carlo* algorithm which estimates the derivation probability. Bod (1995) chooses to randomly sample 5% of the trees of his grammar. Sima'an (1999) proposes restrictions on the grammar such as limiting the number of substitution nodes or limiting the tree depth in the STSG. Goodman (2003) presents a summary of optimizations for DOP-1 from the literature, including his own method of reducing the input STSG to an equivalent PCFG.

PROBABILISTIC TREE-ADJOINING GRAMMAR The first probabilistic variant of TAG, which we will call *Probabilistic Tree-Adjoining Grammar (PTAG)*, has been proposed by Schabes (1992) and at the same conference by Resnik (1992), who provides an equivalent but more direct definition. PTAG works as follows. A probability is assigned to the selection of an initial tree which can be the valid start of a derivation; subsequent adjunction and substitution operations are treated as independent events with their own probabilities. The probability of

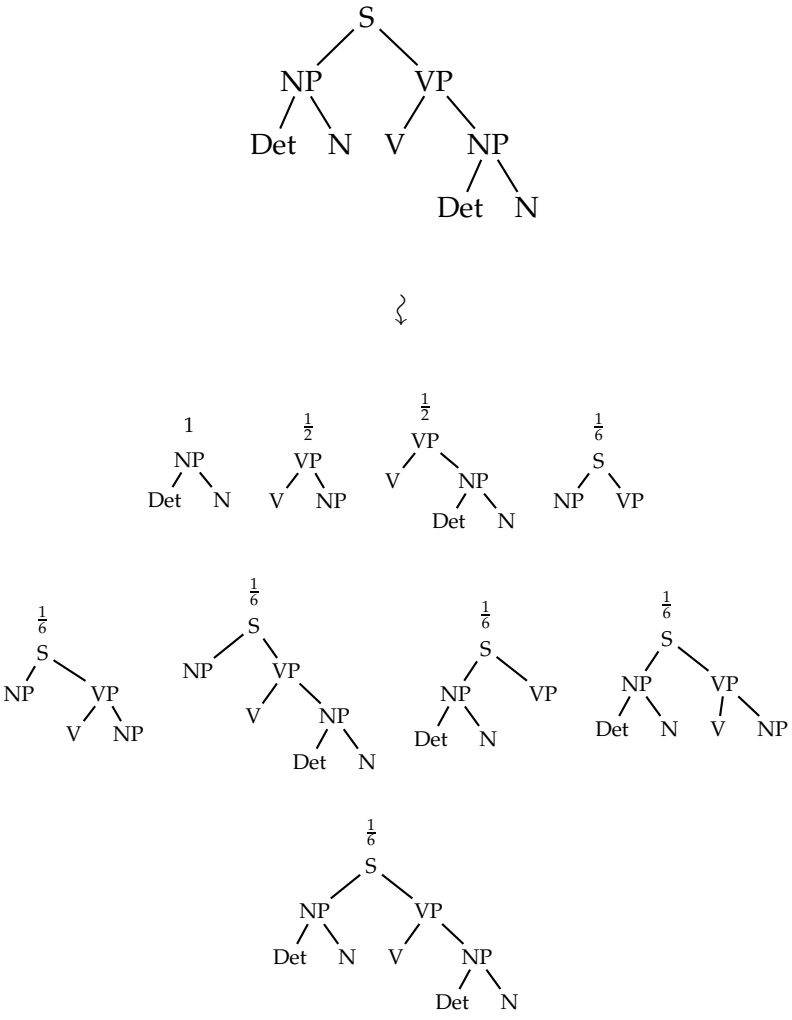


Figure 29: DOP: From a tree to a STSG

a derivation is the product of the operations which have lead to it. Sarkar (1998) investigates the conditions for the consistency of PTAG.

A popular approach for obtaining PTAGs from treebanks is to use heuristics in order to reconstruct derivations from treebank trees, i. e., treebank trees are treated as TAG derived trees and the heuristics are used to identify the elementary trees the derived tree is composed of. Xia (2001) presents a flexible system for obtaining grammars automatically, based on this technique. In her system, the linguistic knowledge used by the extraction heuristics (such as head percolation tables) can be parametrized. She applies her system to the PTB. TAG extraction has also been tried on German treebanks. Frank (2001) extracts TAG from NeGra, Kaeshammer and Demberg (2012) extract a TAG from TIGER. Yet another technique is presented by Chiang (2003), who uses his extracted grammars for data-driven parsing (see below). He uses a variant of TAG which allows for *sister adjunction*, an operation which consists of inserting the root of an initial tree as a new daughter of an interior node. The sister adjunction operation helps to account for the rather flat annotation of the PTB. Other variations of TAG extraction methods exists, most of them tailored to the properties of specific treebanks.

Extracted grammars have been used for data-driven parsing; see, e. g., Joshi and Sarkar (2003), Chiang (2003) (both articles are revised versions of previous work of the resp. authors) and Shen and Joshi (2005). In comparison with PCFG, good output quality can be achieved at a reasonable speed.

PROBABILISTIC COMBINATORY CATEGORIAL GRAMMAR The formalism of *Combinatory Categorial Grammar (CCG)* (Steedman, 2000) has more expressivity than CFG. The additional expressivity is provided by complex syntactic categories which are assigned to every lexical item in the lexicon. Much like elementary trees of a TAG, they encode the lexical item's contribution to a complete derivation, in other words, word order and subcategorization information. The syntactic category of the verb *likes*, for example, is notated (S/NP)\NP. Constituents can be combined using combination rules. The two simplest ones are *backward composition* (e. g., a constituent with the category (S/NP)\NP can be *backward composed* with a adjacent NP constituent on its left into a constituent with category (S/NP)) and *forward composition* (e. g., a constituent with category S/NP can be *forward composed* with a NP

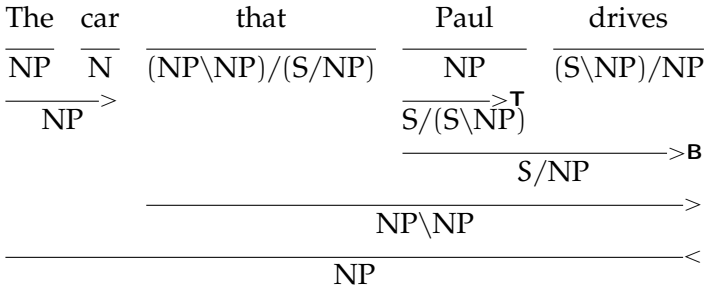


Figure 30: Combinatory Categorial Grammar example

constituent on its right into constituent with category S). Other operations are *type raising* and *functional composition* (the latter allows to combine two categories of the form X/Y and Y/Z into a category of the form X/Z). Figure 30 shows an example of a CCG derivation of an NP with a relative clause in the canonical notation. Forward and backward composition are marked with $>$ and $<$, respectively; type raising is marked with **T**, functional composition with **B**.

Hockenmaier and Steedman (2002a) describe how to transform the trees of the PTB into CCG normal form[13] derivations. Hockenmaier and Steedman (2002b) and Clark et al. (2002) present parsers, the former based on generative modeling and the latter on conditional modeling. Hockenmaier and Steedman (2002b) almost achieve Collins' performance on the PTB, despite their less fine-tuned approach.

4.2.3 *Beyond Context-Free Derivations*

We do not only want to build CFG parse trees. In this thesis, we are particularly interested in ways of reconstructing the annotation information beyond context-freeness. There are different approaches to this task in the literature.

PCFG plus Post- or Preprocessing

One possibility is to pre-, resp. post-process the input, resp. output of a PCFG parser. A post-processing approach is presented by Johnson (2002). He uses a pattern-matching algorithm which recovers empty nodes and their antecedents of the PTB. The algorithm can in fact

13 For a discussion of the CCG normal form, see Eisner (1996a).

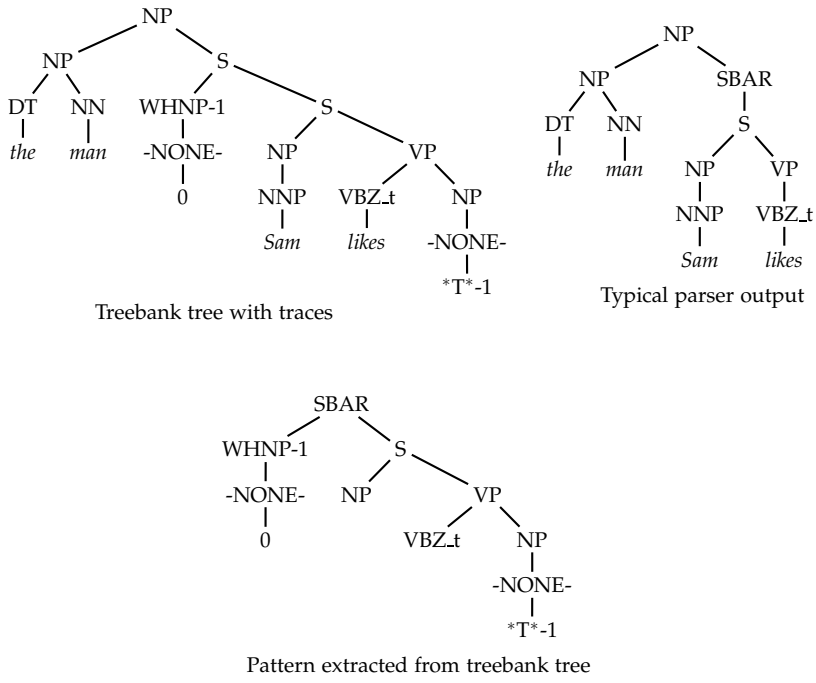


Figure 31: Empty node recovery by pattern substitution

be seen as an instance of Memory-Based Learning (MBL). As patterns, Johnson uses minimal connected tree fragments which contain an empty node and all nodes which are co-indexed with it. The motivation for this is that the path between an empty node and its antecedent gives cues on where empty nodes may occur and where not. Intuitively speaking, a pattern *matches* a certain tree if this tree is an extension of the pattern, ignoring the empty nodes. Empty nodes are then inserted into some parser output tree t by substituting the subtree of t matching p by p (using a pre-order traversal). As an example, consider figure 31, taken from Johnson (2002). It displays a treebank tree, a pattern for the empty nodes which would be extracted from that tree and a tree from the parser output which, by pattern substitution, could be transformed into the original treebank tree. Johnson evaluates the positions of the empty nodes with the Evalb measure and the quality of the coindexation by a version of the Evalb measure in which the label representation is augmented. Jijkoun (2003) and Campbell (2004) present improved versions of Johnson's strategy.

Dienes and Dubey (2003a,b) use a pre-processing approach. They insert empty elements between words of the unparsed sentence using a Maximum Entropy model and give the parser the modified input. Their approach outperforms Johnson's technique.

Levy and Manning (2004) also recur to machine learning techniques to reconstruct empty nodes and their antecedents. Their algorithm can be described as a mixture of Johnson's and Dienes and Dubey's approaches. They use post-processing, like Johnson. However, they do not treat empty node insertion and antecedent identification in one step. In some cases, they identify empty categories first (like Dienes and Dubey), in other cases they proceed in the inverse direction. The enhancement of the CFG parse trees is done in multiple steps. In each phase, loglinear classifiers are used to identify a certain subset of tree nodes, which are then modified using appropriate operations (see section 3 of the article). An evaluation shows that their technique compares favorably to the two previous approaches. A novelty about Levy and Manning's work is that they apply their method not only to the PTB, but also to NeGra.

Discontinuous Phrase-Structure Grammar

Alternatively to augmented PCFG parsing, one can use a formalism which accommodates the direct encoding of information beyond CFG in its derived trees. *Discontinuous Phrase Structure Grammar (DPSG)* (Bunt et al., 1987; Bunt, 1991, 1996) is such a formalism. It has the aim of extending CFG in order to allow for trees with crossing branches (see p. 49). Plaehn (1999, 2004) proposes the use of DPSG for data-driven parsing. His work is particularly relevant for this thesis since he uses DPSG to model the German discontinuous constituents of NeGra. Furthermore, to my knowledge, his work contains the only evaluation results for parsing NeGra without removing the crossing branches.

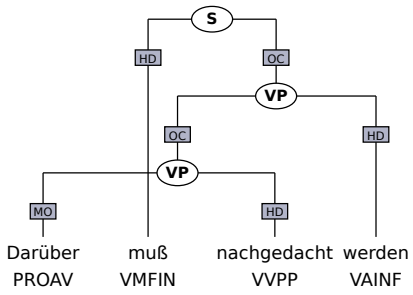
Analogous to a PCFG, a *Probabilistic Discontinuous Phrase Structure Grammar (PDPSG)*, as presented by Plaehn (2004), consists of a DPSG and a function which maps all rules to a probability such that the probabilities of all rules with the same left-hand sides sum up to 1. Plaehn presents a directional bottom-up chart parsing algorithm with which discotrees can be built. Note that the directionality, which is achieved through the use of dotted productions, implies a fixed bi-

narization strategy. Given a grammar which contains no productions with context daughters, the parser behaves as a PCFG parser. The algorithm uses items, an agenda and a chart (cf. p. 56). A constituent is represented by a label and two bit vectors of the length of the input string, one representing the terminals the constituent dominates, the other representing the terminals its context daughters dominate (the ith terminal is dominated iff the ith bit of the resp. vector is set). When building new edges, this representation allows to easily check the compatibility of two constituents by using bitwise operations on the bit vectors.

Plaehn furthermore presents a Viterbi-like extension for his parsing algorithm which finds the most probable parse. His algorithm is exhaustive, i. e., in order to search for the most probable derivation, all possible sub-derivations are built. The need for updates of larger constituents is avoided by making the parser proceed strictly bottom-up, i. e., a constituent is only built when the space of all possible subderivations has been searched (Plaehn, 2004, p. 100).

Plaehn reports results for experiments on "an early version of the second release of the corpus", containing 20,571 sentences (the final second release contains 20,602 sentences), on both an extracted DPSG and an extracted PCFG, obtained from a "continuified" version of NeGra (see above). A method is described for extracting a DPSG from NeGra (Plaehn, 2004, p. 101), the difficulty being the identification of the context daughters. In a nutshell, context daughters convey information about all those gaps in the yield of a constituent which lie *between* the yields of direct children of the constituent (the absence of a gap signifies that there are no context daughters). The set of context daughters for such a gap is the set of the gap's maximal nodes (def. 5.7, p. 135). As an example, figure 32 shows again the NeGra annotation of (3), and the DPSG productions extracted from it. To obtain probabilities, a Maximum Likelihood Estimator is used. Parsing is limited to sentences with a length of 15 words and less. As an evaluation, Plaehn reports among other things labeled and unlabeled precision, recall and F-measure. While for the DPSG, the average parsing time per sentence is almost 50 times[14] higher than for the PCFG, the results

14 Plaehn mentions the parsing complexity to be exponential, citing Reape (1991). DPSG therefore does not seem to be an MCS formalism, other than its restricted variant DPSG$^{\tau}$ (see p. 50).



$$
\begin{array}{rcl@{\qquad}rcl}
\text{S} & \rightarrow & \text{V VP} & & & \\
\text{VP} & \rightarrow & \text{VP} & \text{VP} & \rightarrow & \text{PROAV [V] VVPP} \\
\text{PROAV} & \rightarrow & \text{Darüber} & \text{VVPP} & \rightarrow & \text{nachgedacht} \\
\text{VVFIN} & \rightarrow & \text{muss} & \text{VVINF} & \rightarrow & \text{werden}
\end{array}
$$

Figure 32: DPSG extraction

lie in the same range (73.16 labeled F score for DPSG, 74.57 labeled F score for PCFG).

Multiple Context-Free Grammar

Levy (2005) uses another formalism for data-driven parsing, namely Probabilistic Multiple Context-Free Grammar (PMCFG).[15] The same formalism is used in a slightly restricted variant by Kato et al. (2006) for RNA pseudoknot modeling (see section 3.3). Grove (2010) uses PMCFG as a backend to parse Minimalist Grammar. In the following chapters, we will use Probabilistic Simple Range Concatenation Grammar (PSRCG) for probabilistic parsing. Therefore, we postpone a detailed analysis of these works to section 6.4 in order to compare them to the contributions of this thesis.

15 In fact, on his grammars, which he calls *Probabilistic Wrapping Grammars*, he imposes the bottom-up non-erasing condition, as well as a condition he calls *strict concatenation*, which amounts to either only having variables in a function, or a single terminal. The grammars we will extract from treebanks (see p. 137) have the same form.

4.2.4 *Related Work*

Other Formalisms

Lexical Functional Grammar (LFG) (Bresnan, 1982) and also *Head-Driven Phrase Structure Grammar (HPSG)*[16] (Pollard and Sag, 1994; Sag and Wasow, 1999) are two formalisms the cores of which are based on feature structures (cf. p. 33). Both are much more expressive than the previously mentioned formalisms. They have been used for data-driven parsing.

In LFG, the syntactic information for a sentence is encoded in two levels. The *c-structure* takes the form of context-free derivation trees and represents the constituency structure. The *f-structure* is a feature structure (cf. p. 33) which encodes amongst other things predicate-argument relations and agreement information. Riezler et al. (2002) and Kaplan et al. (2004) present systems which use a hand-written grammar (Butt et al., 1999) for parsing the WSJ, together with a discriminative component for parse selection. Other researchers obtain LFGs directly from the treebank. The difficult part about this is the induction of f-structures. A method for f-structure induction is described, e. g., in Burke et al. (2004) and Cahill (2004). The latter uses the extracted grammars for parsing, out-performing previous systems based on hand-written grammars. LFG-style information can also be used to improve the performance of data-driven parsing for other languages such as Arabic (Tounsi et al., 2009) and French (Schluter and van Genabith, 2008).

HPSG completely relies on feature structures and on axioms (*principles*) prescribing their shape. The field of linguistics has seen many hand-implemented grammars. However, HPSG fragments have also be obtained automatically and used for data-driven parsing. See, e. g., Miyao (2006) and Miyao and Tsujii (2008).

Psycholinguistically Motivated Work

A completely different strain of work which is not of direct relevance to this thesis, but nevertheless very interesting, be mentioned for the sake of completeness. This work has its motivation in the field of psycholinguistics and is concerned with creating parsers which mimic as

16 Consult Levine and Meurers (2006) for an introductory overview.

well as possible the sentence processing of the human mind. A key property of such parsers is that they process input sentences strictly from left to right and maintain at any moment a fully connected parse tree. Formally, most systems are restricted to the expressivity of CFG. One example for such a system is Roark (2001). Recent ideas on how to integrate expressivity beyond context-freeness have been presented by Schuler et al. (2010).

Reranking

Certain constraints in a probabilistic parsing model are most easily expressed as features which discriminate complete derivations. *Reranking*[17] takes advantage of this. Typically, a *reranker* takes as input a list of parses of a sentence ordered by their respective output probabilities and outputs the best parse among them according to derivation-global features. The input list can be obtained from a k-best parser (Huang and Chiang, 2005; Pauls and Klein, 2009). Training is done using machine learning techniques. The reader is referred to the literature for information about successful systems (Collins, 2000; Collins and Duffy, 2002; Kudo and Matsumoto, 2004; Collins and Koo, 2005).

4.3 DATA-DRIVEN DEPENDENCY PARSING

This section contains a short overview of the encoding of dependency annotation and of dependency parsers.

4.3.1 *Dependency Treebank Annotation*

Dependency treebank annotation generally contains more information than the bare dependency structures. The Czech Prague Dependency Treebank (PDT) (Hajič et al., 2000), e. g., has three annotation layers. The first layer contains morphological information, the second layer contains the dependency structure itself, and the third layer (*tectogrammatical layer*) contains further non-local relations.

Here, we are only concerned with the part of the dependency annotation which denotes a dependency structure (def. 2.12, p. 22). The

17 In machine learning terms, reranking is a *discriminative model* (as opposed to a *generative model*).

```
1   Třikrát     _   Cv     Cv     _   2   Adv    _   _
2   rychlejší   _   AAFS   AAFS   _   0   ROOT   _   _
3   než         _   J      J      _   2   AuxC   _   _
4   slovo       _   NNNS   NNNS   _   3   ExD    _   _
```

Figure 33: CoNLL format example

graph can be represented conveniently using the *CoNLL interchange format*. Just as the `export` format, the CoNLL format is line-based. A dependency structure over a sentence of length n is represented by n lines of text, where the ith line contains information about the node associated with the ith word in the sentence. There is neither a separate line for the root node, nor a sentence numbering. As an example, figure 33 shows the annotation of (12), a sentence from the PDT.

(12) Třikrát rychlejší než slovo
Three times faster than word
"Three times faster than words"

Only some of the information provided by the CoNLL format is necessary to reconstruct a dependency structure, namely the first field, containing the index of a token in the sentence, the second field, containing the token itself, the fourth and fifth field[18] which are used for POS tags, the sixth field, containing the number of the head of the word, and the seventh field, containing the label of the edge to the head.

See chapter 8 for more information on dependency treebank preprocessing.

4.3.2 *Data-Driven Dependency Parsers*

Transition-based and grammar-based parsers greedily search for locally optimal *transitions* (edges). Graph-based parsers are globally optimized. They search the space of complete dependency structures without considering the properties of single transitions. Hybrid sys-

18 One of them would of course be enough. We use both in order to maintain compatibility with popular tools using the CoNLL format, such as the evaluation module of the MSTParser (see p. 124).

tems aim at combining the advantages of transition-based and graph-based systems.

Note that dependency parsing is not the core subject of this thesis. I will therefore only mention a fraction of the available literature on dependency parsing; however, the presented material should give a general idea of the field.

Transition-Based Parsing

The MALT parser (Nivre et al., 2007) is a system for *inductive parsing*. It offers various methods for inducing a classifier from a dependency treebank and for using this classifier to produce dependency trees on unannotated sentences. MALT includes algorithms for building projective trees (see, e. g., Nivre (2003)) and non-projective trees (see for instance Covington (2001), Nivre and Nilsson (2005), or Gómez-Rodríguez and Nivre (2010)). Dependencies are generally built from left to right. Thereby, the classifier is used to determine the most likely incoming, resp. outgoing arcs from each word.

Graph-Based Parsing

Another way of formulating the problem of non-projective dependency parsing can be found in McDonald et al. (2005). They view dependency structures as weighted directed graphs, i. e., as directed graphs in which each edge has a weight. The *score* of such a graph is the sum of all of its edges' scores. Considering all possible edges (with given weights) of a graph $\mathcal{G}$, its *Maximum Spanning Tree (MST)* is the tree with the highest score that contains all nodes of $\mathcal{G}$. Finding the most probable dependency parser amounts to searching the space of all possible spanning trees. This can be accomplished efficiently with the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). Given an adequate implementation, the algorithm can run in as few as $\mathcal{O}(n^2)$. For a graphical example, consult McDonald et al. (2005).

The advantages of this greedy algorithm are its efficiency, the accuracy of its output, and the fact that it can treat non-projective dependencies. Its disadvantage is that only *edge factorization* is tractable, i. e., one must make the unrealistic assumption that all edges are independent of one another (McDonald and Satta, 2007). However, elegant approximations of more powerful models exist (McDonald and Pereira, 2006).

Another instance of graph-based systems are constraint-based systems. They generally rely on a (hand-written) set of rules, resp. constraints, which model dependency graphs. An example is *Weighted Constraint Dependency Grammar (WCDG)* (Foth et al., 2004). Systems such as WCDG are generally not restricted to projective dependencies.

Hybrid Systems

Other systems try to combine the advantages of both locally optimized and globally optimized systems. Among the very different approaches, there is a recent promising one which has been presented by Goldberg and Elhadad (2010) under the name of *easy-first parsing*. In a nutshell, Goldberg and Elhadad use a greedy strategy comparable to the strategy of the MALT parser, with the difference that strict left-to-right processing is given up and "easy" decisions are taken first. Easier decisions are decisions with which the parser is more confident; confidence is measured on the basis of scores of feature vectors extracted from the training data. Goldberg and Elhadad (2010) achieve a performance between the MALT parser and the MST parser.

Grammar-Based Parsing

So far, to my knowledge, grammar-based parsing of dependencies has only been explored for projective dependencies. The first parsing models were presented by Eisner (1996b). He exploits the closeness of projective dependencies to CFG and essentially uses bottom-up chart parsing. More recently, Bangalore et al. (2009) have presented a probabilistic dependency parser based on *Tree Insertion Grammar (TIG)*, a restricted variant of TAG (Schabes and Waters, 1995).

The advantage of grammar-based parsing is that bottom-up chart parsing is available, together with all related techniques. This allows for detailed probabilistic models. The disadvantage is that (until now), parsing is restricted to projective structures.

4.4 SIMPLE RCG FOR DATA-DRIVEN PARSING

In the following chapters, techniques for data-driven parsing of discontinuous constituency structures and non-projective dependencies

using Probabilistic Simple Range Concatenation Grammar will be presented, as well as an investigation of the properties of discontinuous treebank annotation. What is the motivation for this work?

4.4.1 *Constituency Parsing*

The data source for the development of most data-driven parsers has been, and continues to be, the Penn Treebank. However, since the first studies on languages other than English, it is clear that models which work well for English do not necessarily generalize to other languages. Even models which work well on the PTB do not automatically generalize to other English treebanks (Gildea, 2001). People have tried to measure parsing difficulty, resp. to find the factors which influence it, and they have tried to identify the features which help most for parsing a certain language (Dubey and Keller, 2003; Levy and Manning, 2003; Corazza et al., 2004, 2008; Kübler, 2005; Maier, 2006; Kübler et al., 2006, 2008; Rehbein and van Genabith, 2007c,b; Tsarfaty and Sima'an, 2007). Recent workshops show a rising interest in this kind of research (Kübler and Penn, 2008; Seddah et al., 2010). This thesis contributes to this area by taking up a German constituency treebank with discontinuous annotation and by investigating the properties of its annotation.

Discontinuous constituents occur especially frequently in languages with a free word order, such as German (cf. chapter 5). A particularly interesting research question for such languages is therefore how discontinuous constituents can be reconstructed. The previous section summarizes the literature on this topic. One of the techniques, namely the direct parsing of discontinuous constituents using an MCS formalism, has not found much attention so far, despite Levy's (2005) promising work (cf. p. 120). We follow up on his approach. For several reasons, we also choose SRCG[19] as a formalism.

- SRCG is a candidate for modeling discontinuous constituents. Grammar extraction is almost immediate due to the proximity of SRCG to CFG (cf. p. 4), and discontinuous structures and their properties are reflected in an immediate way in the grammar. This will be made clear in detail in chapter 5.

19 Recall that SRCG is equivalent to Multiple Context-Free Grammar (MCFG) and Linear Context-Free Rewriting System (LCFRS).

- SRCG is fairly efficiently processable. It has a polynomial parsing complexity which depends directly on the degree of discontinuity which is involved. CFG is simply a special case of SRCG which produces only derivations without discontinuities.
- Probabilistic variants of SRCG have been proposed before (Levy, 2005; Kato et al., 2006; Søgaard, 2011). The proximity of SRCG to CFG allows a direct transfer of techniques which are known from PCFG parsing. Furthermore, we can exploit recent results which are important for parsing SRCG, particularly on binarization techniques (Gómez-Rodríguez et al., 2009a).

Discontinuous Phrase Structure Grammar has also been used for the modeling of discontinuous constituents (Plaehn, 2004) (see p. 118). However, SRCG has advantages over DPSG. In DPSG, discontinuous structures are reflected less directly in the grammar, due to the fact that in the productions, one must specify the material in the gaps of a node yield. This also makes grammar extraction more difficult, as explained before. The other, more essential disadvantage of DPSG is that it is not an MCS formalism: Its parsing complexity is exponential.

Other than Levy (2005), we will perform a detailed evaluation of our parser output. Outside estimates for accelerating parsing speed will be introduced, and the effect of different binarization techniques will be investigated. A further detailed comparison of Levy's work, Plaehn's work and this work is postponed to section 6.4, as mentioned before.

4.4.2 *Dependency Parsing*

Given the fact that there is a highly optimized data-driven dependency parser which runs in $\mathcal{O}(n^2)$ and gives state-of-the-art performance, one might question the benefit of using a highly expressive grammar formalism with a much higher complexity for dependency parsing. The motivation for tackling dependency parsing in this thesis is two-fold.

Firstly, one can immediately interpret non-projective dependency structures as LCFRS derivations (Kuhlmann and Satta, 2009). Given Kuhlmann and Satta's grammar extraction algorithm, and our parser, dependency parsing comes for free. However, the effect of different parameters is not necessarily parallel to the corresponding constituency grammars and therefore worth to be investigated.

Secondly, grammar-based dependency parsing brings a formal advantage. Previous grammar-based approaches (which essentially correspond to bottom-up chart parsing for constituency structures) produce good probabilistic models, however, they only provide projectivity. Previous algorithms for non-projective dependency parsing are greedy, considering the full set of possible dependency structures for a sentence. They work efficiently, but are also problematic due to their unreasonably strong independence assumptions: They require that all edge decisions be treated as independent. If one wants to go beyond these edge-factored models, parsing becomes intractable (McDonald and Satta, 2007). Using chart parsing for dependencies as proposed by Kuhlmann and Satta (2009) allows for a more flexible probabilistic model which can easily be augmented by techniques such as markovization or arity restrictions (McDonald and Satta, 2007).

To our knowledge, this thesis contains the first results on grammar-based non-projective dependency parsing.

5

DISCONTINUITY AND NON-PROJECTIVITY IN TREEBANKS

This chapter is dedicated to a review of the treatment of discontinuity and non-projectivity in treebanks. In section 5.1, I give an overview of the treatment of discontinuous structures, especially constituents, from a linguistic point of view. In section 5.2, I review the treatment of discontinuous constituents in such treebanks, introducing measures for the degree of discontinuity of treebank trees. The starting point for this work are measures established in dependency grammar literature. I will furthermore investigate the presence of synchronous rewriting in treebank trees in section 5.3. Section 5.4 presents related work, and section 5.5 concludes this chapter.

Material in this chapter has been previously published in Maier and Søgaard (2008), Maier and Lichte (2011), and Kallmeyer et al. (2009).

5.1 INTRODUCTION

Discontinuous phrases are common in natural language. They occur particularly frequently in languages with a relatively free word order, such as German. Consider again (11) (repeated here as (13)), taken from NeGra. In this sentence, the discontinuity is due to fronting.

(13) Selbst besucht hat er ihn nie
Self visited has he him never
"He has never visited him personally."

Comparable examples can also be found in languages with a more fixed word order, like Chinese. Consider (14) as an example for topicalization in that language.

(14) shu_1 wo zhi mai pianyi-de t_1.
book_1 I only buy cheap t_1.
"As for books, I only buy cheap ones."

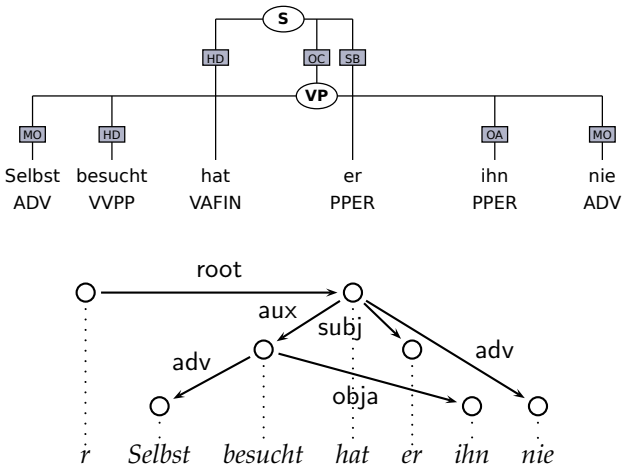


Figure 34: Discontinuity and non-projectivity in NeGra

The same sentence is also acceptable in Bulgarian, a language with a relatively free word order, as shown in (15).

(15) Knigi$_1$ kupuvam samo evtini t_1
books$_1$ buy-I only cheap t_1
"As for books, I only buy cheap ones."

As explained in chapter 4, constituency treebank annotation schemes generally use an annotation backbone based on Context-Free Grammar (CFG), and extend this backbone with an additional mechanism that accounts for non-local dependencies. In the Penn Treebank (PTB), this mechanism consists of trace nodes and labeling conventions, more precisely, co-indexation. In the Tübingen Treebank of Written German (TüBa-D/Z), the mechanism is a combination of labeling conventions and topological field annotation (cf. p. 100). In the TIGER and NeGra treebanks (cf. p. 101), non-local dependencies are represented directly, i. e., all parts of a discontinuous constituent are captured under one node. This requires the annotation backbone based on CFG to be given up, since crossing branches must be allowed. Dependency treebanks, on the other hand, generally contain non-projective structures (see p. 7). As an example, figure 34 shows the NeGra discontinuous constituency and non-projective dependency annotations of (13).

In the last years, the formal characterization of non-projectivity has received a lot of attention. Among other measures, *gap degree* and *well-*

nestedness have been introduced. Both of them were shown to empirically describe non-projective dependency data well (Kuhlmann, 2007). In contrast, a characterization of the degree of discontinuity of constituency trees with crossing branches has not been attempted yet. The aim of section 5.2 is to give a definition of gap degree and well-nestedness for constituency structures with the same descriptive and practical relevance as their Dependency Grammar (DG) counterparts. An empirical investigation on treebanks will show the expressivity of the measures. In the last chapter, we have motivated the use of Simple Range Concatenation Grammar (SRCG) for data-driven parsing of treebanks with direct annotation of discontinuous constituents. The degree of discontinuity of the treebank trees can have direct influence on the shape of the extracted grammars and therewith also on parsing complexity. Therefore, we introduce a grammar extraction algorithm for SRCG from both discontinuous constituency structures and non-projective dependency structures and investigate the relation between extracted grammars and the new measures. In SRCG derivations, unrelated parts of a derivation can evolve in a synchronous way. Since synchronous rewriting can also have an influence on parsing complexity, in section 5.3, we investigate to which degree treebank trees show synchronous rewriting and how synchronous rewriting affects extracted grammars. Section 5.4 contains related work and section 5.5 concludes the chapter.

5.2 QUANTIFYING DISCONTINUITY AND NON-PROJECTIVITY

5.2.1 *Measures for Trees and Graphs*

We formulate the measures *gap degree* (Kuhlmann, 2007) and *well-nestedness* (Bodirsky et al., 2005) which are known from DG such that they can be applied to both dependency structures and constituency structures.

Gap Degree

Both gap degree and well-nestedness can be defined on the yields of syntactic structures. Intuitively, a gap is a discontinuity in the yield of a node and the gap degree is a measure for the amount of discontinu-

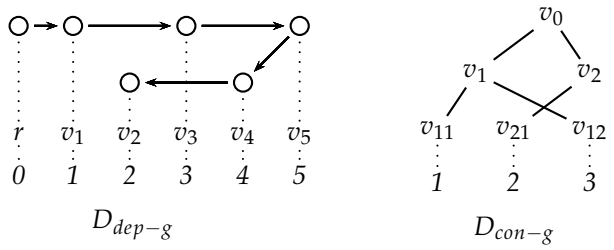


Figure 35: Gap degree

ity in syntactic structures. Recall the definition of the yield of a node (def. 2.14, p. 23).

DEFINITION 5.1 (Gap). Let $D_{syn} = (V, E, r)$ be a syntactic structure. For all $v \in V$, a *gap* in $\pi(v)$ is a pair (i_n, i_m) with $i_n, i_m \in \pi(v)$ and $i_n + 1 < i_m$ such that there is no $i_k \in \pi(v)$ with $i_n < i_k < i_m$. The *size* of (i_n, i_m) is $i_m - i_n$.

DEFINITION 5.2 (Gap degree). Let $D_{syn} = (V, E, r)$ be a syntactic structure.

1. For all $v \in V$, the *gap degree* of a v the number of gaps in $\pi(v)$.
2. The gap degree of D_{syn} is the maximal gap degree of any of its nodes.

A dependency structure with gap degree > 0 is *non-projective*; a constituency structure with gap degree > 0 is *discontinuous*. In other words, the gap degree corresponds to the maximal number of times the yield of a node is interrupted. n gaps of a node entail $n + 1$ yield blocks. This is reflected in the measure of *block degree* (Kuhlmann, 2007, pp. 35). Given the block degree function *dim* from def. 2.17, p. 24, the following is obvious.

LEMMA 5.3. Let $D_{syn} = (V, E, r)$ be a syntactic structure. The gap degree of any $v \in V$ is $dim(v) + 1$.

Figure 35 shows an example. Both syntactic structures in D_{dep-g} and D_{con-g} have gap degree 1. In D_{dep-g} both yields $\pi(v_4)$ and $\pi(v_5)$ have the maximal gap degree 1 due to the fact that they do not contain 3. In D_{con-g}, v_1 has gap degree 1 because 2 is not included in its yield. D_{dep} and D_{con} in figure 36 both have gap degree 0.

Well-Nestedness

We now define *well-nestedness* (as opposed to *ill-nestedness*) as another property of syntactic structures. Intuitively, in a well-nested structure, it holds for all nodes which do not stand in a dominance relation that their yields do not interleave.

DEFINITION 5.4 (Well-nestedness). Let $D_{syn} = (V, E, r)$ be a syntactic structure.

1. A pair of nodes $v_1, v_2 \in V$ with $\pi(v_1) \cap \pi(v_2) = \emptyset$ is *well-nested* iff there are no $i_1, i_2 \in \pi(v_1)$ and $j_1, j_2 \in \pi(v_2)$ such that $i_1 < j_1 < i_2 < j_2$.

2. D_{syn} is *well-nested* iff all pairs of nodes in V with disjoint yields are well-nested.

The following is easy to see.

COROLLARY 5.5. If a syntactic structure is ill-nested, then its gap degree is $\geqslant 1$.

Figure 36 shows well-nested and ill-nested constituency structures and dependency structures. D_{con} and D_{dep} are well-nested, while D_{con-n} and D_{dep-n} are not. Note that, while D_{con} and D_{dep} have gap degree 0 in addition to being well-nested, both D_{con-g} and D_{dep-g} in figure 35 are well-nested but have a gap degree greater than 1.

The definitions of gap degree and well-nestedness for constituency structures are novel. Nevertheless, with respect to dependency structures our definitions correspond to the definitions of gap degree and well-nestedness from the literature (Holan et al., 1998; Bodirsky et al., 2005; Kuhlmann and Nivre, 2006).

The definition of well-nestedness (resp. ill-nestedness) does not provide a notion of a degree. For this purpose, to our knowledge, three measures have been introduced so far in the DG literature. Havelka (2007) introduces *level types*, Gómez-Rodríguez et al. (2009b) introduce *strongly ill-nested structures*, and Gómez-Rodríguez et al. (2011) introduce *mildly ill-nested structures*. All three measures characterize the data considered in the respective articles well. Still, there is a good reason for introducing yet another measure. The first measure is path-based rather than yield-based, i. e., it is not directly adaptable to constituency structures. The second measure only discriminates very complex dependency structures which are unlikely to occur in natural

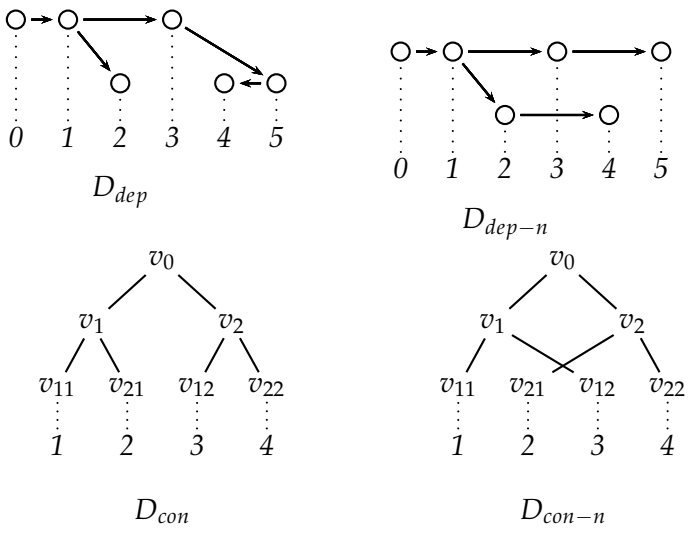


Figure 36: Well- and ill-nestedness

language, and the third measure falls short in providing a linguistic intuition due to its procedural definition.

Our measure is called k-*ill-nestedness*. We intend it to intuitively capture the degree of interleaving of yields in constituency structures and dependency structures. k stands for the number of disjoint yields that interleave with some other single yield which is disjoint from them.

DEFINITION 5.6 (k-ill-nestedness). Let $D_{syn} = (V, E, r)$ be a syntactic structure.

1. A node $v \in V$ is k-*ill-nested* iff there are exactly k nodes $v_1, \ldots, v_k \in V \setminus \{v\}$ such that
   a) the yields of all $\{v, v_1, \ldots, v_k\}$ are pairwise disjoint, and
   b) v and v_i, $1 \leqslant i \leqslant k$, are ill-nested.
2. D_{syn} is k-*ill-nested* iff k is the maximal degree of ill-nestedness of any of its nodes.

Note that 0-ill-nestedness is equivalent to well-nestedness. The dependency structure D_{dep-n} and the constituency structure D_{con-n} in figure 36 are 1-ill-nested. An example for 2-ill-nested syntactic structures is D_{con-lr} in figure 37, p. 136.

Further Properties of Constituency Structures

We now give a further formal characterization of constituency structures. We first define the *maximal nodes* of a gap. Informally speaking, the entire yield of a maximal node lies in the gap, but the yield of its parent node does not.

DEFINITION 5.7 (Maximal node). Let $D_{con} = (V, E, r)$ be a constituency structure, $v \in V$ and (i_1, i_2) a gap in $\pi(v)$. Any $v_{max} \in V$ is a *maximal node* of (i_1, i_2) iff

1. for all $j \in \pi(v_{max})$, it holds that $i_1 < j < i_2$,
2. there is a node $u \in V$ with $\langle u, v_{max} \rangle \in E$, and there is a $k \in \pi(u)$ with $k \leqslant i_1$ or $k \geqslant i_2$.

The node u is called a *gap filler*. A gap (i_1, i_2) can have more than one maximal node, and the combined yield of all maximal nodes is the set $\{i \mid i_1 < i < i_2\}$. As an example, consider D_{con-n} in figure 36. The only maximal node of the gap $(1, 3)$ of v_1 is v_{21}.

Intuitively, in a well-nested constituency structure $D_{con} = (V, E, r)$, all gaps are filled from "above". That means that all gap fillers (i. e., the parent nodes of all maximal nodes) of all gaps (i_1, i_2) in the yield of a $v \in V$ immediately dominate v itself. As an example, compare the well-nested structure D_{con-g} (figure 35) with the ill-nested structure D_{con-n} (figure 36): In the first, v_0 is the maximal node of the gap of v_1 and v_0 directly dominates v_1, while in D_{con-n}, v_2 is the maximal node of the gap of v_1 and v_2 does not stand in a dominance relation with v_1. We formalize this intuition in lemma 5.8.

LEMMA 5.8. Let $D_{con} = (V, E, r)$ be a well-nested constituency structure, $v \in V$ a node with gap degree $\geqslant 1$, (i_1, i_2) a gap in $\pi(v)$ and v_{max} a maximal node of (i_1, i_2). There is no node v' with $\langle v', v_{max} \rangle \in E$ and $\langle v', v \rangle \notin E^+$.

Proof. By contradiction. Assume there is a node v' with $\langle v', v_{max} \rangle \in E$ and $\langle v', v \rangle \notin E^+$. According to the definition of maximal nodes, there must be a $k_i \in \pi(v')$ with $k_i < i_1$ or $k_i > i_2$ and a k_j with $i_1 < k_j < i_2$. That means that it is either the case that $k_i < i_1 < k_j < i_2$ or $i_1 < k_j < i_2 < k_i$, both of which contradict the definition of well-nestedness. □

Ill-nested constituency structures are thus constituency structures in which some gap is filled from a direction other than "above", in

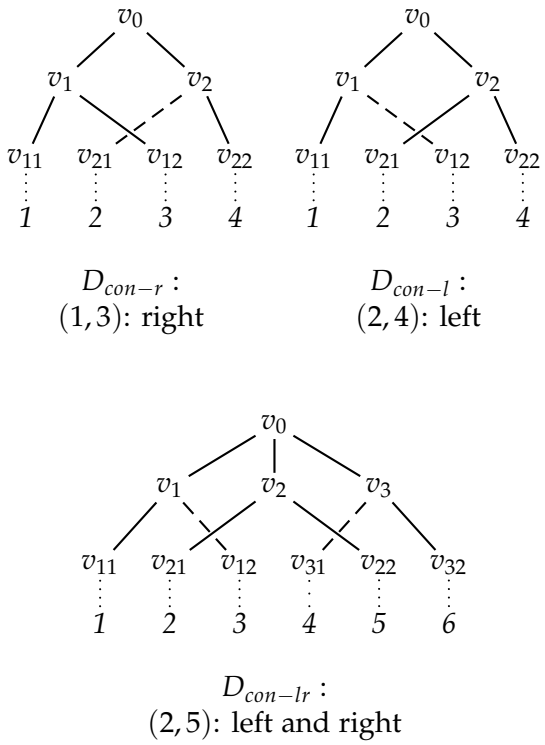


Figure 37: Gap filler positions

other words, in which the gap filler does not dominate the node with the gap. Gaps in such structures can intuitively be filled from the left, from the right, or from both sides. Note that if a gap is filled from the left or the right and additionally from above, we are still dealing with an ill-nested structure (cf. lemma 5.8).

DEFINITION 5.9 (Gap filler locations). Let $D_{con} = (V, E, r)$ be an ill-nested constituency structure, $v \in V$ a node with gap degree $\geqslant 1$, (i_1, i_2) a gap in $\pi(v)$, v_{max} a maximal node of (i_1, i_2), and $v' \in V$ a node with $\langle v', v_{max} \rangle \in E$ and $\langle v', v \rangle \notin E^+$. If $min(\pi(v')) < i_1$ then we say that (i_1, i_2) is *filled from the left*. If $max(\pi(v')) > i_2$ then we say that (i_1, i_2) is *filled from the right*.

As an example, figure 37 shows three trees with different gap filler locations. In D_{con-lr}, the gap is filled from both sides with material coming from two different nodes. Note that it is also possible that a single node fills a gap from both the left and the right.

5.2.2 *Measures for Extracted Grammars*

The gap degree and ill-nestedness of syntactic structures determine certain properties of SRCGs extracted from them. I present an algorithm which extracts an SRCG from a treebank by interpreting all syntactic structures in the treebank as SRCG derivations. With respect to dependency structures, the algorithm does exactly what Kuhlmann and Satta's (2009) extraction algorithm does.

Grammar Extraction

Recall the definition of treebanks (def. 2.19, p. 25), syntactic structures (def. 2.12, p. 22), yield and yield blocks (defs. 2.14 and 2.15, p. 24).

Let Υ be a treebank of length $m \in \mathbb{N}$. For all

$$(i, (V_i, E_i, r_i), \langle w_{i,1}, p_{i,1}\rangle \cdots \langle w_{i,n_i}, p_{i,n_i}\rangle) \in \Upsilon,$$

$1 \leqslant i \leqslant m$, $n_i \in \mathbb{N}$, do the following.

Create set of variables $\mathbb{X} = \{X_1, \ldots, X_{n_i}\}$. Then for all $v_0, v_1, \ldots, v_k \in V_i$ with $k = f_{out}(v_0)$ and $\langle v_0, v_j\rangle \in E_i$, $1 \leqslant j \leqslant k$, do the following.

If $f_{out}(v_0) = 0$, then add the *lexical clause* $p_{i,\Lambda_{V_i}(v_0)}(w_{i,\Lambda_{V_i}(v_0)}) \to \varepsilon$ to the grammar.

Otherwise, create a clause $\psi_0 \to \psi_1 \ldots \psi_k$ and determine the predicate labels for all ψ_r, $0 \leqslant r \leqslant k$ as follows. If (V_i, E_i, r_i) is a dependency structure, label ψ_r with $\Lambda_{E_i}(\langle h, v_r\rangle)$, where $h \in V_i$. Otherwise, label ψ_r with $\Lambda_{V_i}(v_r)$. Finally, append $dim(v_r)$ to the label of ψ_r. Now determine the predicate arguments of ψ_r. For all ψ_r, the i_rth argument, $1 \leqslant i_r \leqslant dim(v_r)$, is the string resulting from the concatenation of all $X_s \in \mathbb{X}$ such that $s \in v_r^{(i_r)}$,[1] and it holds for all $p, q \in v_r^{(i_r)}$ that $p < q$ iff X_p precedes X_q.

In all arguments of all ψ_j, if this string is of length $\geqslant 2$, then there and in ψ_0 (where the same string must occur), it is shortened to length 1 by replacing it with its first symbol (i. e., first variable). If (V_i, E_i, r_i) is a dependency structure and $\Lambda_{V_i}(v_0) = j$, then we exchange the variable X_j in the arguments of ψ_0 with $w_{i,\Lambda_{V_i}(v_0)}$.[2]

1 Recall that $v_r^{(i_r)}$ denotes the i_rth yield block of v_r.

2 This is what Kuhlmann and Satta (2009) do. Later, for parsing, we do not replace the variable. Instead, just as for constituency structures we add a lexical clause $p_{i,\Lambda_{V_i}(v_0)}(w_{i,\Lambda_{V_i}(v_0)}) \to \varepsilon$ to the grammar. The other addition to Kuhlmann and

$$
\begin{array}{rcl}
S_1(X_1X_2X_3X_4) & \rightarrow & VP_2(X_1,X_4)\ VAFIN_1(X_2)\ PPER_1(X_3) \\
VP_2(X_1X_2X_3X_4) & \rightarrow & ADV_1(X_1)\ VVPP_1(X_2)\ PPER_1(X_3)\ ADV_1(X_4)
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
ADV_1(\text{Selbst}) & \rightarrow & \varepsilon & VVPP_1(\text{besucht}) & \rightarrow & \varepsilon \\
VAFIN_1(\text{hat}) & \rightarrow & \varepsilon & PPER_1(\text{er}) & \rightarrow & \varepsilon \\
PPER_1(\text{ihn}) & \rightarrow & \varepsilon & ADV_1(\text{nie}) & \rightarrow & \varepsilon
\end{array}
$$

$$
\begin{array}{rcl}
aux_2(X_1\text{besucht}, X_5) & \rightarrow & adv_1(X_1)\ obja_1(X_5) \\
root_1(X_1\text{hat}X_4X_5X_6) & \rightarrow & aux_2(X_1,X_5)\ subj_1(X_4)\ adv_1(X_5)
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
adv_1(\text{Selbst}) & \rightarrow & \varepsilon & subj_1(\text{er}) & \rightarrow & \varepsilon \\
obja_1(\text{ihn}) & \rightarrow & \varepsilon & adv_1(\text{nie}) & \rightarrow & \varepsilon
\end{array}
$$

Figure 38: Extracted Simple RCGs

Finally, add the clause to the grammar.

Note that due to the nature of the treebanks, the extraction algorithm produces only ε-free ordered SRCGs. Furthermore, the presence of terminals in predicate arguments is limited to predicates of lexical clauses.[3]

As an example, figure 38 shows the two grammars extracted from the constituency and dependency structures in figure 34.

Properties of Extracted Grammars

The properties in lemma 5.10 are immediately obvious.

LEMMA 5.10 (Grammar Properties). Let G be a simple k-RCG of rank m. A syntactic structure D derived from G

1. has at most gap degree $g-1$ and
2. does not have nodes with more than m children.

Assuming the relation between simple RCGs and syntactic structures given by the grammar extraction algorithm, we can also draw

Satta's work we make for parsing is that we add a separate top clause which expands to all root clauses that have been extracted. Otherwise we would not be able to produce a connected graph. See p. 215.

3 This holds for constituency treebanks only, and for the version of the dependency extraction algorithm we use for parsing, cf. previous footnote.

conclusions from the kind of clauses in a simple RCG G on the well-nestedness of its derivations. Lemma 5.11 relates the interleaving of the variables in the arguments of clauses to the conditions given by the definition of well-nestedness. Recall the definition of argument numbering in RCG clauses (def. 2.46, p. 43). For convenience, we introduce the function $\eta : P \times \mathbb{N} \to \mathfrak{P}_{fin}(\mathbb{N})$, which, given an SRCG (N, T, V, P, S), returns for a clause $\psi_0 \to \psi_1 \cdots \psi_m \in P$ the numbers of the variables used in the arguments of ψ_i, $1 \leqslant i \leqslant m$, given the argument numbering.

LEMMA 5.11 (Ill-nestedness and Grammars). Let $G = (N, T, V, P, S)$ be an ordered simple RCG. If it produces k-ill-nested derivations with $k \geqslant 1$, then there is a clause $c \in P$, $c = \psi_0 \to \psi_1 \cdots \psi_m$ with ψ_i, ψ_j, $1 \leqslant i < j \leqslant m$ such that

1. $i_1, i_2 \in \eta(p, i)$ and $j_1, j_2 \in \eta(p, j)$, and
2. $i_1 < j_1 < i_2 < j_2$.

Proof. By contradiction. Assume there is no such clause and G produces ill-nested derivations. Due to the definition of well-nestedness, there must be a derivation tree (V, E, r) of G with some $\nu_1, \nu_2 \in V$ with disjoint yields such that for some $i_1, i_2 \in \pi(\nu_1)$ and $j_1, j_2 \in \pi(\nu_2)$ it holds that $i_1 < j_1 < i_2 < j_2$. The fact that $\pi(\nu_1)$ and $\pi(\nu_2)$ are disjoint entails that ν_1 and ν_2 do not dominate each other. Furthermore, it entails that the yield of their least common ancestor ν_{lca} must contain both $\pi(\nu_1)$ and $\pi(\nu_2)$ and that for all nodes ν' with $\langle \nu_{lca}, \nu' \rangle \in E^+$, it holds that $\pi(\nu') \cap \pi(\nu_1) = \emptyset$ or $\pi(\nu') \cap \pi(\nu_2) = \emptyset$. Assume ν_{lca} and its children have been generated by a clause $\psi_0 \to \psi_1 \cdots \psi_m$. Due to the aforementioned condition on the yields of all nodes dominated by ν_{lca} and the way predicate arguments are built by the grammar extraction algorithm, there must be at least two predicates ψ_i, ψ_j, $1 \leqslant i < j \leqslant m$, for which it holds that $i_1, i_2 \in \eta(p, i)$ and $j_1, j_2 \in \eta(p, j)$ with $i_1 < j_1 < i_2 < j_2$. This contradicts our initial assumption. □

Note that given a simple RCG which produces k-ill-nested structures with $k \geqslant 1$, it is not possible to determine the exact k without inspecting all possible derivations, i. e., the k can not be determined by properties of the clauses of the grammar alone. This is due to the fact that k-ill-nestedness is not clause-local.

One can also build well-nestedness directly into the definition of the grammar. This has first been done by Kanazawa (2009b) for Multiple

Context-Free Grammar (MCFG). The well-nestedness requirement effectively reduces the expressivity of the formalism.

LEMMA 5.12. Multiple Context-Free Languages (MCFL) properly contain well-nested MCFLs (Kanazawa and Salvati, 2010).

Here, we define Wellnested Ordered Simple Range Concatenation Grammar. Note that the ordering requirement does not change the expressivity of SRCG (see lemma 2.55, p. 47).

DEFINITION 5.13 (Wellnested Ordered Simple Range Concatenation Grammar (WOSRCG)). Let $G = (N, T, V, P, S)$ be a Ordered Simple Range Concatenation Grammar (OSRCG). Assume for all $c \in P$ that c only contains variables $\{X_1, \ldots, X_n\}$ and in all predicates of c, X_i precedes X_j iff $i < j$ for $1 \leqslant i < j \leqslant n$.

1. A clause $c = \psi_0 \rightarrow \psi_1 \cdots \psi_m \in P$ is *ill-nested* if there are two predicates ψ_i, ψ_j, $1 \leqslant i < j \leqslant m$ which are such that
   a) ψ_i contains two variables X_{i_1}, X_{i_3},
   b) ψ_j contains two variables X_{i_2}, X_{i_4}, and
   c) $i_1 < i_2 < i_3 < i_4$.

   If c is not ill-nested, then it is *well-nested*.
2. G is *well-nested* (a WOSRCG) iff all $c \in P$ are well-nested.

5.2.3 *Empirical Investigation*

We now verify our new measures in an empirical investigation.

Non-Projective Dependency Structures

The empirical relevance of gap degree and well-nestedness has been shown for dependency treebanks in works such as Kuhlmann and Nivre (2006) on the basis of the Prague Dependency Treebank (PDT) (Hajič et al., 2000) and the Danish Dependency Treebank (DDT) (Kromann, 2003). We extended the previous investigations on two other treebanks, namely the dependency versions of the NeGra and TIGER, the TIGER dependency treebank (TIGER-Dep) and the NeGra dependency treebank (NeGra-Dep). Both have been built with the dependency converter for TIGER-style trees from Daum et al. (2004). We are aware of the fact that all dependency conversion methods introduce

	DANISH DEP. TREEBANK		PRAGUE DEP. TREEBANK	
number of sent.	4,393		73,088	
av. sent. length	18		17	
gap degree 0	3,732	84.95%	56,168	76.85%
gap degree 1	654	14.89%	16,608	22.72%
gap degree 2	7	0.16%	307	0.42%
gap degree 3	–	–	4	0.01%
gap degree 4	–	–	1	$< 0.01\%$
gap degree $\geqslant 5$	–	–	–	–
well-nested	4,388	99.89%	73,010	99.89%

Table 5: Gaps, well-nestedness (DDT, PDT)

undesired noise, but we choose this well-established method rather than using other German dependency data sets like the CoNLL-X TIGER data (used by Havelka (2007)) or the very small TIGER dependency treebank (TIGER-DP) of Forst et al. (2004) due to our desire of obtaining comparable dependency structures for both constituency treebanks.

Since punctuation is generally not attached to the trees and therefore not part of the annotation, we remove it in a preprocessing step from all trees in both treebanks prior to the conversion. This leads to the exclusion of a handful of sentences from TIGER and NeGra, since they consist only of punctuation. Table 5 contains the gap degree figures and the ratios of well-nestedness of the PDT and the DDT, borrowed from Kuhlmann and Nivre (2006). Table 6 contains our findings for NeGra-Dep and TIGER-Dep.

The gap degree figures of NeGra-Dep and TIGER-Dep lie in the same range as the figures of DDT and PDT. A closer look at the dependency annotations of TIGER-Dep and NeGra-Dep reveals that the most common causes for a high gap-degree (greater than two) are enumerations, appositions and parenthetical constructions. The constituency counterparts of these sentences have no or very few gaps.

Well-nestedness covers almost all dependency structures found in treebanks (Kuhlmann, 2007, p. 62). This is confirmed by our quantitative findings in the German treebanks. With the exception of a sin-

	NEGRA-DEP		TIGER-DEP	
number of sent.	20,597		40,013	
av. sent. length	15		16	
gap degree 0	16,695	81.06%	32,079	80.17%
gap degree 1	3,662	17.78%	7,466	18.66%
gap degree 2	225	1.09%	438	1.09%
gap degree 3	12	0.05%	22	0.05%
gap degree 4	2	0.01%	4	0.01%
gap degree $\geqslant 5$	1	$< 0.01\%$	4	0.01%
well-nested	20,472	99.39%	39,750	99.34%
1-ill-nested	124	0.60%	263	0.66%
2-ill-nested	1	$< 0.01\%$	–	–

Table 6: Gaps, well-nestedness (NeGra-Dep, TIGER-Dep)

gle 2-ill-nested dependency structure[4] in NeGra-Dep, the structures which do not adhere to the well-nestedness constraint are all 1-ill-nested. Note that 1-ill-nestedness is a stronger constraint than non-well-nestedness.[5] A closer linguistic inspection of the ill-nested dependency structures in Negra-Dep and TIGER-Dep shows that the most common reason for ill-nestedness is not erroneous annotation, but linguistically acceptable analyses of extraposition phenomena. Even though well-nestedness is a very useful constraint which can be exploited for efficient parsing (Gómez-Rodríguez et al., 2010), ill-nested structures should therefore not be discarded. Figure 39 shows a typical ill-nested dependency analysis of (16), a sentence of NeGra.

(16) Im Grenzverkehr hat sich ein Modus eingespielt, der
In border traffic has itself a mode equipoised, which
...zulässt
...admits
"Border traffic has ended up in a mode which admits for ..."

4 The edge which is responsible for the 2-ill-nestedness cannot be linguistically motivated and is an artifact introduced by the conversion procedure.

5 The fact that there are no k-ill-nested structures with $k \geqslant 2$ can be explained with the fact that they would require a kind of simultaneous right and leftward extraposition, something which German does not exhibit as a feature.

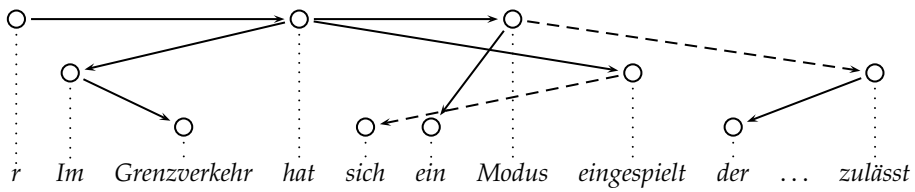


Figure 39: Ill-nested dependency structure in Negra-Dep

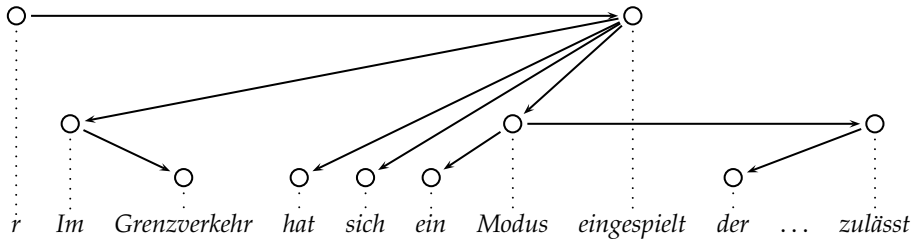


Figure 40: Well-nested alternative analysis to fig. 39

The edges relevant for the ill-nestedness are dashed. In this sentence, the subject noun *ein Modus* dominates a non-adjacent, extraposed relative clause, while being surrounded by a disjoint subtree, namely the non-finite main verb *eingespielt* and its dependent.

The annotation can be related to linguistic generalizations on dependency structures discussed in the literature (particularly on German):

1. The subject depends on the finite verb (Kunze, 1975, p. 110; Hudson, 1984, pp. 83).

2. The non-finite verb depends on the finite verb and governs its objects and modifying expressions (Engel, 1988, p. 189). In our treebanks, this is only true for objects and modifying expressions outside the *Vorfeld* since *Vorfeld* material is systematically attached to the finite verb by the conversion procedure.

3. Extraposed material is dependent on its antecedent (Kunze, 1975, pp. 130; Hudson, 1984, pp. 101).

By changing certain assumptions about how dependency structures are built, one can get rid of certain cases of ill-nestedness. An alternative well-nested structure for (16) is shown in figure 40. However, there are good linguistic arguments for the fact that ill-nestedness in

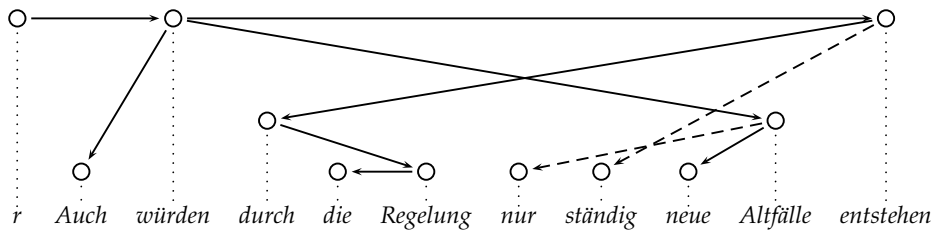


Figure 41: Ill-nested dependency structure in TIGER-Dep

certain other structures cannot be avoided (Chen-Main and Joshi, 2010, 2012).

In the remaining sentences, there are two phenomena which give rise to ill-nestedness, namely coordinated structures and discontinuous subjects. Figure 41 shows the annotation of (17), an example from TIGER-Dep for a discontinuous subject. Again, the relevant edges are dashed.

(17) Auch würden durch die Regelung nur ständig neue
Also would through the regulation only always new
Altfälle entstehen.
cases emerge
"Another effect of the regulation would be constantly emerging new cases."

The ill-nested annotation of the coordination cases is often disputable. This can be explained with the lack of a general linguistic theory of coordination (Lobin, 1993). The situation for discontinuous subjects is more clear, since one can argue that the components of the discontinuous subject distinguish themselves from the material in the gap by making up a semantic unit.

To sum up, ill-nested dependency annotation in NeGra-Dep and TIGER-Dep can generally be linguistically justified and is not only the result of an accidental interplay of annotation principles. An accurate linguistic survey of (k)-ill-nestedness of structures in DDT and PDT has not been presented in the literature and is left for future work.

Discontinuous Constituency Structures

We also investigate gap degree and well-nestedness of constituency treebanks in order to verify if both measures are as informative for

	NEGRA		TIGER	
gap degree 0	14,904	72.36%	28,832	72.06%
gap degree 1	5,002	24.29%	9,902	24.75%
gap degree 2	682	3.31%	1,264	3.16%
gap degree 3	9	0.04%	15	0.04%
well-nested	20,339	98.75%	39,573	98.90%
1-ill-nested	258	1.25%	440	1.10%
2-ill-nested	–	–	–	–

Table 7: Gaps, well-nestedness (NeGra)

constituency structures as they are for dependency structures. We conduct our study on the constituency versions of the treebanks in the previous section, using exactly the same set of sentences (with removed punctuation). The results are summarized in table 7.

The constituency gap degree figure of both German treebanks again lie close together. The number of constituency structures with gap degree greater or equal than three is considerably lower than the corresponding number of dependency structures. The reason is that the phenomena which cause a high gap degree in dependency structures (enumerations and appositions) generally receive a constituency structure without gaps. The most frequent reasons for gaps in constituency structures are parenthetical constructions, as well as finite verbs, subjects and negative markers, which are generally annotated as immediate constituents of the highest S node and therefore may cause gaps in the VP yield.

Table 7 shows that the ratio of ill-nested structures in constituency data is comparable to the ratio in dependency data. This suggests that ill-nestedness has a comparable explanatory value as a constraining feature for constituency structures. The only degree of ill-nestedness that can be observed is 1-ill-nestedness. A linguistic inspection of the ill-nested constituency structures shows that most of them are ill-nested due to the interplay of several annotation principles. Again, most of the ill-nested constituency structures in TIGER and NeGra arise from extraposition phenomena. Furthermore we can also find cases of discontinuous subjects annotated with ill-nested structures.

Other than for the dependency structures, coordination is no trigger for ill-nestedness in the constituency data.

As an example for ill-nested constituency structure, see the embedded sentence (18) and its tree annotation in figure 42.

(18) ... ob auf deren Gelände der Typ von Abstellanlage
... whether on their premises the type of parking facility
gebaut werden könne, der ...
built be could, which ...
"whether on their premises precisely the type of parking facility could be built, which ..."

The two overlapping, disjunctive constituents are the lower VP, and the NP with its extraposed relative clause.

The following annotation principles seem to be respected throughout:

1. the subject is an immediate constituency of the sentence;
2. the finite verb is another immediate constituency (and the head) of the sentence;
3. the non-finite verb is the head of another immediate constituency that also includes objects and modifying expressions;
4. extraposed material is included in the antecedent constituency.

We will not argue in favor or against these annotation principles from a linguistic point of view. A mapping to common linguistic theories is highly non-trivial, since very different means of expressing constituency relations and a variety of shapes of constituency structures would have to be taken into account. On the other hand, the similarity to the above stated annotation principles for dependency structures is striking.

Ill-nestedness does not affect the same set of structures across treebank variants, i. e., the ill-nested dependency structures are no subset of the ill-nested constituency structures and vice versa. We leave an exhaustive investigation of the differences for future work.

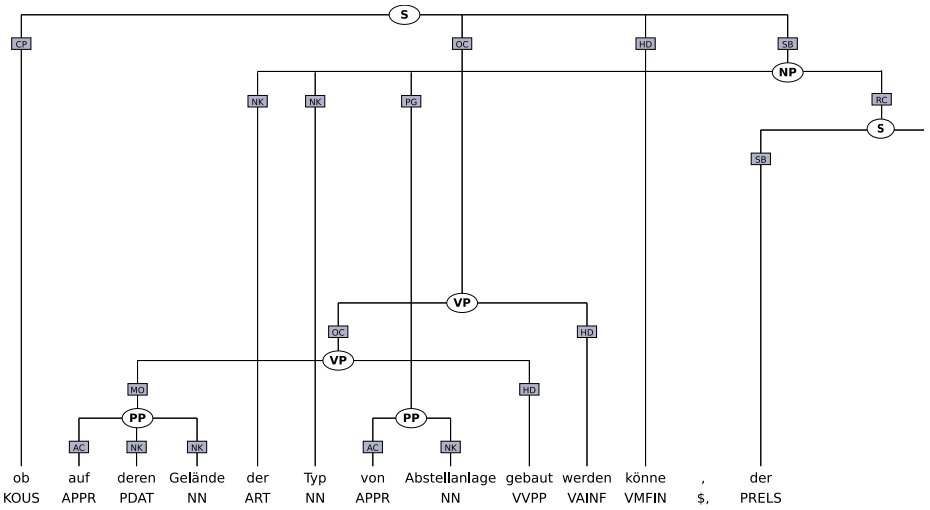


Figure 42: Ill-nested constituency structure in NeGra

5.3 SYNCHRONOUS REWRITING

Simple Range Concatenation Grammar allows for *synchronous rewriting*. We speak of synchronous rewriting when two or more context-free derivation processes are instantiated in a synchronous way. Unlike SRCG, Discontinuous Phrase Structure Grammar (DPSG), which has also been proposed for modeling discontinuities, does not allow for synchronous rewriting because the different discontinuous parts of the yield of a non-terminal are treated locally, i.e., their derivations are independent from each other (cf. p. 49). So far, synchronous rewriting has not been empirically motivated by linguistic data from treebanks. Here, we fill this gap by investigating the existence of structures indicating synchronous rewriting in treebanks with discontinuous annotations. The question of whether we can find evidence for synchronous rewriting is important for the choice of grammar formalisms to model discontinuities and, furthermore, it has consequences for the complexity of parsing. In fact, parsing with synchronous formalisms can be carried out in time polynomial in the length of the input string, with a polynomial degree depending on the maximum number of synchronous branches one can find in derivations (Seki et al., 1991).

In this section, we characterize synchronous rewriting as a property of trees with crossing branches and in an empirical evaluation and we confirm that constituency treebanks do contain recursive syn-

chronous rewriting which can be linguistically motivated. Furthermore, we show how this characterization transfers to the simple RCGs describing these trees.

5.3.1 *Treebank Trees with Synchronous Rewriting*

By *synchronous rewriting* we indicate the synchronous instantiation of two or more context-free derivation processes. As an example, consider the language $\mathcal{L} = \{a^n b^n c^n d^n \mid n \geqslant 1\}$. Each of the two halves of some $w \in \mathcal{L}$ can be obtained through a stand-alone context-free derivation, but for w to be in $\mathcal{L}$ the two derivations must be synchronized somehow. For certain tasks, synchronous rewriting is a desired property for a formalism. In machine translation, e.g., synchronous rewriting is extensively used to model the synchronous dependence between the source and target languages (Chiang, 2007). The question we are concerned with here is whether we can find instances of recursive synchronous rewriting in treebanks that show discontinuous phrases.

We make the assumption that, if the annotation of a treebank allows to express synchronous rewriting, then all cases of synchronous rewriting are present in the annotation. This means that, on the one hand, there are no cases of synchronous rewriting that the annotator "forgot" to encode. Therefore unrelated cases of parallel iterations in different parts of a tree are taken to be truly unrelated. On the other hand, if synchronous rewriting is annotated explicitly, then we take it to be a case of true synchronous rewriting, even if, based on the string, it would be possible to find an analysis that does not require synchronous rewriting. This assumption allows us to concentrate only on explicit cases of synchronous rewriting.

We concentrate on the NeGra and TIGER treebanks. In the trees of those treebanks, synchronous rewriting amounts to cases where different components of a non-terminal category develop in parallel. In particular, we search for cases where the parallelism can be iterated. An example is the relative clause in (19), found in TIGER. Figure 43 gives the annotation. As can be seen in the annotation, we have two VP nodes, each of which has a discontinuous span consisting of two parts. The two parts are separated by lexical material not belonging to the VPs. The two components of the second VP (*Pop-Idol* and *werden*) are included in the two components of the first, higher, VP (*genau-*

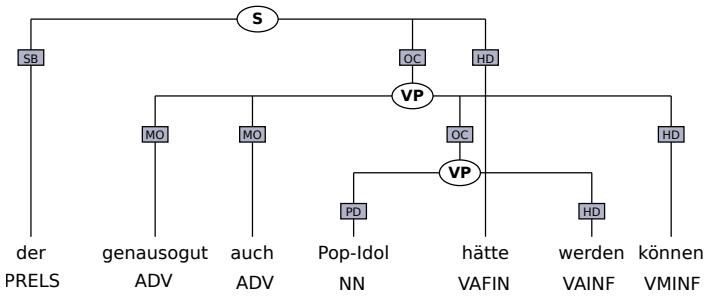


Figure 43: Recursive synchronous rewriting

	NEGRA	TIGER
number of trees	20,597	40,013
total num. of RSRS in all trees	600	1476
av. RSRS length in all trees	2.12	2.13
max. RSRS length in all trees	4	5

Table 8: Synchronous rewriting in treebanks

sogut auch Pop-Idol and *werden können*). In other words, the two VP components are rewritten in parallel and contain again two smaller VP components.

(19) . . . der genausogut auch Pop-Idol hätte werden können
. . . who as well also pop-star have become could
"who as well also could have become a pop-star"

Trees showing *recursive synchronous rewriting* can be characterized as follows. We have a non-terminal node n_1 with label A whose yield has a gap. n_1 dominates another node n_2 with label A such that for some $i \neq j$, the ith component of the yield of n_2 is contained in the ith component of the yield of n_1 and similar for the jth component. We call the path from n_1 to n_2 a *recursive synchronous rewriting segment (RSRS)*.

Table 8 shows the results obtained from searching for recursive synchronous rewriting in the German TIGER and NeGra treebanks. As in the last section, punctuation has been removed.

(19) shows that we find instances of recursive synchronous rewriting where each of the rewriting steps adds something to both of the parallel components. (19) was not an isolated case.

The annotation of (19) in figure 43 could be turned into a context-free structure if the lowest node dominating the material in the gap while not dominating the synchronous rewriting nodes (here VAFIN) is attached lower, namely below the lower VP node. (Note however that there is good linguistic motivation for attaching it high.) Besides such cases, we even encountered cases where the discontinuity cannot be removed this way. An example is (18) (resp. figure 42) where we have a gap containing an NP such that the lowest node dominating this NP while not dominating the synchronous rewriting nodes has a daughter to the right of the yields of the synchronous rewriting nodes, namely the extraposed relative clause. This structure is of the type $a^n c b^n d$, where a and b depend on each other in a left-to-right order and can be nested, and c and d also depend on each other and must be generated together. This is a structure that requires synchronous rewriting, even on the basis of the string language. Note that the nesting of VPs can be iterated, as can be seen in (20), resp. in figure 44.

(20) . . . ob auf deren Gelände der Typ von Abstellanlage
. . . whether on their premises the type of parking facility
eigentlich hätte schon gebaut werden sollen, der . . .
actually had already built be should, which . . .
"whether on their premises precisely the type of parking facility should actually already have been built, which . . . "

As a conclusion from these empirical results, we state that to account for the data we can find in treebanks with discontinuities, i.e., with crossing branches, we need a formalism that can express synchronous rewriting.

5.3.2 *Synchronous Rewriting in Extracted Grammars*

We also check SRCGs extracted from TIGER and NeGra for the possibility to generate recursive synchronous rewriting. For this purpose, we use the extraction algorithm from section 5.2.2. For the tree in figure 43, the algorithm produces for instance the following clauses:

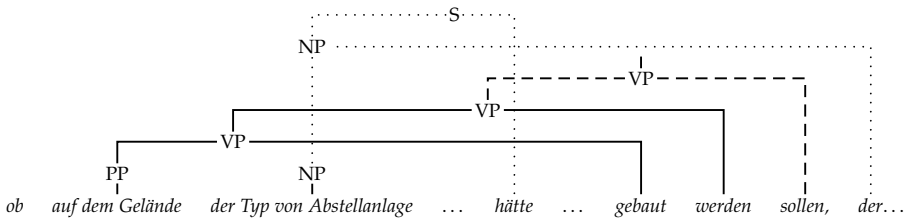


Figure 44: Iterable synchronous rewriting

$\text{PRELS(der)} \rightarrow \varepsilon$

$\text{ADV(genausogut)} \rightarrow \varepsilon$

...

$\text{S}(X_1X_2X_3X_4) \rightarrow \text{PRELS}(X_1)\text{VP}_2(X_1,X_4)\ \text{VAFIN}(X_3)$

$\text{VP}_2(X_1X_2X_3,X_4X_5) \rightarrow \text{ADV}(X_1)\ \text{ADV}(X_2)\text{VP}_2(X_3,X_4)\ \text{VMINF}(X_5)$

$\text{VP}_2(X_1,X_2) \rightarrow \text{NN}(X_1)\ \text{VAINF}(X_2)$

We distinguish different usages of the same category depending on their numbers of yield components. E.g., we distinguish non-terminals VP_1, VP_2, ... depending on the arity of the VP. We define $cat(A)$ for $A \in N$ as the category of A, independent from its arity. For instance, $cat(\text{VP}_2) =$VP.

In terms of simple RCG, synchronous rewriting means that in a single clause distinct variables occurring in two different arguments of the LHS predicate are passed to two different arguments of the same RHS predicate. We call this *recursive* if, by a sequence of synchronous rewriting steps, we can reach the same two arguments of the same predicate again. Derivations using such cycles of synchronous rewriting lead exactly to the recursive synchronous rewriting trees characterized in section 5.3.1. In the following, we check to which extent the extracted simple RCG allows for such cycles.

In order to detect synchronous rewriting in a simple k-RCG G, we build a labeled directed graph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G}, l)$ from the grammar with $V_\mathcal{G}$ a set of nodes, $E_\mathcal{G}$ a set of arcs and $\Lambda : V_\mathcal{G} \rightarrow N' \times \{0,\ldots,k\} \times \{0,\ldots,k\}$ where $N' = \{cat(A) \mid A \in N\}$ a labeling function. $\mathcal{G}$ is constructed as follows. For each clause $A_0(\vec{\alpha}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m}) \in P$ we consider all pairs of variables X_s, X_t for which the following conditions hold:

1. X_s and X_t occur in different arguments i and j of A_0, $1 \leqslant i < j \leqslant dim(A_0)$; and

2. X_s and X_t occur in different arguments q and r of the same occurrence of predicate A_p in the RHS, $1 \leqslant q < r \leqslant dim(A_p)$ and $1 \leqslant p \leqslant m$.

For each of these pairs, two nodes are created and labeled $[cat(A_0), i, j]$ and $[cat(A_p), q, r]$, respectively. They are added to $V_\mathcal{G}$ (if they do not yet exist, otherwise the already existing nodes are used) and a directed arc from the first node to the second node is added to $E_\mathcal{G}$. The intuition is that an arc in $\mathcal{G}$ represents one or more clauses from the grammar in which a gap between two variables in the LHS predicate is transferred to the same RHS predicate.

To detect recursive synchronous rewriting, we then need to discover all elementary cycles in $\mathcal{G}$, i.e., all cycles in which no vertex appears twice except the first and the last node. In order to accomplish this task efficiently, we exploit the algorithm presented in Johnson (1975). On a grammar extracted from NeGra, the algorithm yields a graph with 28 nodes containing 206,403 cycles of an average length of 12.86 and a maximal length of 28.

Our algorithm runs in asymptotical polynomial time, with a polynomial degree that is independent of k. Johnson's algorithm runs in $\mathcal{O}((|V_\mathcal{G}| + |E_\mathcal{G}|)(c + 1))$, where c is the number of cycles in $\mathcal{G}$. Note that the algorithm detects synchronous rewriting with at least two context-free derivations involved, but not whether there are more than two. The second task would be more complex, resulting in a running time of polynomial degree depending on k. Our algorithm can be easily adapted to such a task.

5.4 RELATED WORK

5.4.1 *Dependency Grammar*

Linguistic Modeling

Current dependency frameworks do not require projectivity, as mentioned before. Interestingly, there is an intersection with constituency-based work in the way non-projectivity is modeled. Some dependency frameworks incorporate a separation of dominance and linear precedence in the style of the framework of Immediate Dominance/Linear Precedence (ID/LP). In fact, the separation is already present in Tes-

nière (1959), and can also be found, e. g., in Debusmann et al. (2004) and (partly) in Gerdes and Kahane (2001).

A particularly interesting, very ambitious work which brings together statistical natural language processing and psycholinguistics on the basis of DG is Buch-Kromann's Discontinuous Grammar (Buch-Kromann, 2009). The Danish Dependency Treebank (Kromann, 2003) is based on this framework.

Quantifying Non-Projectivity

Some of the literature on characterizations of non-projectivity of dependency structures has been left out so far; either because it did not have as much impact as the literature discussed earlier, or because the respective characterizations of non-projectivity cannot be straightforwardly transferred to constituency structures.

REGULAR DEPENDENCY LANGUAGES Offering a formal relation between dependency structures and MCS formalisms is is the idea behind *Regular Dependency Languages* (Kuhlmann and Möhl, 2007a,b; Kuhlmann, 2007). They are a proper, mildly context-sensitive extension of Context-Free Languages (CFLs) and have been introduced as a characterization of the string languages introduced by certain types of dependency structures.

EDGE DEGREE The *degree of an edge* (Nivre, 2006; Kuhlmann and Nivre, 2006) is the number of components not dominated by the head of the edge which intervene in its span, the span being all string positions between the two endpoints of the edge. The edge degree of a dependency structure is the maximum edge degree of any of its edges.

LEVEL TYPES *Level types* and *level signatures* (Havelka, 2007) cannot be straight-forwardly transferred to constituency structures, because they are formulated in terms of path lengths in dependency structures, instead of node yields.

PLANARITY The *planarity* constraint (Sleator and Temperley, 1993; Havelka, 2007) prohibits crossing edges in a dependency structure. The same constraint is also known as *weak non-projectivity* (Dikovsky and Modina, 2000). It can be defined as a stronger version of well-

nestedness, in which one does not require the considered yields to be disjoint.

MULTI-PLANARITY Yli-Jyrä (2003) extends the planarity constraint to *multi-planarity with a bounded number of planes*. He does not consider only dependency trees, but dependency graphs, i. e., he discards acyclicity and connectedness. Roughly, the idea is that a dependency graph is m*-planar* if its set of edges can be partitioned into m sub-sets such that when considering only one of the sub-sets, we have a planar graph. Yli-Jyrä (2003) notes that in spite of being linguistically useful, from a formal language point of view, multiplanarity is a very loose constraint, and introduces additional restrictions on his concept.

PSEUDO-PROJECTIVITY With the motivation of allowing discontinuities while still allowing for polynomial parsing, Kahane et al. (1998) introduce *pseudo-projectivity*. Pseudo-projectivity relies on an operation called *lifting*, with which one can transform certain non-projective graphs into projective graphs. The transformation is reversible, because the information provided by the non-projective edges is not lost, but encoded in the node labels. The transformation resembles the algorithm for resolving the crossing branches in NeGra by Boyd (2007) (cf. p. 102).

5.4.2 *Constituency-Based Frameworks*

To my knowledge, there has not been any attempt of a quantification of discontinuity in constituencies before Maier and Lichte (2011). However, there is a very large body of linguistic literature on the modeling of discontinuous constituents within grammatical frameworks. I present a short overview, in which I follow Lichte (2012). He divides the literature in two parts.

The first part contains approaches that model discontinuity indirectly in the tradition of generative grammar. Roughly put, this happens through some kind of movement which generates a discontinuous structure out of a continuous one. This area is out of the scope of this thesis. Consult Wurmbrand (2001) for a detailed overview.

The second part contains frameworks which provide a direct representation of discontinuity. They include

- all those frameworks which relax the ordering of the right-hand sides of context-free rules, among others DPSG (cf. p. 49) and Suhre's (2000) Linear Specification Language (LSL);
- all frameworks which allow non-terminals to span more than a single yield block such as Linear Context-Free Rewriting System (LCFRS) and its equivalent formalisms; and
- the ID/LP variant of GPSG[6] (Shieber, 1984) (see above) and its extensions, as presented, e. g., by Zwicky (1986), Blevins (1990) and Daniels (2005); furthermore the HPSG-based work such as Reape (1994) and Richter and Sailer (1995).

There are also Tree-Adjoining Grammar (TAG) variants which allow for *derived* trees with crossing branches. Sarkar and Joshi (1996) build derived trees with crossing branches to account for cases of coordination with ellipsis. Chen-Main (2006) considers elements which take a different grammatical role depending on their linear position as being dominated by multiple parents. In her thesis on these *multi-dominance structures*, she follows up on Sarkar and Joshi's work. For a further excellent overview on TAG and discontinuity, see again Lichte (2012).

Note that this is only a tiny fraction of the literature in order to give some ideas, a complete overview is beyond the scope of this thesis.

5.4.3 *Synchronous Rewriting*

While to my knowledge, synchronous rewriting has never been explored in relation to treebanks, it is formally well-understood, and, as mentioned before, an important tool in certain NLP applications. A further exploration of this topic is beyond the scope of this thesis. The interested reader is referred to the literature. A good starting point are Rambow and Satta (1996, 1999).

5.5 CONCLUSION

This chapter was dedicated to a review of the treatment of discontinuity and non-projectivity in treebanks. We have presented a grammar

6 See also Uszkoreit (1986), who presents a Generalized Phrase Structure Grammar (GPSG), resp. ID/LP analysis of German.

extraction algorithm for both constituency and dependency treebanks showing discontinuities. We have formulated the measures of gap degree and well-nestedness which are known from DG such that they apply to both dependency structures and constituency structures. An empirical evaluation on treebanks has confirmed the expressivity of the measures for constituency structures. Furthermore we have confirmed the presence of synchronous rewriting in treebank trees and extracted grammars.

There are treebanks in languages other than German which provide both non-projective dependency annotation and discontinuous constituency annotation, such as the Bulgarian BulTreebank (Osenova and Simov, 2004) and the recently created Discontinuous Penn Treebank, a version of the PTB in which trace nodes and co-indexation have been converted to crossing branches (Evang, 2011). In future research, our investigations will be extended to these additional resources. This will allow us to investigate our suspicion that gap degree and well-nestedness are language-specific parameters.

6

DATA-DRIVEN PARSING BEYOND CONTEXT-FREE GRAMMAR

In this chapter, I present methods for statistical data-driven parsing using Simple Range Concatenation Grammar (SRCG) as an underlying formalism. Firstly, techniques for binarization and markovization are presented. Secondly, I present a CYK parser for Probabilistic Simple Range Concatenation Grammar (PSRCG), together with context-summary estimates for parse items which are used to speed up parsing. Eventually, I present related work and conclude this chapter.

Material in this and the following two chapters has been published previously in Maier (2010), Kallmeyer and Maier (2010), Maier and Kallmeyer (2010) and Kallmeyer and Maier (2012).

6.1 OBTAINING A PROBABILISTIC GRAMMAR

We use the grammar extraction algorithm presented on p. 137 to obtain SRCGs from dependency and constituency treebanks.[1] In order to allow the computation of a Maximum Likelihood Estimate, we count the occurrences of each clause. In all experiments, we parse sequences of Part-of-Speech (POS) tags and not words, i. e., we provide the parser with the gold POS tags. This means that we do not use the extracted lexical clauses and that in dependency grammar extraction, we do not put the lexical elements directly in the predicate arguments (consider the footnote on p. 137).

6.1.1 *Improving the Unbinarized Grammar*

The unbinarized grammar can be improved by modifying the treebank before grammar extraction, or just after it.

1 Treebank-specific preprocessing, e. g., for the treatment of punctuation, is described in chapters 7 and 8.

Grammar Annotation

Probabilistic Context-Free Grammar (PCFG) suffers from its strong independence assumptions. This comes from the fact that a single production only covers a subtree of height one in a derivation. PSRCG shares this problem with PCFG. Klein and Manning (2003b) (cf. p. 110) take advantage of the fact that certain symbols only occur in certain contexts. They find that annotating symbols with their context (*splitting* them) improves parsing results. This method is not specific to PCFG and we apply it to PSRCG. The corresponding experiments will be called SPLIT.[2]

Consider the NeGra treebank as an example. One possibility for splitting is to use the edge labels. The edge label of a node encodes its grammatical function. While even with PCFG, one runs into sparse data problems when discriminating all node labels by their edge labels (Kübler, 2005), using some of them can still be beneficial. E. g., one can use the RC edge label in order to distinguish relative clauses from regular sentences (both share the node label S). The annotations used for the experiments will be described in chapters 7 and 8.

Cutoff

Charniak (1996) finds that removing productions with a low frequency in a PCFG only has a minor effect on parsing results. Since with SRCG, we have to keep an eye on parsing efficiency, in the experiments, we will study this method (CUTOFF).

Note that removing clauses may render symbols useless. This happens when a clause with some left-hand side (LHS) predicate label A is removed which, after removing the clause, only occurs on the right-hand side (RHS) of other clauses. Since from these clauses, nothing will be derived, we apply the removal procedure recursively.

6.1.2 *Binarization*

The binarization of SRCG is similar to the transformation of a Context-Free Grammar (CFG) into Chomsky Normal Form (CNF). The result is an SRCG of rank 2. As in the CFG case, in the transformation, we

2 All methods and techniques we refer to in the discussion of the experiments are given names written in small capital letters.

introduce a non-terminal for each RHS longer than 2 and split the clause into two clauses, using this new intermediate non-terminal (we call it a *binarization non-terminal*). This is repeated until all RHS are of no greater length than 2.

For the presentation of the transformation algorithm, we need the notion of a *reduction* of a vector $\vec{\alpha} \in [(T \cup V)^*]^i$ by a vector $\vec{x} \in V^j$ where all variables in $\vec{x}$ occur in $\vec{\alpha}$. A reduction is, roughly, obtained by keeping all variables in $\vec{\alpha}$ that are not in $\vec{x}$. This is defined as follows.

DEFINITION 6.1 (Vector reduction). Let $\langle N, T, V, P, S\rangle$ be a SRCG, $\vec{\alpha} \in [(T \cup V)^*]^i$ and $\vec{x} \in V^j$ for some $i, j \in \mathbb{N}$. Let $w = \vec{\alpha}_1\$\ldots\$\vec{\alpha}_i$ be the string obtained from concatenating the components of $\vec{\alpha}$, separated by a new symbol $\$ \notin (V \cup T)$. Let w' be the image of w under a homomorphism h defined as follows: $h(a) = \$$ for all $a \in T$, $h(X) = \$$ for all $X \in \{\vec{x}_1, \ldots \vec{x}_j\}$ and $h(y) = y$ in all other cases. Let $y_1, \ldots y_m \in V^+$ such that $w' \in \$^* y_1 \$^+ y_2 \$^+ \ldots \$^+ y_m \$^*$. Then the vector $\langle y_1, \ldots y_m\rangle$ is the *reduction* of $\vec{\alpha}$ by $\vec{x}$.

For instance, $\langle aX_1, X_2, bX_3\rangle$ reduced with $\langle X_2\rangle$ yields $\langle X_1, X_3\rangle$ and $\langle aX_1X_2bX_3\rangle$ reduced with $\langle X_2\rangle$ yields $\langle X_1, X_3\rangle$ as well.

Let (N, T, V, P, S) be an SRCG
for all clauses $c = A(\vec{\alpha}) \to A_0(\vec{\alpha_0}) \ldots A_m(\vec{\alpha_m})$ in P with $m > 1$ **do**
  remove c from P
  $R = \emptyset$
  pick new non-terminals $C_1, \ldots, C_{m-1}$
  add the clause $A(\vec{\alpha}) \to A_0(\vec{\alpha_0})C_1(\vec{\gamma_1})$ to R where $\vec{\gamma_1}$ is obtained by reducing $\vec{\alpha}$ with $\vec{\alpha_0}$
  for all i, $1 \leqslant i \leqslant m-2$ **do**
    add the clause $C_i(\vec{\gamma_i}) \to A_i(\vec{\alpha_i})C_{i+1}(\vec{\gamma_{i+1}})$ to R where $\vec{\gamma_{i+1}}$ is obtained by reducing $\vec{\gamma_i}$ with $\vec{\alpha_i}$
  end for
  add the clause $C_{m-1}(\vec{\gamma_{m-2}}) \to A_{m-1}(\vec{\alpha_{m-1}})A_m(\vec{\alpha_m})$ to R
  for every clause $r' \in R$ **do**
    replace RHS arguments of length > 1 with new variables (in both sides) and add the result to P
  end for
end for

Algorithm 3: SRCG binarization

Algorithm 3 is the binarization algorithm. As already mentioned, it proceeds like the one for CFGs in the sense that for RHSs longer than

Original SRCG:

$$S(XYZUVW) \rightarrow A(X,U)B(Y,V)C(Z,W)$$
$$A(aX,aY) \rightarrow A(X,Y) \qquad A(a,a) \rightarrow \varepsilon$$
$$B(bX,bY) \rightarrow B(X,Y) \qquad B(b,b) \rightarrow \varepsilon$$
$$C(cX,cY) \rightarrow C(X,Y) \qquad C(c,c) \rightarrow \varepsilon$$

Clause with RHS of length > 2:

$$S(XYZUVW) \rightarrow A(X,U)B(Y,V)C(Z,W)$$

For this clause, we obtain

$$R = \{S(XYZUVW) \rightarrow A(X,U)C_1(YZ,VW),$$
$$C_1(YZ,VW) \rightarrow B(Y,V)C(Z,W)\}$$

Equivalent binarized SRCG:

$$S(XPUQ) \rightarrow A(X,U)C_1(P,Q)$$
$$C_1(YZ,VW) \rightarrow B(Y,V)C(Z,W)$$
$$A(aX,aY) \rightarrow A(X,Y) \qquad A(a,a) \rightarrow \varepsilon$$
$$B(bX,bY) \rightarrow B(X,Y) \qquad B(b,b) \rightarrow \varepsilon$$
$$C(cX,cY) \rightarrow C(X,Y) \qquad C(c,c) \rightarrow \varepsilon$$

Figure 45: SRCG Binarization

2, we introduce a new non-terminal that covers the RHS without the first element. Figure 45 shows an example. In this example, there is only one clause with a RHS longer than 2. In a first step, we introduce the binarization non-terminals and clauses that binarize the RHS. This leads to the set R. In a second step, before adding the clauses from R to the grammar, whenever a RHS argument contains several variables, they are collapsed into a single new variable.

The equivalence of the original SRCG and the binarized grammar is rather straight-forward. Note however that the arity of the SRCG can increase because of the binarization.

We can alternatively add additional unary clauses when introducing the highest and the lowest new binarization non-terminal. We call those unary clauses *top unary clause* (UnaryTop vs. BinaryTop) and *bottom unary clause* (UnaryBottom vs. BinaryBottom). While this introduces an additional factorization which is potentially beneficial for parsing, it also introduces more clauses and symbols, which potentially harms parsing efficiency.

In SRCG, in contrast to CFG, the order of the RHS elements of a clause does not matter for the result of a derivation. Therefore, we can reorder the RHS of a clause before binarizing it. If no reordering of the RHS predicates is performed, and if every non-terminal which is introduced during binarization is unique, then we talk about *deterministic binarization* (DETERM). The binarized grammar in this case is equivalent to the unbinarized grammar. Apart from that, we also use different reorderings combined with *markovization* (Collins, 1999) as used by Klein and Manning (2003b) (see p. 110). Those reorderings either result in a head-outward binarization or aim at optimizing the binarized grammar for efficient parsing by optimizing the arity of resulting grammar.

6.1.3 *Beyond Deterministic Binarization*

Head-Outward Binarization

We first want to achieve a *head-outward binarization* (Collins, 1999; Klein and Manning, 2003b) (cf. p. 106). For this, we must first mark the lexical heads[3] of every constituent. In the tradition of Collins, we employ a rule-based approach. We decide for each node in a constituency structure which of its child nodes is its head by following a list of rules for its node label. The goal of these rules is to express the linguistic constraints on what one would consider a head. A single rule consists of a search direction (left-to-right or right-to-left) and an ordered list of labels. If the list of labels is empty, then the first label in the search direction, i. e., the leftmost or the rightmost label, is marked as head. Otherwise, for each rule and node label in the node label list of the rule, the child nodes are traversed in the specified search direction. If a node label equals the label of a child, this child is marked as head and we exit (i. e., we proceed to mark the head child of the next node). Since the binarization depends on it, head rules must be designed robustly such that a head is marked for all nodes.

Consider the following example for NeGra. We use the following two rules for the label VZ (zu-marked infinitive):[4]

3 Here and in the following, *head* is not meant to refer to a particular linguistic concept of lexical heads.

4 The labels have the following meaning: VVINF – infinitive, full; VAINF – infinitive, auxiliary; VMINF – infinitive, modal; VVFIN – finite verb, full; VVIZU – infinitive

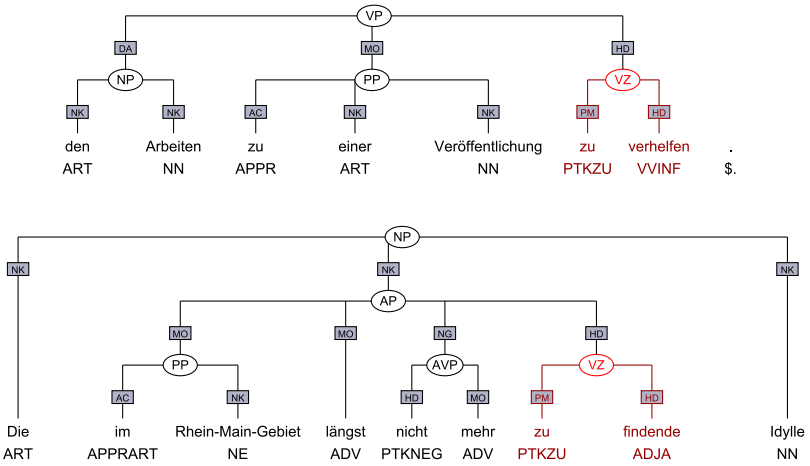


Figure 46: Head marking

```
VZ right-to-left VVINF VAINF VMINF VVFIN VVIZU
VZ left-to-right APPR PTKZU
```

In the upper tree fragment in figure 46, the first rule applies and VVINF is marked, while in the lower one, the second rule applies and PTKZU is marked.

The head finding algorithm is applied to grammars extracted from constituency treebanks only. In dependency treebanks, the head child is always the child with the POS tag non-terminal.[5]

We can now perform a reordering HeadOutward that results in a head-outward binarization where the head is the lowest subtree. The head is extended by adding first all sisters to its left and then all sisters to its right. Consequently, before binarizing we reorder the RHS of the clauses extracted from the treebank such that first, all elements to the right of the head are listed in reverse order, then all elements to the left of the head in their original order and then the head itself.

We also investigate alternatives to this reordering.

- We add first the sisters to the right and then the ones to the left. This is what Klein and Manning (2003c) do, therefore, we call this reordering HeadOutwardKM.

with zu, voll; APPR – preposition, circumposition left; PTKZU – zu in front of infinitive.

5 There exists an unsupervised method for finding heads in a PCFG (Sangati and Zuidema, 2009). The exploration of its usability for Probabilistic SRCG is left for future work.

- We add all children from left to right (L-To-R), resp. from right to left (R-To-L). For the former, the unbinarized clause does not have to be changed, while for the latter, we reverse the order of its right-hand side.

Figure 47 shows a sample HEADOUTWARD binarization with UNARYTOP and UNARYBOTTOM of a tree in the NeGra format.

Minimizing the Arity

An alternative to choosing a linguistically motivated head as the starting point for binarization and then adding the children to the left and to the right of this head is to order the RHS such that the arity and the number of variables in the binarized grammar are as minimal as possible. The merit of this approach, which we name OPTIMAL, is a possibly lower parsing complexity, since the arity has direct influence on it.

In recent work, it has been shown how to do a fully optimal binarization (Gómez-Rodríguez et al., 2009a) (see also section 6.4.2 for other work related to binarization). Here, we resort to a simpler method which yields a minimal arity and a minimal variable number per clause and binarization step.[6] We assume that we are only considering partitions of right-hand sides where one of the sets contains only a single non-terminal. For the presentation of the algorithm, we need a supplementary definition.

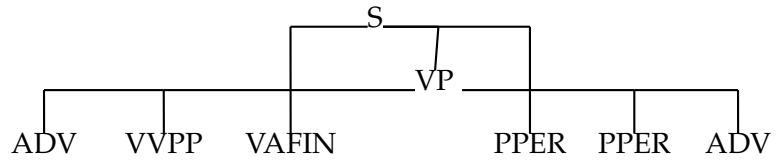
DEFINITION 6.2 (Characteristic string). Let $G = (N, T, V, P, S)$ be an SRCG. Let $c = A_0(\vec{x_0}) \rightarrow A_1(\vec{x_1}) \dots A_m(\vec{x_m}) \in P$, $m \in \mathbb{N}$. Introduce a new symbol $\$ \notin (V \cup T)$.

1. For all $1 \leqslant i \leqslant m$, the *characteristic string* $s(c, A_i)$ of the A_i-reduction of c is defined as follows: Concatenate the elements of $\vec{x_0}$, separated with \$, while replacing every component from $\vec{x_i}$ with a \$.

2. The arity of the characteristic string, $dim(s(c, A_i))$, is the number of maximal substrings $x \in V^+$ in $s(c, A_i)$.

6 This version of the algorithm has first been published in Kallmeyer (2010b) and Kallmeyer and Maier (2012).

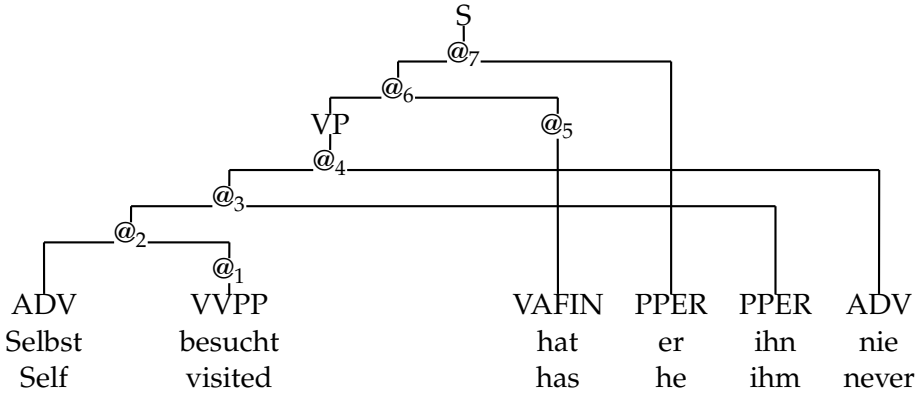
NeGra tree:



"He has never paid him a visit personally."

Rule extracted for the S node (head marked):
$\text{S}(X_1X_2X_3X_4) \rightarrow \text{VP}(X_1, X_4)\ \text{VAFIN'}(X_2)\ \text{PPER}(X_3)$
Reordering for head-outward binarization:
$\text{S}(X_1X_2X_3X_4) \rightarrow \text{PPER}(X_3)\ \text{VP}(X_1, X_4)\ \text{VAFIN}(X_2)$
New rules resulting form binarizing this rule (with unary rules):
$\text{S}(X) \rightarrow @_1(X)$
$@_1(X_1X_2X_3) \rightarrow @_2(X_1, X_3)\ \text{PPER}(X_2)$
$@_2(X_1X_2, X_3) \rightarrow \text{VP}(X_1, X_3)\ @_3(X_2)$
$@_3(X_1) \rightarrow \text{VAFIN}(X_1)$

Tree after binarization:



"He has never paid him a visit personally."

Figure 47: Binarization of a NeGra tree

The function OPTIMALCANDIDATE, algorithm 4, shows how in a first step, for a given clause c with right-hand side length > 2, we determine the optimal candidate for binarization: On all right-hand side predicates B we check for the maximal arity (given by $dim(s(c, B))$) and the number of variables ($dim(s(c, B)) + dim(B)$) we would obtain when binarizing with this predicate. This check provides the optimal candidate. In a second step we then perform a binarization as before, except that we use the optimal candidate now instead of the first element of the right-hand side. Then we repeat the process, i. e., calling OPTIMALCANDIDATE on the remaining un-binarized clause and binarizing with its result, until the length of the RHS is shorter or equal to 2.

function OPTIMALCANDIDATE($c = A_0(\vec{x_0}) \rightarrow A_1(\vec{x_1}) \ldots A_m(\vec{x_m})$)
cand $= 0$
arity = number of variables in c
vars = number of variables in c
for all $i = 0$ **to** m **do**
  cand-arity = $dim(s(r, A_i))$;
  if *cand-arity* $<$ *arity* and $dim(A_i) <$ *arity* **then**
    arity $= max(\{$*cand-arity*$, dim(A_i)\})$;
    vars $=$ *cand-arity* $+ dim(A_i)$;
    cand $= i$;
  else if *cand-arity* $\leqslant$ *arity*, $dim(A_i) \leqslant$ *arity* and *cand-arity* $+ dim(A_i) <$ *vars* **then**
    arity $= max(\{$*cand-arity*$, dim(A_i)\})$;
    vars $=$ *cand-arity* $+ dim(A_i)$;
    cand $= i$
  end if
end for
return *cand*

Algorithm 4: Determining binarization order

Markovization

For markovization, we pick a single new binarization non-terminal which does not occur in the treebank. Then we add vertical and horizontal context from the original trees to all occurrences of this new non-terminal. This is done as follows. During grammar extraction

from a treebank Υ, resulting in an SRCG $G = (N, T, V, P, S)$, we collect the *vertical contexts* of all clauses. A vertical context of a clause is the concatenation of the first v labels on the path from some node u to the root of a treebank tree (including u itself), given that the clause has been obtained from u. More precisely, for each clause $c \in P$ occurring in $k \in \mathbb{N}$ different contexts, we build the set $\mathfrak{C}_c = \{\langle \mathfrak{c}_1^c, r_1^c \rangle, \ldots, \langle \mathfrak{c}_k^c, r_k^c \rangle\}$, such that for all $1 \leqslant i \leqslant k$, c occurs r_i^c times with the vertical context $\mathfrak{c}_i^c$. We then binarize c once per vertical context, i. e., k times in total. In the ith binarization, we append the vertical context $\mathfrak{c}_i^c$ to the binarization non-terminal and we set the occurrence count of all resulting binary clauses to r_i^c. The horizontal context is built as follows. During the binarization of a (reordered) clause $A_0(\vec{\alpha_0}) \to A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$, for the new non-terminal that comprises the RHS elements $A_i \ldots A_m$ (for some $1 \leqslant i \leqslant m$), we additionally append the first h elements of $A_i, A_{i-1}, \ldots, A_1$ to the binarization non-terminal.

Figure 48 shows an example of a markovization of the tree from figure 47 with $v = 2$ and $h = 2$. Here, the superscript is the vertical context and the subscript the horizontal context of the binarization non-terminal @.

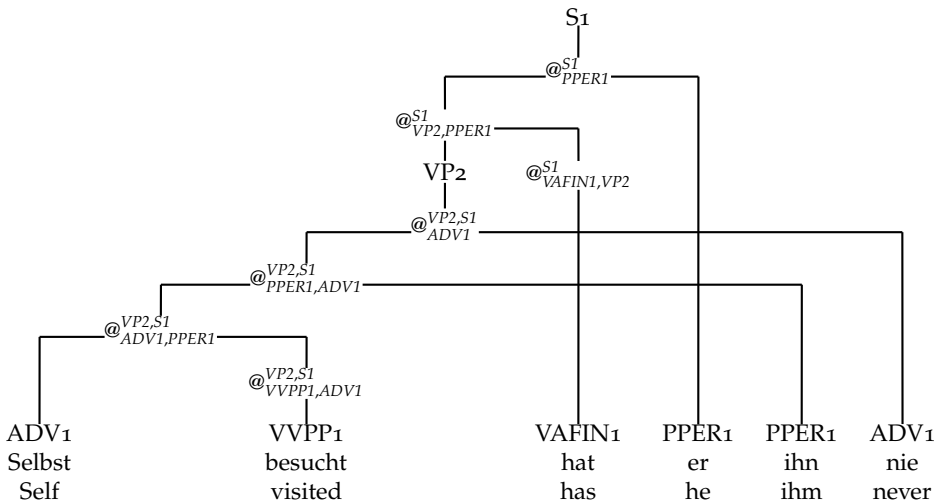


Figure 48: Markovization ($v = 2, h = 2$) of a NeGra tree

6.1.4 *Adding Probabilities*

We now extend SRCG with probabilities. This is done in parallel to PCFG.

DEFINITION 6.3 (Probabilistic Simple Range Concatenation Grammar). A *Probabilistic Simple Range Concatenation Grammar (PSRCG)* is a tuple $G = (N, T, V, P, S, p)$ where (N, T, V, P, S) is a SRCG and $p : P \to [0, 1]$ is a function which is such that for all $A \in N$,

$$\sum_{A(\vec{\alpha}) \to \vec{\Psi} \in P} p(A(\vec{\alpha}) \to \vec{\Psi}) = 1$$

The probabilities of the clauses of the binarized grammar are then computed using a Maximum Likelihood Estimator (MLE). This is also done exactly as in the context-free case. In a PSRCG $G = (N, T, V, P, S)$ which has been extracted from a treebank (def. 2.19, p. 25)

$$\Upsilon = \{(1, \upsilon_1, s_1), \ldots, (n, \upsilon_n, s_n)\},$$

$n \in \mathbb{N}$, for all $A(\vec{\alpha}) \to \alpha \in P$, the estimated probability $p(A(\vec{\alpha}) \to \vec{\Psi})$ is

$$p(A(\vec{\alpha}) \to \vec{\Psi}) = \frac{\sum_{i=1}^{n} f(A \to \vec{\Psi})}{\sum_{\vec{\Phi} \text{ s.t. } A \to \vec{\Phi} \in P} \sum_{i=1}^{n} f(A \to \vec{\Phi})}$$

where f is the function which yields the occurrence count of a clause in a treebank tree. This count can easily be obtained from the treebank.

Probabilistic versions of SRCG, resp. equivalent formalisms, have been used before (Levy, 2005; Kato et al., 2006; Søgaard, 2011). The estimation should ensure tightness, since, as Levy (2005) remarks on p. 122, in the proof of Chi and Geman (1998) (see section 4.1.3), there is nothing specific to CFG.

Just as with PCFG, we must make sure that equivalent derivations are only counted once. Analogously to definition 4.2, p. 92, in definition 6.4, we revise 1. of definition 2.48, p. 44, defining the relation *leftmost derives* for SRCG. In the leftmost derivation, always the leftmost instantiated predicate is substituted first.

DEFINITION 6.4 (Leftmost derivation (SRCG)). Let $G = (N, T, V, P, S)$ be an SRCG and $w \in T^*$ a string. The relation $\Longrightarrow_{G,w}$ is a relation called *leftmost derives* on strings of instantiated predicates the following way. Let Γ be a string of instantiated predicates. If $A_0(\vec{\psi_0}) \to$

$A_1(\vec{\psi_1}) \ldots A_m(\vec{\psi_m})$, $m \in \mathbb{N}_0$, is the instantiation of some clause $c \in P_G$, then

$$A_0(\vec{\psi_0})\Gamma \Longrightarrow_{G,w} A_1(\vec{\psi_1}) \ldots A_m(\vec{\psi_m})\Gamma.$$

We use the relation symbol of *derives* for *leftmost derives*, since we will not refer to non-leftmost derivations anymore. The probability of a derivation is also computed exactly as in the context-free case (cf. def. 4.4, p. 92).

DEFINITION 6.5 (Derivation probability (SRCG)). Let

$$G = (N, T, V, P, S, p)$$

be a PSRCG, let $w \in T^*$ be a string, and let Γ_1, Γ_2 be strings of instantiated predicates of clauses in P with respect to w.

1. Let $A(\vec{\alpha}) \to \vec{\Psi} \in P$. The probability of a derivation step

   $$\Gamma_1 \Longrightarrow_{G,w}^{A(\vec{\alpha})\to\vec{\Psi}} \Gamma_2$$

   is defined as follows.

   $$p(\Gamma_1 \Longrightarrow_{G,w}^{A(\vec{\alpha})\to\vec{\Psi}} \Gamma_2) = p(A(\vec{\alpha}) \to \vec{\Psi})$$

2. Let $A_1(\vec{\alpha_1}) \to \vec{\Psi}_1, \ldots, A_m(\vec{\alpha_m}) \to \vec{\Psi}_m \in P$, $m \in \mathbb{N}$. The probability of a derivation $\Gamma_1 \Longrightarrow_{G,w}^{A_1(\vec{\alpha_1})\to\vec{\Psi}_1} \cdots \Longrightarrow_{G}^{A_m(\vec{\alpha_m})\to\vec{\Psi}_m} \Gamma_2$ is defined as follows.

   $$p(\Gamma_1 \Longrightarrow_{G,w}^{A_1(\vec{\alpha_1})\to\vec{\Psi}_1} \cdots \Longrightarrow_{G,w}^{A_m(\vec{\alpha_m})\to\vec{\Psi}_m} \Gamma_2) = \prod_{i=1}^{m} p(A_i(\vec{\alpha_i}) \to \vec{\Psi}_i)$$

3. The probability of $\Gamma_1 \overset{*}{\Longrightarrow}_{G,w} \Gamma_2$ is defined as the sum over the probabilities of all leftmost derivations of Γ_1 from Γ_2:

   $$p(\Gamma_1 \overset{*}{\Longrightarrow}_{G,w} \Gamma_2) = \sum_{i=0}^{k} \prod_{j=0}^{m_i} p(A_j^i(\vec{\alpha_j^i}) \to \vec{\psi_j^i})$$

   where $k \in \mathbb{N}$ is the number of leftmost derivations of Γ_2 from Γ_1 and $m_i \in \mathbb{N}$ is the derivation length of the ith derivation and $A_j^i(\vec{\alpha_j^i}) \to \vec{\psi_j^i}$ is clause used in the jth derivation step of the ith leftmost derivation.

6.2 PARSING

6.2.1 *CYK Parsing*

We employ a probabilistic CYK parser, using Weighted Deductive Parsing (WDP) (see section 4.1.2). Without loss of generality, we can make the assumptions

- that due to binarization, all clauses in the grammar are of rank 1 or 2,
- that the parser input is POS tagged, and therefore there are no clauses in which predicate components are terminals, and
- that due to the extraction algorithm, we do not have clauses in which LHS components are ε.

In summary, the algorithm must only handle binary clauses in which all predicate argument components are variables.

For an input string w and a PSRCG $G = (N, T, V, P, S, p)$, our items have the form $[A, \vec{\rho}]$ where $A \in N$, $\vec{\rho} \in (Pos(w) \times Pos(w))^{dim(A)}$ is the vector of ranges characterizing all components of the span of A. We specify the set of weighted parse items via the deduction rules in figure 49, using algorithm 2, p. 96, and build the best parse from the goal item as described on p. 56. Recall that as the weights of clause, we use the absolute value of the natural logarithm of its probability, i. e., for a clause c with probability p, we use $|ln(p)|$. Nevertheless, abbreviating, we just write $log(p)$.

6.2.2 *Outside Score Estimates*

In order to speed up parsing, we add an estimate of the log of the outside score of the items to their inside weights in the agenda (see A^* parsing in section 4.1.2).[7] All our outside estimates are admissible, which means that they never underestimate the actual outside score of the item, but one of them is not monotonic. While the monotonic estimates allow for true A^* parsing, with the non-monotonic one, it can happen that we deduce an item I_2 from an item I_1 where the

7 Recall that *inside score* and *outside score* always denote the corresponding Viterbi scores and not the actual inside and outside probabilities.

$$\textsc{scan} \quad \frac{}{0 : [A, \langle\langle i, i+1\rangle\rangle]} \; A \text{ POS tag of } w_{i+1}$$
where A is the POS tag of w_{i+1}.

$$\textsc{unary} \quad \frac{in : [B, \vec{\rho}]}{in + log(p) : [A, \vec{\rho}]} \; p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$
where $p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$.

$$\textsc{binary} \quad \frac{in_B : [B, \vec{\rho_B}], in_C : [C, \vec{\rho_C}]}{in_B + in_C + log(p) : [A, \vec{\rho_A}]}$$ where
$p : A(\vec{\rho_A}) \to B(\vec{\rho_B})C(\vec{\rho_C})$ is an instantiated rule.

$$\textsc{goal} \quad [S, \langle\langle 0, n\rangle\rangle]$$

Figure 49: Weighted CYK deduction system

weight of I_2 is greater than the weight of I_1. The parser can therefore end up with a local maximum that is not the global maximum we are searching for. In other words, this outside estimate is actually a Figure-of-Merit (FOM) (as in best-first parsing).

We precompute all outside estimates offline up to a certain maximal sentence length len_{max}.

Full SX estimate

The full SX estimate,[8] for a given sentence length n, is supposed to give the maximal probability of completing a category C with a span $\vec{\rho}$ into an S with span $\langle\langle 0, n\rangle\rangle$. In order to avoid computing the same partial results several times, we use a deductive approach that allows for tabulation.

For the computation, we need an estimate of the inside score of a category C with a span $\vec{\rho}$, regardless of the actual terminals in our input. This inside estimate is computed as shown in figure 50. Here, we do not need to consider the number of terminals outside the span of C (to the left or right or in the gaps), they are not relevant for the inside score. Therefore the items have the form $[A, \langle l_1, \ldots, l_{dim(A)}\rangle]$, where A is a non-terminal and l_i gives the length of its ith component. It holds

8 The full SX estimate is the naive SRCG adaption of the *SX estimate* of Klein and Manning (2003a).

$$\textsc{pos tags}\quad \frac{}{0 : [A, \langle 1\rangle]}\ A \text{ a POS tag}$$

$$\textsc{unary}\quad \frac{in : [B, \vec{l}]}{in + log(p) : [A, \vec{l}]}\ p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$

$$\textsc{binary}\quad \frac{in_B : [B, \vec{l}_B], in_C : [C, \vec{l}_C]}{in_B + in_C + log(p) : [A, \vec{l}_A]}$$

where $p : A(\vec{\alpha_A}) \to B(\vec{\alpha_B})C(\vec{\alpha_C}) \in P$ and the following holds: we define $\mathcal{B}(i)$ as $\{1 \leqslant j \leqslant dim(B) \mid \vec{\alpha_B}(j)$ occurs in $\vec{\alpha_A}(i)\}$ and $\mathcal{C}(i)$ as $\{1 \leqslant j \leqslant dim(C) \mid \vec{\alpha_C}(j)$ occurs in $\vec{\alpha_A}(i)\}$. Then for all i, $1 \leqslant i \leqslant dim(A)$: $\vec{l}_A(i) = \Sigma_{j\in\mathcal{B}(i)}\vec{l}_B(j) + \Sigma_{j\in\mathcal{C}(i)}\vec{l}_C(j)$.

Figure 50: Inside estimate

that $\Sigma_{1\leqslant i\leqslant dim(A)} l_i \leqslant len_{max} - dim(A) + 1$ because the way our grammars are extracted from the treebank, we can be sure that the different components in the yield of a non-terminal are never adjacent. There is always at least one terminal in between two different components that does not belong to the yield of the non-terminal.

The first rule in figure 50 tells us that POS tags always have a single component of length 1, therefore this case has probability 1 (weight 0). The rules UNARY and BINARY are like the ones in the CYK parser, except that they combine items with length information. The rule UNARY for instance tells us that if the log of the probability of building $[B, \vec{l}]$ is greater or equal to *in* and if there is a clause that allows to deduce an A item from $[B, \vec{l}]$ with probability p, then the log of the probability of $[A, \vec{l}]$ is greater or equal to $in + log(p)$. For each item, we record its maximal weight, i.e., its maximal probability. The rule BINARY is slightly more complicated since we have to compute the length vector of the LHS of the clause from the RHS length vectors.

A straightforward extension of the CFG algorithm from Klein and Manning (2003b) for computing the SX estimate is given in figure 51. Here, the items have the form $[A, \vec{l}]$ where the vector $\vec{l}$ tells us about the lengths of the string to the left of the first component, the first component, the string in between the first and second component and so on.

The algorithm proceeds top-down. The outside estimate of completing an S with component length *len* and no terminals to the left or to the right of the S component (item $[S, \langle 0, len, 0\rangle]$) is 0. If we expand

$$\textsc{axiom}\quad \frac{}{0 : [S, \langle 0, len, 0\rangle]}\ 1 \leqslant len \leqslant len_{max}$$

$$\textsc{unary}\quad \frac{w : [A, \vec{l}]}{w + log(p) : [B, \vec{l}]}\ p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$

$$\textsc{binary-right}\quad \frac{w : [X, \vec{l}_X]}{w + in(A, \vec{l}'_A) + log(p) : [B, \vec{l}_B]}$$

$$\textsc{binary-left}\quad \frac{w : [X, \vec{l}_X]}{w + in(B, \vec{l}'_B) + log(p) : [A, \vec{l}_A]}$$

where, for both rules, there is an instantiated rule $p : X(\vec{\rho}) \to A(\vec{\rho}_A)B(\vec{\rho}_B)$ such that $\vec{l}_X = l_{out}(\vec{\rho})$, $\vec{l}_A = l_{out}(\vec{\rho}_A), \vec{l}'_A = l_{in}(\vec{\rho}_A)$, $\vec{l}_B = l_{out}(\vec{\rho}_B), \vec{l}'_B = l_{in}(\vec{\rho}_B)$.

Figure 51: Full SX estimate top-down

with a unary clause (UNARY), then the outside estimate of the RHS item is greater or equal to the outside estimate of the LHS item plus the log of the probability of the clause. In the case of binary clauses, we have to further add the inside estimate of the other daughter. For this, we need a different length vector (without the lengths of the parts in between the components). Therefore, for a given range vector $\vec{\rho} = \langle\langle l_1, r_1\rangle, \ldots, \langle l_k, r_k\rangle\rangle$ and a sentence length n, we distinguish between the *inside length vector* $l_{in}(\vec{\rho}) = \langle r_1 - l_1, \ldots, r_k - l_k\rangle$ and the *outside length vector* $l_{out}(\vec{\rho}) = \langle l_1, r_1 - l_1, l_2 - r_1, \ldots, l_k - r_{k-1}, r_k - l_k, n - r_k\rangle$.

This algorithm has two major problems: Since it proceeds top-down, in the BINARY rules, we must compute all splits of the antecedent X span into the spans of A and B which is very expensive. Furthermore, for a category A with a certain number of terminals in the components and the gaps, we compute the lower part of the outside estimate several times, namely for every combination of number of terminals to the left and to the right (first and last element in the outside length vector). In order to avoid these problems, we now abstract away from the lengths of the part to the left and the right, modifying our items such as to allow a bottom-up strategy.

The idea is to compute the weights of items representing the derivations from a certain lower C up to some A (C is a kind of "gap" in the yield of A) while summing up the inside costs of off-spine nodes and the log of the probabilities of the corresponding clauses. We use items $[A, C, \vec{\rho}_A, \vec{\rho}_C, \mathit{shift}]$ where $A, C \in N$ and $\vec{\rho}_A, \vec{\rho}_C$ are range vec-

$$\textsc{pos tags}\quad \frac{}{0 : [C, C, \langle 0, 1\rangle, \langle 0, 1\rangle, 0]}\ C \text{ a POS tag}$$

$$\textsc{unary}\quad \frac{0 : [B, B, \vec{\rho}_B, \vec{\rho}_B, 0]}{log(p) : [A, B, \vec{\rho}_B, \vec{\rho}_B, 0]}\ p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$

$$\textsc{binary-right}\quad \frac{0 : [A, A, \vec{\rho}_A, \vec{\rho}_A, 0], 0 : [B, B, \vec{\rho}_B, \vec{\rho}_B, 0]}{in(A, l_{in}(\vec{\rho}_A)) + log(p) : [X, B, \vec{\rho}_X, \vec{\rho}_B, i]}$$

$$\textsc{binary-left}\quad \frac{0 : [A, A, \vec{\rho}_A, \vec{\rho}_A, 0], 0 : [B, B, \vec{\rho}_B, \vec{\rho}_B, 0]}{in(B, l_{in}(\vec{\rho}_B)) + log(p) : [X, A, \vec{\rho}_X, \vec{\rho}_A, 0]}$$

where i is such that for $shift(\vec{\rho}_B, i) = \vec{\rho}'_B$ $p : X(\vec{\rho}_X) \to A(\vec{\rho}_A)B(\vec{\rho}'_B)$ is an instantiated clause.

Starting sub-trees with larger gaps:

$$\frac{w : [B, C, \vec{\rho}_B, \vec{\rho}_C, i]}{0 : [B, B, \vec{\rho}_B, \vec{\rho}_B, 0]}$$

Transitive closure of sub-tree combination:

$$\frac{w_1 : [A, B, \vec{\rho}_A, \vec{\rho}_B, i], w_2 : [B, C, \vec{\rho}_B, \vec{\rho}_C, j]}{w_1 + w_2 : [A, C, \vec{\rho}_A, \vec{\rho}_C, i + j]}$$

Figure 52: Full SX estimate bottom-up

tors, both with a first component starting at position 0. The integer $shift \leqslant len_{max}$ tells us how many positions to the right the C span is shifted, compared to the starting position of the A. $\vec{\rho}_A$ and $\vec{\rho}_C$ represent the spans of C and A while disregarding the number of terminals to the left the right. I.e., only the lengths of the components and of the gaps are encoded. This means in particular that the length n of the sentence does not play a role here. The right boundary of the last range in the vectors is limited to len_{max}. For any i, $0 \leqslant i \leqslant len_{max}$, and any range vector $\vec{\rho}$, we define $shift(\vec{\rho}, i)$ as the range vector one obtains from adding i to all range boundaries in $\vec{\rho}$ and $shift(\vec{\rho}, -i)$ as the range vector one obtains from subtracting i from all boundaries in $\vec{\rho}$.

The weight of $[A, C, \vec{\rho}_A, \vec{\rho}_C, i]$ estimates the *log* of the probability of completing a C tree with yield $\vec{\rho}_C$ into an A tree with yield $\vec{\rho}_A$ such that, if the span of A starts at position j, the span of C starts at position $i + j$. Figure 52 gives the computation. The value of $in(A, \vec{l})$ is the inside estimate of $[A, \vec{l}]$.

$$\textsc{axiom} \quad \frac{}{0 : [S, len, 0, 0, 0]} \; 1 \leqslant len \leqslant len_{max}$$

$$\textsc{unary} \quad \frac{w : [X, len, l, r, g]}{w + log(p) : [A, len, l, r, g]}$$

where $p : X(\vec{\alpha}) \rightarrow A(\vec{\alpha}) \in P$.

$$\textsc{binary-right} \quad \frac{w : [X, len, l, r, g]}{w + in(A, len - len_B) + log(p) : [B, len_B, l_B, r_B, g_B]}$$

$$\textsc{binary-left} \quad \frac{w : [X, len, l, r, g]}{w + in(B, len - len_A) + log(p) : [A, len_A, l_A, r_A, g_A]}$$

where, for both rules, $p : X(\vec{\alpha}) \rightarrow A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$.

Figure 53: SX with length, left, right, gaps

The SX estimate for some predicate C with span $\vec{\rho}$ where i is the left boundary of the first component of $\vec{\rho}$ and with sentence length n is then given by the maximal weight of $[S, C, \langle 0, n\rangle, \mathit{shift}(\vec{\rho}, -i), i]$.

SX with Left, Gaps, Right, Length

A problem of the previous estimate is that with a large number of non-terminals the computation of the estimate requires too much space. Our experiments have shown that for treebank parsing where we have, after binarization and markovization, appr. 12,000 non-terminals, its computation is not feasible. We therefore turn to simpler estimates with only a single non-terminal per item. We now estimate the Viterbi outside score probability of a non-terminal A with a span of a length *len* (the sum of the lengths of all the components of the span), with *left* terminals to the left of the first component, *right* terminals to the right of the last component and *gaps* terminals in between the components of the A span, i.e., filling the gaps. Our items have the form $[X, len, left, right, gaps]$ with $X \in N$, $len + left + right + gaps \leqslant len_{max}$, $len \geqslant dim(X)$, $gaps \geqslant dim(X) - 1$.

Let us assume that, in the clause $X(\vec{\alpha}) \rightarrow A(\vec{\alpha_A})B(\vec{\alpha_B})$, when looking at the vector $\vec{\alpha}$, we have $left_A$ variables for A-components preceding the first variable of a B component, $right_A$ variables for A-components following the last variable of a B component and $right_B$ variables for B-components following the last variable of a A component. (In our grammars, the first LHS argument always starts with the

$$\textsc{pos tags} \quad \frac{}{0 : [A, 1]} \; A \text{ a POS tag}$$

$$\textsc{unary} \quad \frac{in : [B, l]}{in + log(p) : [A, l]} \; p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$

$$\textsc{binary} \quad \frac{in_B : [B, l_B], in_C : [C, l_C]}{in_B + in_C + log(p) : [A, l_B + l_C]}$$

where either $p : A(\vec{\alpha_A}) \to B(\vec{\alpha_B})C(\vec{\alpha_C}) \in P$ or $p : A(\vec{\alpha_A}) \to C(\vec{\alpha_C})B(\vec{\alpha_B}) \in P$.

Figure 54: Inside estimate with total span length

first variable from A.) Furthermore, $gaps_A = dim(A) - left_A - right_A$, $gaps_B = dim(B) - right_B$.

Figure 53 gives the computation of the estimate. It proceeds top-down, as the computation of the full SX estimate in Figure 51, except that now the items are simpler. The following side conditions must hold: For BINARY-RIGHT to apply, the following constraints must be satisfied:

1. $len + l + r + g = len_B + l_B + r_B + g_B$,
2. $l_B \geqslant l + left_A$,
3. if $right_A > 0$, then $r_B \geqslant r + right_A$, else ($right_A = 0$), $r_B = r$,
4. $g_B \geqslant gaps_A$.

Similarly, for BINARY-LEFT to apply, the following constraints must be satisfied:

1. $len + l + r + g = len_A + l_A + r_A + g_A$,
2. $l_A = l$,
3. if $right_B > 0$, then $r_A \geqslant r + right_B$, else ($right_B = 0$), $r_A = r$
4. $g_A \geqslant gaps_B$.

The value $in(X, l)$ for a non-terminal X and a length l, $0 \leqslant l \leqslant len_{max}$ is an estimate of the probability of an X category with a span of length l. Its computation is specified in figure 54.

The SX estimate for a sentence length n and for some predicate C with a range characterized by $\vec{\rho} = \langle\langle l_1, r_1\rangle, \ldots, \langle l_{dim(C)}, r_{dim(C)}\rangle\rangle$ where $len = \Sigma_{i=1}^{dim(C)}(r_i - l_i)$ and $r = n - r_{dim(C)}$ is then given by the maximal weight of the item $[C, len, l_1, r, n - len - l_1 - r]$.

$$\textsc{axiom}\quad \frac{}{0 : [S, len, 0, 0]}\ 1 \leqslant len \leqslant len_{max}$$

$$\textsc{axiom}\quad \frac{w : [X, len, lr, g]}{w + \log(p) : [A, len, lr, g]}$$

where $p : X(\vec{\alpha}) \to A(\vec{\alpha}) \in P$.

$$\textsc{binary-right}\quad \frac{w : [X, len, lr, g]}{w + in(A, len - len_B) + \log(p) : [B, len_B, lr_B, g_B]}$$

$$\textsc{binary-left}\quad \frac{w : [X, len, lr, g]}{w + in(B, len - len_A) + \log(p) : [A, len_A, lr_A, g_A]}$$

where, for both rules, $p : X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$.

Figure 55: SX estimate with length, LR, gaps

SX with LR, Gaps, Length

In order to further decrease the space complexity of the computation of the outside estimate, we can simplify the previous estimate by subsuming the two lengths *left* and *right* in a single length *lr*. I.e., the items now have the form $[X, len, lr, gaps]$ with $X \in N$, $len + lr + gaps \leqslant len_{max}$, $len \geqslant dim(X)$, $gaps \geqslant dim(X) - 1$. This estimate will be called the LR estimate.

The computation is given in figure 55. Again, we define $left_A$, $gaps_A$, $right_A$ and $gaps_B$, $right_B$ for a clause $X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B})$ as above. The side conditions are as follows: For *Binary-right* to apply, the following constraints must be satisfied:

1. $len + lr + g = len_B + lr_B + g_B$,
2. $lr < lr_B$, and
3. $g_B \geqslant gaps_A$.

For *Binary-left* to apply, the following must hold:

1. $len + lr + g = len_A + lr_A + g_A$,
2. if $right_B = 0$ then $lr = lr_A$, else $lr < lr_A$ and
3. $g_A \geqslant gaps_B$.

Furthermore, in both BINARY-LEFT and BINARY-RIGHT, we have limited *lr* in the consequent item to the *lr* of the antecedent plus the

$$\textsc{axiom}\quad \frac{}{0 : [S, len, len]}\ 1 \leqslant len \leqslant len_{max}$$

$$\textsc{unary}\quad \frac{w : [X, l_X, slen]}{w + log(p) : [A, l_X, slen]}$$

where $p : X(\vec{\alpha}) \rightarrow A(\vec{\alpha}) \in P$.

$$\textsc{binary-right}\quad \frac{w : [X, l_X, slen]}{w + in(A, l_X - l_B) + log(p) : [B, l_B, slen]}$$

$$\textsc{binary-left}\quad \frac{w : [X, l_X, slen]}{w + in(B, l_X - l_A) + log(p) : [A, l_A, slen]}$$

where, for both rules, $p : X(\vec{\alpha}) \rightarrow A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$.

Figure 56: SX estimate with span and sentence length

length of the sister (len_B, resp. len_A). This results in a further reduction of the number of items while having only little effect on the parsing results.

The estimate for a sentence length n and for some predicate C with a span $\vec{\rho} = \langle\langle l_1, r_1\rangle, \ldots, \langle l_{dim(C)}, r_{dim(C)}\rangle\rangle$ where $len = \Sigma_{i=1}^{dim(C)}(r_i - l_i)$ and $r = n - r_{dim(C)}$ is then the maximal weight of $[C, len, l_1 + r, n - len - l_1 - r]$.

SX with Span and Sentence Length

We will now present a further simplification of the last estimate that records only the span length and the length of the entire sentence. The items have the form $[X, len, slen]$ with $X \in N$, $dim(X) \leqslant len \leqslant slen$. The computation is given in figure 56. This last estimate, called LN estimate, is actually monotonic and allows for true A^* parsing. For a proof, see Kallmeyer and Maier (2012).

The SX estimate for a sentence length n and for some predicate C with a span $\vec{\rho} = \langle\langle l_1, r_1\rangle, \ldots, \langle l_{dim(C)}, r_{dim(C)}\rangle\rangle$ where $len = \Sigma_{i=1}^{dim(C)}(r_i - l_i)$ is then the maximal weight of $[C, len, n]$.

6.3 IMPLEMENTATION

The techniques described in this chapter have been implemented in a single system, called `rparse`. `rparse` is publicly available under the

GNU General Public License 2.0 from `http://www.wolfgang-maier.net/rparse/`. Some implementation details follow.

6.3.1 *Items*

When parsing a single sentence, the parser can easily produce several millions of items. Especially for the COMPLETE operation, which checks the compatibility of two range vectors and the variable configuration of a certain clause, efficiency is therefore crucial.

We use Plaehn's (2004) item representation. Let w be the input string of length n. In an item $[A, \vec{\rho}]$ where A is a non-terminal and $\vec{\rho}$ is a range vector, we represent $\vec{\rho}$ by a *bit string* $b_1 \cdots b_{|w|} \in \{0,1\}^+$ where for all b_i, $1 \leqslant i \leqslant |w|$ it holds that if there is a range $\langle u, v\rangle \in \vec{\rho}$ with $u < i \leqslant v$, then $b_i = 1$; otherwise $b_i = 0$. For example, if $\vec{\rho} = (\langle 0,1\rangle, \langle 5,8\rangle)$ and $|w| = 9$, then the corresponding bit string is 100001110.

For the combination of two items in the COMPLETE rule, one must ensure the compatibility of the range vectors and the variables of the clause which is involved. Since the parser only has to handle binary clauses which have variables as their arguments, we can represent the variables as a two-dimensional boolean array y. y specifies how the arguments of the LHS predicate are composed of the arguments of the RHS predicates.[9] Thereby, given a binary clause $A_0(\vec{\alpha}_0) \rightarrow A_1(\vec{\alpha}_1)A_2(\vec{\alpha}_2)$, $y[i][j]$ is false if the jth variable of the ith argument of the LHS comes from A_1, otherwise it is true. The implementation of the COMPLETE operation consists of a recursive traversal of the boolean array representing the yield function. As the traversal advances, range boundaries from both RHS predicates can be checked for conformity with the yield function.

6.3.2 *Priority Queue*

The implementation of the priority queue has a decisive impact on parsing performance. In an earlier implementation which was used for previously published experiments (Maier, 2010; Maier and Kallmeyer, 2010; Kallmeyer and Maier, 2010), a binary heap was used (Cormen

9 In fact, y rather represents an Linear Context-Free Rewriting System (LCFRS) yield function than SRCG variables, since intuitively, an SRCG derivation proceeds top-down, while LCFRS proceeds bottom-up.

et al., 2003, p. 164). However, the implementation was faulty: The update operation violated the heap property, i. e., the node keys were updated but on an update, the heap was not reordered.[10] On the other hand, an update took only $\mathcal{O}(1)$. The current version of `rparse` realizes the priority queue as a Fibonacci heap (Fredman and Tarjan, 1987), a data structure which can serve as a priority queue and is particularly apt to back incarnations of Dijkstra's algorithm, such as WDP (Cormen et al., 2003, pp. 505).

6.3.3 *Filtering*

Simple Range Concatenation Grammar, as used in this chapter, allows gaps to be arbitrarily large. However, gaps are not arbitrarily large in the treebank data. Blocking items which create implausibly large gaps should therefore help to prune the search space. The blocking of items which contain gaps with a size greater than some $\Delta \in \mathbb{N}$ can easily be integrated into the function which computes the combination of range vectors. Such a filter amounts to adding the following side condition to the BINARY rule in figure 49. Take the bit string representation of $\vec{\rho_A}$ and remove all leading and trailing 0s. The resulting string may not contain a substring $s \in \{0\}^*$ with $|s| > \Delta$. We will investigate the effect of this filter in the GAPFILTER experiments.

Another type of filter which can be applied within the BINARY rule of figure 49 is a filter for items which represent ill-nested constituents (ILL-NESTEDFILTER). It requires to add the side condition that constituents represented by antecedent items fulfill the condition for well-nestedness, i. e., it requires that there be no $i_1, i_2 \in \vec{\rho_A}$ and $j_1, j_2 \in \vec{\rho_B}$ with $i_1 < j_1 < i_2 < j_2$.

10 Thanks to Andreas van Cranenburgh for pointing this out.

6.4 RELATED WORK

6.4.1 *Data-Driven Parsing*

Direct Modeling of Discontinuous Constituents

PROBABILISTIC DPSG Plaehn's (2004) approach to parsing NeGra using Probabilistic Discontinuous Phrase Structure Grammar has already been presented in section 4.2.3.

PROBABILISTIC MCFG Levy (2005) presents a parser for *Probabilistic Wrapping Grammars*. As mentioned in chapter 4, a Probabilistic Wrapping Grammar is a Probabilistic Multiple Context-Free Grammar which respects the bottom-up non-erasing condition, as well as a condition Levy calls *strict concatenation*. This conditions requires that in a function, there are either only variables, or a single terminal. The grammars we obtain with our extraction algorithm (cf. p. 137) have the exact same form. Even though Levy describes some experiments (reporting numbers of items which have been generated, etc.), he does not evaluate them. His parser builds on a variety of ideas.

- His head finding algorithm (p. 128) describes an additional reordering of the children by their respective heads. We do not do that since we assume non-terminals to be ordered by the leftmost terminal they dominate.
- His binarization algorithm (p. 126) creates a head-outward binarization, with a binary top clause and an unary head clause. However, Levy does not describe how the predicate arguments are handled during the binarization.
- His probability estimation using MLE is identical to the method presented earlier.
- Concerning his parser, Levy references the parsing algorithms for MCFG presented by Harkema (2001). The CYK algorithm is identical to Seki et al.'s algorithm. Apart from that, he performs Weighted Deductive Parsing like we do.
- On p. 131, he sketches the idea for the MCFG adaption of the SX estimate, i. e., the full SX estimate presented in this work in section 6.2.2. However, no parsing results are reported.

Levy notes two deficiencies of the simple Maximum Likelihood model. First, the arity distinction of predicates is unmotivated and only an artifact of the formalism, and second, it is not possible to express distances as a feature (in the sense of Collins (1999)). This would be needed, for example, to make the fact whether a constituent is extraposed or not a parameter which can be estimated. He therefore sketches a more sophisticated estimation technique, based on Immediate Dominance/Linear Precedence (ID/LP), which would remedy both problems. However, he also notes that such a factorized grammar cannot be learned in a completely supervised manner anymore.

Recent work by Grove (2010) describes the implementation of a probabilistic parser for MCFG to be used as a backend for parsing Minimalist Grammar. To my knowledge, there is no publication on this system, besides the webpage.

As already mentioned in chapter 3, Kato et al. (2006) use a restricted version of Probabilistic MCFG for the modeling of RNA pseudoknotted structures. Pseudoknot is a typical substructure in secondary structures of several RNAs. It poses a particular difficulty for *RNA structure prediction*, the task of determining nucleic acid secondary or tertiary structure from its sequence. Long-distance resp. crossing dependencies are what it makes difficult. Figure 17, p. 88, shows such a structure together with an alternative representation called *arc depiction*.

PROBABILISTIC SRCG The annotation of the Penn Treebank (PTB) contains discontinuities in the form of trace nodes and coindexations (cf. p. 99) on top of a annotation backbone based on CFG. Evang (2011), resp. Evang and Kallmeyer (2011), convert the PTB trees into trees with crossing branches, such as in NeGra. Many linguistic considerations are involved in this process, since the conversion is far from straight-forward. `rparse` is then used to parse the transformed treebank. The results show that the additional expressivity of SRCG has a positive influence on the results.

An extension of the approach in this thesis is presented by van Cranenburgh et al. (2011). They extend it to Data-Oriented Parsing (DOP), by adapting Goodman's (2003) reduction (cf. p. 112) to LCFRS. Their approach outperforms ours slightly.

DEFINITE CLAUSE GRAMMAR Johnson (1985) uses *Definite Clause Grammar (DCG)* to parse discontinuous constituents. Many ideas of those presented in this chapter already exist in his work. Johnson drops the requirement that every non-terminal has to dominate a single range. Essentially, his DCG predicates look just like SRCG predicates, with the exception that their arguments are also used to pass other information than range boundaries, such as case information. He also proposes a representation of range boundaries as bit sets, such as the ones we use. For parsing, he employs a deductive framework (Pereira and Warren, 1983) and remarks that a "head-first" strategy would be the most appropriate (i. e., first expanding the head and then the complements).

Factoring of Complexity

Alternatively to directly using crossing branches, one can convert the syntactic structures with discontinuities to a simpler representation (i. e., a representation which is simpler to process) and make this alternative representation reversible by using a suitable labeling scheme. In other words, information encoded in crossing branches is moved into the labeling schema.

DEPENDENCIES Hall and Nivre (2008b,a) present an interesting approach to parsing discontinuities in NeGra which is not directly comparable to any of the other approaches presented before in this chapter. The basis for their work consists of a conversion algorithm from discontinuous constituents to a dependency representation, comparable to Lin's (1995) dependency conversion, and a labeling schema for the converted dependencies, generated from the discontinuous constituents. The conversion is reversible, i. e., the dependencies can be converted back to constituents. The dependencies can be parsed with any dependency parser; Hall and Nivre use the MALT parser and achieve high-quality results.

Kahane et al. (1998) introduce *pseudo-projectivity* (cf. p. 154). Their motivation is to allow non-projectivity while maintaining a polynomial parsing complexity. Pseudo-projectivity allows for a reversible transformation of certain non-projective graphs into projective graphs by an operation called *lifting*. The transformation is reversible, because the information provided by the non-projective edges is not lost,

but encoded in the node labels. Nivre and Nilsson (2005) provide a pseudo-projective dependency parser.

CONSTITUENTS Another technique on the same lines is the algorithm for resolving crossing branches by Boyd (2007) (cf. p. 102), with which the crossing branches in NeGra can be resolved. It removes the crossing branches of a non-terminal by introducing new non-terminals for each of its yield blocks. Boyd introduces this algorithm only as an alternative to the standard method of resolving crossing branches in NeGra. However, it could also serve as the starting point for parsing discontinuous constituents. If one additionally introduces a suitable labeling scheme to make the transformation reversible, one could parse the converted representation with a PCFG parser and then reintroduce the crossing branches in the parser output. The context-free representation could alternatively be used as a guide in the sense of Barthélemy et al. (2001) (see p. 85), or possibly as a coarse grammar within a *coarse-to-fine* framework (see p. 112). This will be pursued in future work.

6.4.2 *Formal Properties of LCFRS and SRCG*

Rank and fan-out of LCFRS, resp. rank and arity of SRCG, are two dimensions of grammars which independently influence their parsing complexity (Rambow and Satta, 1999). Recently, for this reason, rank and fan-out reduction of LCFRS have attracted interest, more precisely, the question of how grammars can be transformed to equivalent grammars with lower rank and/or fan-out, ideally such that an optimal parsing complexity is guaranteed. This problem is also called *grammar factorization*.

Rank and Fan-Out Reduction

Gómez-Rodríguez et al. (2009a) present an algorithm which transforms an LCFRS into a strongly equivalent LCFRS which has rank 2 or less and has minimal fan-out.

For those LCFRSs which have fan-out at most 2, Gómez-Rodríguez and Satta (2009) present a binarization algorithm, i. e., an algorithm for the reduction of the rank to 2, which runs in optimal time (linear time in the rule length) and does not increase the fan-out of the rule.

Sagot and Satta (2010) present the counterpart to Gómez-Rodríguez et al. (2009a). Their algorithm transforms an LCFRS with a fan-out of at most 2 into a strongly equivalent LCFRS with a minimal rank.

Parsing Complexity

Recently, work has appeared which is concerned with algorithms that find the optimal parsing algorithm for a given grammar (*meta parsing algorithms*). Concerning LCFRS, these are algorithms in which fan-out and rank are optimized in function of the parsing complexity instead of assuming a fixed fan-out or rank as the algorithms above.

Gildea (2010) investigates how one can find the grammar factorization which guarantees optimal parsing complexity. Gildea (2011) describes a more general construction for finding the optimal parsing strategy for different formalisms.

Crescenzi et al. (2011) extend Gildea's work to head-driven parsing, i. e., parsing algorithms which take lexical heads into account. Their work is motivated by the work on data-driven parsing presented in this thesis (as published in Kallmeyer and Maier (2010)). A *head-driven strategy* is a parsing strategy which first expands the lexical head of a rule and then, one by one, adds the modifiers to the partial parse with the lexical head. Crescenzi et al. show that choosing the parsing strategy which has either the lowest time complexity or the lowest space complexity out of the $n!$ different strategies which exist for a single LCFRS rule is NP hard. Crescenzi et al.'s approach is different to Gildea's (2010). They show strict NP hardness while Gildea only shows that a polynomial time algorithm for finding the optimal parsing strategy would imply the existence of better algorithms for the underlying graph-theoretical problems. However, their results only hold for head-driven strategies, while Gildea's results hold for unrestricted parsing strategies.

Note finally that the problem of grammar factorization has been addressed for other formalisms as well, e. g., for Synchronous CFG. See Gildea (2010) for pointers to the corresponding literature.

Well-Nestedness

Gómez-Rodríguez et al. (2010) present a parsing algorithm for well-nested LCFRS (def. 5.13, p. 140) which runs in $\mathcal{O}(f \cdot |G| \cdot |w|^{2f+2})$, for an input string w and a grammar G with fan-out f. In contrast, Seki's

algorithm (the one we use) runs in $\mathcal{O}(|G| \cdot |w|^{f \cdot (r+1)})$ where r is the rank of G (Gómez-Rodríguez and Satta, 2009). In other words, as Gómez-Rodríguez et al. remark, by removing ill-nestedness, we make the parsing complexity depend exclusively on the fan-out, instead of the rank and the fan-out. This clear is made clear by lemma 5.11, p. 139: In well-nested clauses, RHS predicates are independent from each other, only in the case of ill-nestedness one must keep track of associations between them in other to find the desired crossing configuration, and therefore gets a parsing complexity which depends on the rank of the grammar.

6.5 CONCLUSION

In this section, methods have been presented which are necessary for data-driven parsing of PSRCG. These include methods for improving the unbinarized grammar, a binarization algorithm for head-outward binarization and markovization, and a probabilistic parser for PSRCG with outside score estimates for speed-up, to be used in the Weighted Deductive Parsing framework.

Many interesting points remain open for future research.

- One path which should be investigated is the benefit of the binarization algorithms which aim at minimizing arity and rank of clauses for data-driven parsing.

- Well-nestedness is a property which captures almost all trees with discontinuities (see the previous chapter); and disallowing well-nestedness allows for more efficient parsing. To allow well-nestedness nevertheless, one could exploit the work of Gómez-Rodríguez et al. (2011); or one could try to use the parser for well-nested grammars and add ill-nestedness in post-processing with techniques such as the ones presented on p. 116 for adding discontinuities.

- The Maximum Likelihood Estimate as used here also has potential for improvement. It is a requirement of the definition SRCG that predicate names be distinguished by their arities. From a linguistic point of view, however, one would prefer that the probabilistic model is not aware of them. A model in which expansions depend only on the symbols and not on the arity of the symbols

should be investigated. Another idea which should be pursued is the separate learning of ID and LP, as already explored by Levy (2005).

- The algorithm of Boyd (2007) for crossing branches resolution in NeGra provides an interesting perspective: Its output could possibly be used directly with a PCFG parser combined with a re-introduction of crossing branches in post-processing, or it could serve as a guide for the parsing with a SRCG, resp. LCFRS, in the sense of Barthélemy et al. (2001), or it could be used within a coarse-to-fine framework (p. 112).

7

PARSING DISCONTINUOUS CONSTITUENTS

In this chapter, I present an application of the parsing techniques from the previous chapter. I parse the German NeGra treebank with `rparse` (cf. section 6.3). In section 7.1, I present the evaluation procedures we use. In section 7.2, I present the treebank-specific preprocessing. Section 7.3 is dedicated to the experiments and the discussion of their results. In section 7.4 I present related work, section 7.5 concludes this chapter.

Material in this chapter has been published previously in Maier (2010) and Maier and Kallmeyer (2010).

7.1 EVALUATION

In the following, our evaluation metrics will be presented. They are all implemented within `rparse`.

7.1.1 *Bracket Scoring*

The first is *bracket scoring* (henceforth EVALB) (see section 4.2.2). While it has its known shortcomings, it also has the advantage that to a certain extent, we can compare our work to previous work on parsing NeGra. In the context of LCFRS, we compare sets of tuples of the form $[i, A, \vec{\rho}]$, where $i \in \mathbb{N}$ is a sentence number, A is a non-terminal in the derivation tree for sentence i and $\vec{\rho}$ is a range vector representing the ranges in the input string covered by A. One set is obtained from the parser output, and one from the corresponding treebank trees. We do not exclude or collapse labels (Black et al., 1991). Using these tuple sets, we compute labeled and unlabeled recall (LR/UR), precision (LP/UP), and the F_1 measure (LF_1/UF_1), with the same approach as the Evalb program (Sekine and Collins, 1997) (see again section 4.2.2). Note that range vectors must be an exact match, there is no partial credit for partially correct range vectors.

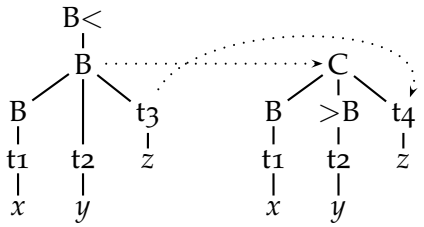


Figure 57: TDIST example

7.1.2 *Tree Edit Distance*

EVALB does not necessarily reflect parser output quality (Rehbein and van Genabith, 2007a; Emms, 2008; Kübler et al., 2008). One of its major problems is that attachment errors are penalized too hard. A wrong attachment may cause many other (otherwise correct) spans to become wrong. As the second evaluation method, we therefore choose the *tree-distance measure* (henceforth TDIST) of Zhang and Shasha (1989), which levitates this problem. It has been proposed for parser evaluation by Emms (2008). TDIST is an ideal candidate for the evaluation of the output of a PLCFRS parser, since whether trees have crossing branches or not is not relevant to it.

Following Emms (2008), for two trees $v_K = (V_K, E_K, r_K)$ and $v_A = (V_A, E_A, r_A)$ to be compared, the basis for this metric are partial mappings $\sigma : V_K \to V_A$. For a so-called T-*mapping*, only those node mappings are considered which preserve left-to-right order and ancestry. Within these mappings, insertion, deletion, and swap operations are identified. They are represented by the respective sets $\mathcal{I} = \{v \in V_A \mid v \notin \mathit{range}(\sigma)\}$, $\mathcal{D} = \{v \in V_K \mid v \notin \mathit{domain}(\sigma)\}$, $\mathcal{S} = \{v \in V_K \mid \Lambda_{V_K}(v) \neq \Lambda_{V_A}(\sigma(v))\}$. Furthermore, we consider the set $\mathcal{M} = \{v \in V_A \mid \Lambda_{V_K}(v) = \Lambda_{V_A}(\sigma(v))\}$ representing the matched nodes. The cost of a T-mapping is the total number of these operations, i.e. $|\mathcal{I}| + |\mathcal{D}| + |\mathcal{S}|$. The tree distance between two trees v_K and v_A is the cost of the cheapest T-mapping.

Consider figure 57, borrowed from Emms, as an example for a T-mapping. Inserted nodes are prefixed with $>$, deleted nodes are suffixed with $<$, and swapped nodes are linked with arrows. Since in total, four operations are involved, to this T-mapping, a cost of 4 is assigned. For the computation of TDIST, we exploit the algorithm of

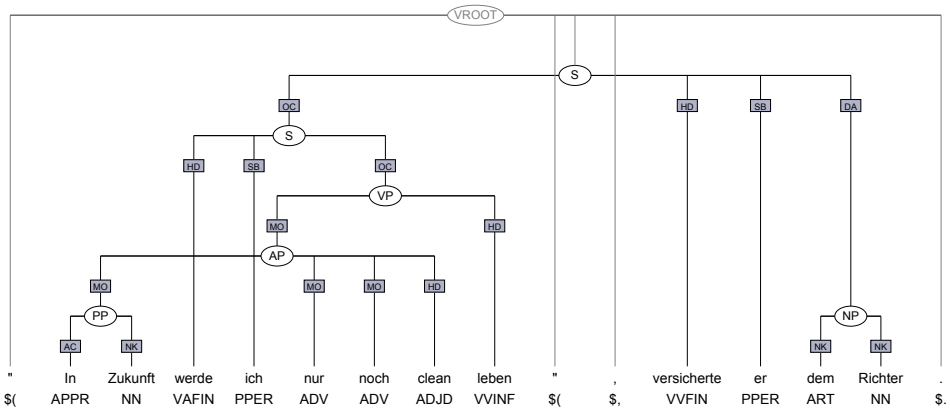


Figure 58: NeGra tree

Klein (1998), as it is presented in section 3.2.1 of Bille (2005). The algorithm runs in polynomial time in the size of the treebanks which are to be compared. We evaluate whole trees (not roof trees, see Emms (2008)); furthermore, in order to convert the tree distance measure into a similarity measure like EVALB, we use the macro-averaged Jaccard normalization as defined by Emms. For a T-mapping σ from υ_K to υ_A and the sets $\mathcal{D}$, $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{M}$ as introduced above, we compute it as follows.

$$jaccard(\sigma) = 1 - \frac{|\mathcal{D}| + |\mathcal{I}| + |\mathcal{S}|}{|\mathcal{D}| + |\mathcal{I}| + |\mathcal{S}| + |\mathcal{M}|}$$

where, in order to achieve macro-averaging, the numerators and denominators are summed over all tree pairs before the division.

7.2 TREEBANK-SPECIFIC PREPROCESSING

We preprocess NeGra in accordance with the usual practices known from the literature. These will be explained here. Figure 58 shows the NeGra annotation of (21). We will use this tree as running example.

(21) "In Zukunft werde ich nur noch clean leben, versicherte er dem Richter.
"In future will I only still clean live, assured he the judge.
"In the future, the only thing I will do is to live clean", he assured the judge.

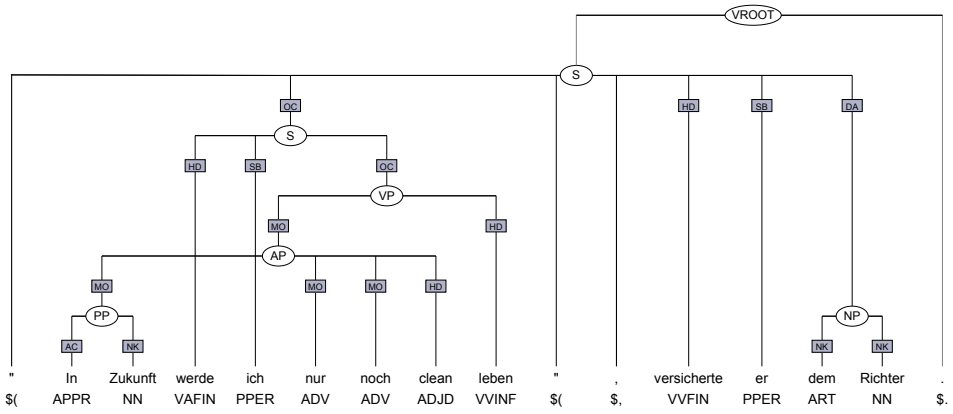


Figure 59: NeGra tree with attached punctuation

7.2.1 *Punctuation*

In NeGra, punctuation is not included in the annotation. It is simply attached to the virtual root node. This is not satisfactory for parsing, since it augments the arity of the extracted grammar, and furthermore makes few sense from a linguistic point of view (in the Penn Treebank (PTB), punctuation is included in the annotation). We use a two-pass algorithm for punctuation attachment. In the first pass, we attach all punctuation as high as possible in the tree, but as low as necessary in order to not introduce new crossing branches. In a second pass, we review the attachment of all punctuation nodes carrying the `$(` tag. This is punctuation which comes in pairs, such as quotation marks, parentheses, etc. The goal is to identify pairs and to attach both parts to the same level, preferably to the lower position of both. Note that it is very difficult to find satisfactory attachment positions for punctuation in a fully automatic fashion. Heuristics tend to always introduce new discontinuities in a fraction of the sentences. The two-pass algorithm we use, in this formulation due to Kilian Evang, is algorithm 5/6. As input, it expects a treebank $\Upsilon = \{(i, v_i, s_i) \mid 1 \leqslant i \leqslant m\}$, $m \in \mathbb{N}$. The function LOWER, algorithm 6, expects a punctuation node v and a node d being a candidate to which v could be lowered.

Figure 59 shows the tree from figure 58 with attached punctuation. Note that the first quotation mark is lowered to the S node in the second pass – the first pass does not touch it.

for all $(i, (V_i, E_i, r_i), s_{i,1}/p_{i,1} \cdots s_{i,n}/p_{i,n}) \in \Upsilon$ **do**
  for all $v \in V_i$ with $\pi(v) = \{j\}$ and $p_{i,j}$ is a punctuation POS tag **do**
    LOWER(v, j, r_i)
  end for
  Let $M = \emptyset$ the set holding possible left matches {*second pass starts here*}
  for all $l \in \{s_{i,1}, \ldots, s_{i,n}\}$ **do**
    for all $r \in M$ **do**
      if r is matching right part of paired punct. for l **then**
        if rightmost terminal dominated by parent of l is left adjacent to r **then**
          move r to be the rightmost child of the parent of l
        else
          if leftmost terminal dominated by parent of r is right adjacent to l **then**
            move l to be the leftmost child of the parent of r
          end if
        end if
        remove l from M and continue with next l
      end if
    end for
    if l is a potential left part of a paired punctuation **then**
      add l to M
    end if
  end for
end for

Algorithm 5: Punctuation attachment (NeGra)

```
function LOWER(v, j, d)
Let n ∈ ℕ be the number of children of d
for all children v_i of d, 1 ⩽ i ⩽ n do
  if j < min(π(v_i)) then
    move v to be the ith child of d
    break
  else
    if j < max(π(v_i)) then
      LOWER(v, j, v_i)
      break
    end if
  end if
end for
```

Algorithm 6: Punctuation attachment, function LOWER

7.2.2 *Resolving Crossing Branches*

To test the parser performance on PCFG, i. e., Probabilistic Simple 1-RCG, we create a version of NeGra with no crossing branches. For this task, we use the algorithm of Thorsten Brants' program, i. e., the one which is included in the NeGra distribution for this task. Due to a lack of publications on this program, the algorithm was obtained through reverse-engineering (Maier, 2004). Originally, the reverse-engineering was done to allow for PCFG parsing of TIGER. The original program does not resolve all crossing branches in TIGER.

The algorithm, which is formulated on top of the `export` format node numbering (cf. p. 102), traverses the tree bottom-up. At each non-terminal it determines the respective head[1] and moves as few children as possible (except the head) to higher positions in the tree such that all discontinuities of the node are resolved.

DEFINITION 7.1 (`export` numbering). Let $\upsilon = (V, E, r)$ be a constituency structure which is such that V contains at most 499 terminals and 500 non-terminals. An `export` numbering of υ is given by a function $f : V \to \mathbb{N}_0$ which is as follows.

1. $f(r) = 0$, and for all $v \in V \setminus \{r\}$,

1 Note that we do not use the head marking algorithm from 6.1.3. Instead, we mark as head of a node its leftmost child with the edge label `SC`. If there is no such child, then we mark as head its rightmost child with the edge label `NK`. If there is no such child we mark the leftmost child as head.

a) if $f_{out}(v) = 0$ and $\Lambda(v) = i$, then $f(v) = i$,

b) if $f_{out}(v) > 0$, then $f(v) \geqslant 500$ and $f(v) \leqslant 999$, and for all $500 \leqslant i < f(v)$, there is a $u \neq v$ in V with $f(u) = i$.

2. For all $v', v'' \in V$,

a) if $v' = v''$ then $f(v') = f(v'')$,

b) if $\langle v', v'' \rangle \in E^+$ then $f(v') > f(v'')$, and

c) if $\langle v'', v' \rangle \in E^+$ then $f(v') < f(v'')$.

For convenience, given a constituency structure (V, E, r), we introduce the function $f_p : V \to V$ which is undefined for $v = r$ and which yields the parent node of v for all $v \in V \setminus \{r\}$.

The algorithm expects a treebank $\Upsilon = \{(i, \upsilon_i, s_i) \mid 1 \leqslant i \leqslant m\}$, $m \in \mathbb{N}$, as input, with f_i being the `export` numbering for υ_i and f_i' being the inverse of f_i (i. e., the function which yields a node given a number). All trees $\upsilon_i = (V_i, E_i, r_i)$ are transformed by iterating through all nonterminals $v \in V_i$ in `export` order, i. e., such that for all $v_1, v_2 \in V_i$, v_1 is visited before v_2 iff $f_i(v_1) < f_i(v_2)$. If v has more than one yield block, then the children of v excluding its head daughter are considered for movement. The target node, i. e., the new parent node for those children of v which are to be moved is the node with the `export` number $max(\{f_1(v), f_2(v)\})$, where $f_1 : V \to \mathbb{N}$, $f_2 : V \to \mathbb{N}$ are two functions which are defined as follows.

1. For the computation of f_1 for some given node v, one regards the sequence of terminals from the leftmost terminal dominated by the parent of v to the leftmost terminal dominated by v itself and computes for each of the terminals the node with the lowest `export` number higher than the `export` number of the parent of v which dominates the terminal. $f_1(v)$ is then the node with the highest `export` number taken from this set of nodes. More formally, f_1 is computed as follows. First, let $V_t = \{t \in V \mid f_{out}(t) = 0\}$. For all $v \in V$, compute $V' = \{v' \in V_t \mid \Lambda(v') \geqslant min(\pi(f_p(v)))$ and $\Lambda(v') \leqslant min(\pi(v))\}$. Let $g : V' \to \mathbb{N}$ be a function such that for all $v' \in V'$, $g(v') = min(\{f_i(v'') \mid v'' \in V$ and there is a $\langle v'', v' \rangle \in E^*$ and $f_i(v'') > f_i(v)\})$. Finally, $f_1(v) = f_i'(max(\{g(v''') \mid v''' \in V'\}))$.

2. For some given node v, f_2 yields the node with the highest `export` number from the set of parent nodes of terminals in

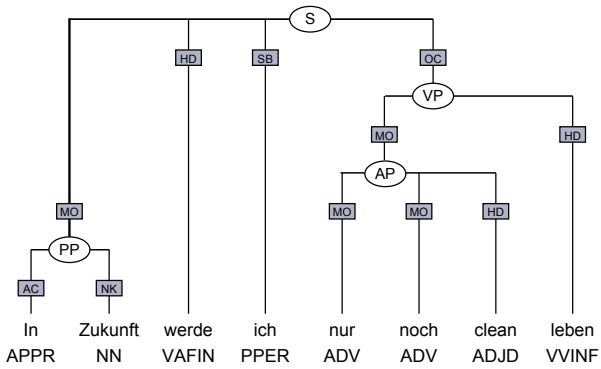


Figure 60: Resolving crossing branches in NeGra

the gaps of v. More formally, for all $v \in V$, let $V_{gaps} = \{v_{gap} \in V \mid f_{out}(v_{gap}) = 0$ and $\Lambda(v_{gap}) \notin \pi(v)$ and $\Lambda(v_{gap}) > min(\pi(v))$ and $\Lambda(v_{gap}) < max(\pi(v))\}$; finally $f_2(v) = f'_i(max(\{f_i(v_p) \mid v_p \in V$ and there is a $v_{gap} \in V_{gaps}$ with $\langle v_p, v_{gap}\rangle \in E\}))$.

While the two functions f_1 and f_2 may seem to be defined in a somewhat arbitrary way, note that they do mirror exactly the behavior of Brants' original program. Figure 60 shows the relevant part of the result of transforming the tree from figure 58.

We do not use the algorithm by Boyd (2007) here (cf. p. 102) since to our best knowledge, there are no parsing results we could compare ours too. However, Boyd's algorithm provides an interesting perspective for future work, as already mentioned in section 6.4.

7.2.3 *Grammar Annotation*

Grammar annotation (cf. section 6.1.1) has previously proven to be successful for German treebanks (Versley, 2005). For NeGra, we try two manually introduced splits of non-terminal labels. The grammatical function annotation provided with the edge labels already provides a finer-grained discrimination of the non-terminal labels. However, first, not all of them are equally informative, and second, using all of them leads to sparse data problems, even with PCFG parsing (Kübler, 2005). We therefore pick only some of the grammatical function labels.

SENTENCE-LEVEL REFINEMENT The S category is very coarse. Two types of sentences which can be clearly distinguished but carry

the same label S are relative clauses and (regular) clauses. In this split, we therefore change the label of all relative clause S (identifiable via the RC edge label) into S-RC.

NP REFINEMENT The NP category, which is very frequent, occurs in rather different contexts. We take advantage of the fact that its different grammatical functions are given in its edge labels and extend all NP labels with their edge labels.

7.2.4 *Head Rules*

For head-driven binarization, we must mark lexical heads (cf. section 6.1.3). As rules for this approach, we use the head rules for NeGra from the Stanford parser (Klein and Manning, 2003c) as obtained from `http://nlp.stanford.edu/software/lex-parser.shtml`. A complete listing of the rules can be found in Appendix A.

7.3 EXPERIMENTS

We now apply the parser to the actual data.

7.3.1 *Data*

Due to time and space constraints, we conduct all experiments in which we test different parameter combinations with a data set in which sentence length is restricted to not more than 30 words, called NeGra30. As a proof of feasibility, we also conduct one experiment with all sentences with a length up to 40 words. The corresponding data set is called NeGra40. According to common practice, for both sets, we use the first 90% for training and the remaining 10% as a test set. Table 9 shows the properties of the data (before splitting). Per-node values are values for non-terminal nodes only. The most important fact about the data sets is that a gap occurs in one out of about four sentences, but only in one out of about twenty nodes. While the former makes clear that gaps cannot be just ignored, the latter indicates that there could be a sparse data situation with respect to learning a probabilistic model.

	NEGRA30	NEGRA40
sentence len. limit	30	40
number of sentences	18,335	19,931
tokens	267,097	322,179
non-terminal nodes	127,302	151,331
av. sentence length	14.57	16.16
av. tree height	4.63	4.86
max. tree height	13	13
av. children per node	2.95	3.00
max. children per node	22	34
sentences with no gaps	13,842 (75.49%)	14,615 (73.33%)
av. gap degree per sent.	0.30	0.33
nodes with no gaps	120,902 (94.97%)	143,409 (94.77%)
av. gap degree per node	0.06	0.06
max. gap degree	6	8
well-nested sentences	17,937 (97.83%)	18,879 (97.36%)

Table 9: NeGra: Data sets for PSRCG

As for the properties of the SRCGs we extract from the data sets, from NEGRA30, we obtain a 7-SRCG with 16,599 clauses and 73 unique clause LHS predicate labels, while from NEGRA40 we obtain a 9-SRCG with 22,587 clauses and 134 unique clause LHS predicate labels. Note that the arities are higher than the ones in chapter 5 because we do not remove punctuation as we did before; inspite of the attachment algorithm presented before, achieving a lower arity is very difficult.

We conduct all experiments in parallel on a version of NeGra with resolved crossing branches, i. e., we use the same data set as before but with the transformation described in the previous section. These data sets will be called NEGRA30CF and NEGRA40CF. The main goal of this is to investigate the differences between PCFG and PSRCG parsing. For those experiments in which both behave similarly, we will therefore not elaborate on the context-free experiments.

7.3.2 *Parsing Results*

For the experiments, the newest Oracle Java 7 was used, running on a 3.16GHz Intel Xeon node. The virtual machine was provided with 7 GB per experiment unless noted otherwise.[2]

Binarization

We start out by investigating the different markovizations and binarization and the interplay between both.

UNARY VS. BINARY TOP AND BOTTOM CLAUSES At first, we investigate the effect of binary, resp. unary top and bottom clauses. For this, we perform experiments with NEGRA30 and NEGRA30CF for all four possible combinations, using OPTIMAL and markovization settings $v = 1, h = 2$. In a second turn, we repeat the four experiments with HEADOUTWARD, in a third turn with DETERM.

2 I am indebted to the Computation Services Department at the Faculty for Philosophy of the University of Düsseldorf for helping me with the access to this machine, as well as I am indebted to Jochen Saile for helping me with the access to machines of the Department for Computational Linguistics at the University of Tübingen for previous experiments. Thanks furthermore for helping me with experiments without red tape to Sandra Kübler, University of Indiana, and Ivelina Nikolova, Bulgarian Academy of Sciences.

For NEGRA30, the results lie very close together. Using OPTIMAL, we get 74.83 labeled F_1 for BINARYTOP with UNARYBOTTOM and 74.90 for BINARYTOP with BINARYBOTTOM. For UNARYTOP with UNARYBOTTOM, we get 74.80, while for UNARYTOP with BINARYBOTTOM , we get 74.87. Using HEADOUTWARD, we get 74.09 labeled F_1 for BINARYTOP with UNARYBOTTOM, 74.12 for BINARYTOP with BINARYBOTTOM, 74.07 for UNARYTOP with UNARYBOTTOM, and 74.11 for UNARYTOP with BINARYBOTTOM. The behavior of NEGRA30CF is very similar, but the bracket scoring gives better results (which is normal, given that parsing with an SRCG is a more difficult task than parsing with a CFG): For HEADOUTWARD, we get 76.00, resp. 75.97 labeled F_1 for the BINARYBOTTOM experiments (with BINARYTOP and UNARYTOP, respectively), and 75.82, resp. 75.77 for the UNARYBOTTOM experiments (again with BINARYTOP and UNARYTOP, respectively). OPTIMAL is a little better—for both BINARYBOTTOM experiments, we get 76.34, and for both the UNARYBOTTOM experiments we get 76.25.[3]

What is more interesting than the results is that for both OPTIMAL and HEADOUTWARD, UNARYBOTTOM produces about two to four times as many items as the BINARYBOTTOM. UNARYTOP in turn produces only slightly more items than BINARYTOP (around 10-15% more). DETERM, for which, of course, all results are identical (labeled F_1 72.10 for the SRCG and 74.85 for the CFG), behaves similarly, even though the difference between the number of produced items is even bigger (4.5 to 5 times as many items for UNARYBOTTOM vs. BINARYBOTTOM, same small difference for unary vs. binary top clauses). Again, also with respect to the numbers of items produced, the data without crossing branches behaves very similarly.

An explanation for the difference in the number of items is the fact that in order to arrive on top of a binarized clause, one must simply produce more items. Parsing time gets higher, not only because a bigger agenda has to be managed, but also because a bigger grammar has to be searched. As for the non-deterministic binarizations, having a unary clause at the bottom introduces a bigger factorization than the unary clause at the top. More factorization causes a bigger search space and more items. As the evaluation shows, the additional factorization is not even beneficial.

3 As for the experiments with the identical results, the output is not totally identical. There are differences in a small number of sentences; however, the number of matching brackets turns out to be identical.

	LP	LR	LF_1	UP	UR	UF_1
HEADOUTWARD	74.00	74.24	74.12	77.09	77.34	77.22
HEADOUTWARDKM	74.00	74.13	74.07	77.20	77.33	77.26
OPTIMAL	74.92	**74.88**	**74.90**	77.77	**77.73**	**77.75**
R-TO-L	74.94	74.44	74.69	77.71	77.20	77.45
L-TO-R	**75.08**	74.69	74.88	**77.95**	77.54	**77.75**
DETERM	72.40	71.80	72.10	75.67	75.04	75.35

Table 10: NEGRA30: Binarizations, BINARYBOTTOM

	LP	LR	LF_1	UP	UR	UF_1
HEADOUTWARD	73.77	74.41	74.09	76.90	77.57	77.23
HEADOUTWARDKM	74.15	74.58	74.36	77.41	**77.86**	77.64
OPTIMAL	74.70	**74.96**	**74.83**	**77.55**	77.82	**77.68**
R-TO-L	74.57	74.63	74.60	77.31	77.37	77.34
L-TO-R	**74.77**	74.84	74.81	77.51	77.59	77.55
DETERM	72.40	71.80	72.10	75.67	75.04	75.35

Table 11: NEGRA30: Binarizations, UNARYBOTTOM

BINARIZATION ORDERS We now investigate the performance of different binarization orders. Again, we set the markovization to $v = 1$ and $h = 2$ and perform experiments with NEGRA30 and NEGRA30CF. Since using BINARYTOP or UNARYTOP did not make much of a difference, except BINARYTOP being slightly faster, we limit ourselves to BINARYTOP, and combine it with both BINARYBOTTOM and UNARYBOTTOM. Tables 10 and 11 contain the results for NEGRA30.

For NEGRA30, all results for markovizing binarizations lie very close together (between 74.07 and 74.90 labeled F_1), DETERM is worse (72.10 labeled F_1). NEGRA30CF behaves almost identically, the results being about two points better (between 75.82 and 76.84 labeled F_1 for the markovizing binarizations, 74.85 for DETERM). Throughout, for both, BINARYBOTTOM is very slightly better than UNARYBOTTOM, and the best results are obtained with OPTIMAL and L-TO-R, resp. R-TO-L for NEGRA30CF.

The different binarization strategies result in a different number of produced items and therefore in different parsing speeds. Figure 61 depicts the numbers of items which are produced for each of the binarizations with NeGra30, for BinaryBottom (top) and UnaryBottom (bottom). We can make the following observations for NeGra30.

- The proximity of the results from the evaluation indicates that both BinaryBottom und UnaryBottom are about equally successful. Consequently, a higher number of items indicates that the search space is just traversed in an unfavorable way. As for the difference between BinaryBottom and UnaryBottom in terms of the number of items which is produced for each of the binarization orders, we see they behave similarly, with the difference that BinaryBottom produces very roughly about a third of the items of UnaryBottom.

- HeadOutward and HeadOutwardKM also behave very similarly. This indicates that the actual order of adding the sisters of the head does not make a big difference.

- The numbers of items produces by L-To-R and Determ are also fairly close together, especially for BinaryBottom. This is obviously due to the fact that both binarize the clauses strictly from left to right. Nevertheless, the deterministic strategy produces more items, while yielding worse results. This indicates that the presence or absence of markovization is more important for the results than the actual binarization order. The coverage is also improved by markovization. In the Determ experiment, 78 sentences had no parse, while in the other experiments, only 15 to 35 sentences had none.

For NeGra30CF, much less items are produced, and a much higher speed is achieved (for HeadOutward with UnaryBottom, 2 seconds per sentence instead 10 seconds with NeGra30). This reflects the lower parsing complexity. Furthermore, we can observe that the number of produced items for the different binarization orders lie closer together.

Markovization

For the investigation of different markovizations, we turn to the most successful settings so far: BinaryTop with BinaryBottom and Optimal. Table 12 contains the parsing results and the number of clauses

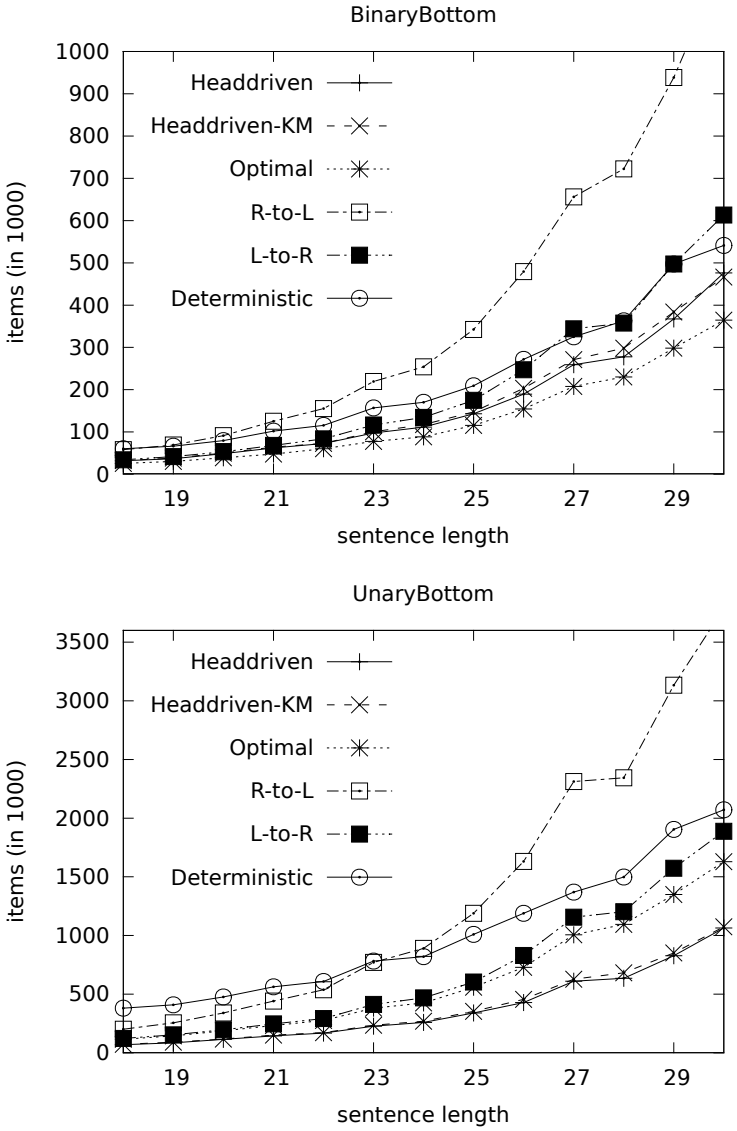


Figure 61: NEGRA30: Binarization items

V	H	LP	LR	LF1	UP	UR	UF1	CL.	SYM.
1	1	73.63	73.58	73.61	76.76	76.71	76.73	10,430	1,316
1	2	**74.92**	**74.88**	**74.90**	**77.77**	**77.73**	**77.75**	20,035	5,212
1	∞	71.98	71.97	71.97	75.32	75.31	75.32	34,260	17,686
2	1	73.91	73.75	73.83	76.95	76.78	76.86	21,231	3,991
2	2	74.66	74.58	74.62	77.41	77.33	77.37	34,110	11,789
2	∞	71.61	71.67	71.64	74.78	74.85	74.82	47,634	25,602
3	1	73.82	73.62	73.72	76.64	76.43	76.53	32,684	8,444
3	2	74.09	74.18	74.13	76.77	76.87	76.82	46,399	18,710
3	∞	71.17	71.42	71.29	74.43	74.68	74.55	59,386	32,550

Table 12: NEGRA30: Markovizations

and symbols in the binarized grammars for NEGRA30. The corresponding results and grammar properties for NEGRA30CF are contained in table 13.

For NEGRA30, among the different horizontal markovizations, $h = 2$ seems to produce the best results. Parent annotation ($v = 2$) and grandparent annotation ($v = 3$) actually let the parsing results deteriorate slightly, and additionally bloat the grammar. This is due to the flatness of the NeGra annotation. We get an average times of about 200, 40 and 20 sec. per sentence for all experiments with $h = 1$, $h = 2$, and $h = 3$, respectively; the different vertical markovizations vary among themselves only slightly. In other words, the parsing times depend rather on the horizontal than on the vertical markovization. For NEGRA30CF, the best results are actually obtained with parent annotation and $h = 2$.

On the basis of our results, we will adopt $v = 1$, $h = 2$ as the standard setting for further experiments with markovization: It offers the best trade-off between speed and results.

Grammar Annotation

To investigate the effect of grammar annotation (SPLIT), we continue to use BINARYTOP with BINARYBOTTOM, OPTIMAL, and markovization with $v = 1$ and $h = 2$. Table 14 contains the experimental results

v	h	LP	LR	LF_1	UP	UR	UF_1	CL.	SYM.
1	1	75.03	74.11	74.57	78.01	77.05	77.53	6,615	527
1	2	76.32	76.36	76.34	79.12	79.17	79.14	15,441	3023
1	∞	73.51	76.22	74.84	76.70	79.52	78.09	32,910	17,659
2	1	75.36	74.29	74.83	78.15	77.04	77.59	13,548	1,659
2	2	**76.95**	**76.77**	**76.86**	**79.48**	**79.30**	**79.39**	26,055	7,171
2	∞	73.67	76.23	74.92	76.60	79.26	77.90	43,119	23,667
3	1	75.42	74.44	74.93	78.03	77.02	77.52	21,213	3,882
3	2	76.39	76.46	76.42	78.93	79.00	78.97	35,022	11,725
3	∞	72.92	75.67	74.27	75.90	78.76	77.30	51,909	28,727

Table 13: NEGRA30CF: Markovizations

	LP	LR	LF_1	UP	UR	UF_1
Baseline	74.92	74.88	74.90	77.77	77.73	77.75
NP	75.21	74.95	75.08	78.16	77.88	78.02
S	75.81	**75.65**	75.73	78.31	78.15	78.23
NP ∘ S	**75.93**	75.57	**75.75**	**78.60**	**78.22**	**78.41**

Table 14: NEGRA30: Grammar annotation

for the baseline (OPTIMAL), the two grammar annotations from p. 194, both separately, and the combined annotation, on NEGRA30.

We see that the grammar annotations improve the results and that the combined annotations indeed combine the advantages of both. Furthermore, the numbers of produced items shown in figure 62 indicate that the grammar annotation helps to find a better parse more quickly.

For NEGRA30CF, both the results and the number of produced items show the same pattern. With NP ∘ S, we obtain a labeled F_1 of 77.80 vs. 76.34 for the baseline.

Outside Estimates

The computation of the context summary estimates in 6.2.2 is very demanding in terms of time and space requirements. We limit ourselves

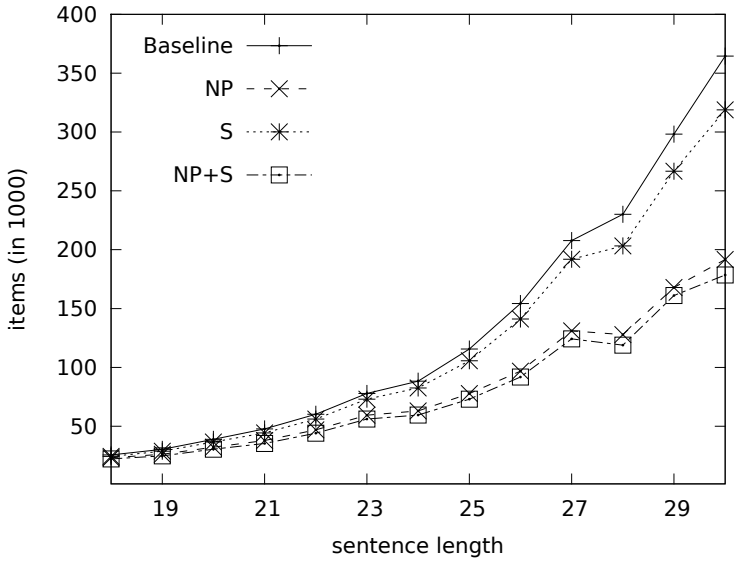


Figure 62: NeGra30: Grammar annotation items

to the two estimates which were computable at all, namely the LR estimate (p. 176) and the LN estimate (p. 177). While the former is non-monotonic and therefore gives us a best-first parser instead of an A^* parser, the latter is monotonic and allows for true A^* parsing.

On NeGra30 with Optimal, markovization $v = 1$ and $h = 2$, the results are 73.76 labeled F_1 for the LR estimates and 74.88 for the LN estimate. The parser output for the baseline differs by two brackets (within a single sentence), hence the result difference of 0.02 points. The two analyses of the sentence have the same probability, therefore, this is due to a different tie break at some point during parsing. As for the difference in the numbers of items which are produced, consider figure 63. For sentences of length 30, for instance, the LN estimate reduces the parsing time from an average of over 500 seconds to around 150 seconds; with the LR estimates, sentences of length 30 get parsed in average in around 50 seconds. The speedup for other binarization orders is even higher, as additional experiments have shown.

On NeGra30CF the behavior with the same settings is comparable in terms of speed gain. However, we obtain a labeled F_1 of 76.32 for both estimates vs. 76.34 for the baseline. The closeness of the result of the LR estimate to the baseline shows that the estimate acts more monotonous without the crossing branches. The parser output for the

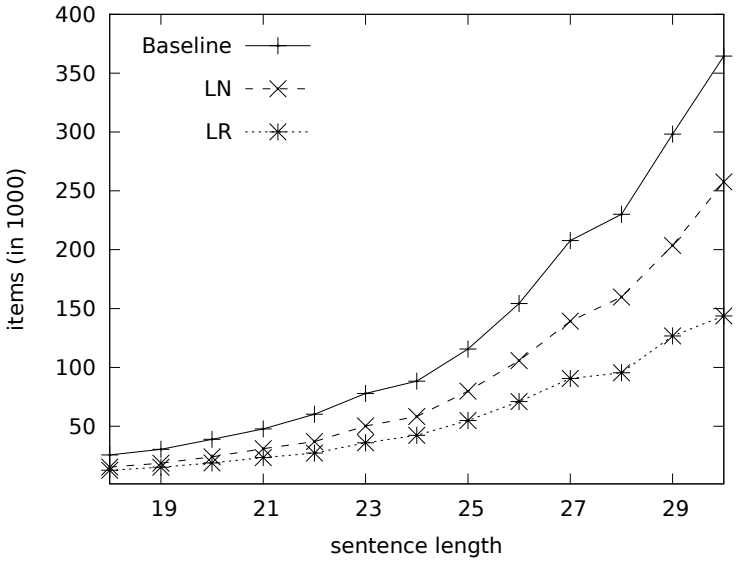


Figure 63: NEGRA30: Estimates

LN estimate and the baseline differ again by two brackets in a single sentence. Both analyses for the sentence have the same probability, therefore this is also due to a different tie break during parsing.

Cutoff, Gaps and Well-Nestedness

With the same settings, we now perform experiments with CUTOFF, GAPFILTER and ILL-NESTEDFILTER. Table 15 contains all results.

CUTOFF Cutoff causes a major drop in result quality. A look at the sizes of the grammars after applying the cutoff reveals the cause: After CUTOFF 1, the unbinarized grammar contains only 29% of the clauses of the original grammar (4,848 out of 16,599), after CUTOFF 2 18% (2,998), and after CUTOFF 3 13% (2,256). This shows that we are faced with extreme data sparseness. CUTOFF also drastically reduces parsing time (from an average of around 30 seconds per sentence to 25 sentences per second with CUTOFF 3).

As for the number of removed clauses with NEGRA30CF, we obtain the same picture. The baseline grammar contains 15,333 clauses and CUTOFF 1, 2 and 3 grammars contain 4,534, 2,811 and 2,127 clauses, respectively. Nevertheless the result quality does not deteriorate as

	LP	LR	LF_1	UP	UR	UF_1
Baseline	74.92	74.88	74.90	77.77	77.73	77.75
CUTOFF 1	71.99	67.83	69.85	75.27	70.91	73.03
CUTOFF 2	71.39	61.68	66.18	74.93	64.75	69.47
CUTOFF 3	70.71	57.28	63.29	74.31	60.19	66.51
GAPFILTER 0	73.77	73.29	73.53	77.03	76.53	76.78
GAPFILTER 1	74.11	73.72	73.92	77.17	76.76	76.96
GAPFILTER 2	74.26	73.92	74.09	77.26	76.92	77.09
GAPFILTER 5	74.90	74.87	74.88	77.80	77.77	77.78
GAPFILTER 10	74.90	74.87	74.88	77.80	77.77	77.78
ILL-NESTEDFILTER	74.97	74.83	74.90	77.87	77.72	77.80

Table 15: NEGRA30: Cutoff, gaps, well-nestedness

much as with NEGRA30. While we obtain 76.34 labeled F_1 with the baseline, we obtain 73.15, 70.78 and 68.61 with CUTOFF 1, 2 and 3, respectively. In any case, this indicates that Charniak's (1996) result does not generalize to PSRCG.

GAP FILTER With the GAPFILTER, we get results which are noticeably lower than the baseline when blocking all gaps and gaps longer than one, resp. two words. As for the number of items which gets blocked, when we use GAPFILTER 5, an effect is noticeable with sentences longer than 13 words, for which the number of blocked items climbs more or less proportional to the number of calls the range vector compatibility check method. The average parsing time for a single sentence is about ten seconds lower than with the baseline.

ILL-NESTEDNESS FILTER With ILL-NESTEDFILTER, labeled F_1 does not change compared to the baseline. We even obtain a sightly higher unlabeled F_1, indicating not only that ill-nested structures can not be recovered given our probabilistic model, but also that they can be detrimental for it. Nevertheless, an additional experiment with L-To-R binarization yielded results slightly higher than the baseline. This shows that additional experiments are necessary to determine the exact status of ill-nested structures. The amount of times the check of

range vector compatibility fails because the well-nestedness condition is not fulfilled is more or less constant over all sentence lengths > 3.

More Data

We now choose parser settings which allow for a compromise between speed and result quality inclining more towards speed. These are a $v = 1, h = 2$ markovization with HEADOUTWARD and BINARYBOTTOM, BINARYTOP, both category splits, GAPFILTER with 10, and the LN estimate. With those settings, we try to parse NEGRA40 as a test of mere feasibility. We also parse NEGRA40CF with the same settings, for comparison.

As results for NEGRA40, we obtain a labeled F_1 of 74.06 (LP 73.98, LR 74.14). In comparison, on 30 words the results are 75.10 F_1, LP 75.19 and LR 75.02. For the 40 words, the memory provided to the Java Virtual Machine had to be increased from 7 to 14 GB. The difference in average speed is noticeable: While with 30 words, the average parsing time for a sentence is 0.4 seconds, with 40 words it is 6.8 seconds. For NEGRA40CF, the distance between the results is comparable to the previous experiments: We get a labeled F_1 of 75.98 (LP 75.76, LR 76.20) with an average parsing time of 3 seconds per sentence. The results for NEGRA30CF with identical settings are labeled F_1 77.28, LP 77.75 and LR 77.75, with an average parsing time of 0.6 seconds per sentence.[4]

Figure 64 shows the numbers of items which are produced. These numbers explain the higher time. Additionally, with sentences over 30 words, we can see very clearly the effect of the higher parsing complexity of NEGRA40 compared to NEGRA40CF.

Alternative Evaluation with Tree Edit Distance

Bracket scoring is not a fair evaluation measure (see section 7.1). In order to show the possibilities of a different evaluation technique, we re-evaluate the SPLIT experiments and the corresponding DETERM experiment with TDIST. Table 16 shows the results for both NEGRA30 and NEGRA30CF.

4 While those parsing times might seem long in comparison with other PCFG parsers on the market, note that this is due to the fact that the parser engine is optimized for PSRCG parsing. For PCFG parsing, many of the data structures are not needed and a much more compact implementation would be possible.

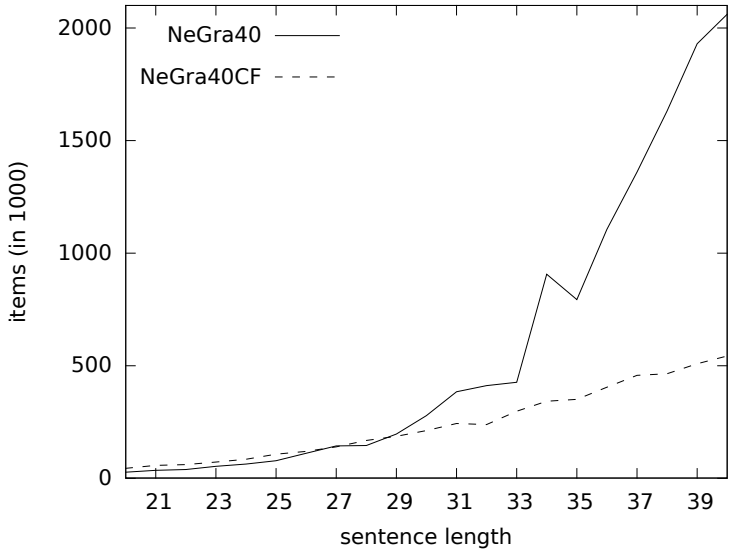


Figure 64: Results for sentences with a length $\leqslant 40$

	DETERM	BASEL.	NP	S	NP ∘ S
NEGRA30	79.79	80.77	81.19	81.37	81.53
NEGRA30CF	85.82	86.70	87.03	87.19	87.41

Table 16: NEGRA30 and NEGRA30CF: TDIST

While for all experiments, it is the case that an experiment with a lower bracket score than another one also has a lower TDIST score, we see that the TDIST results lie much closer together than the bracket scoring results. This shows the importance of a point which remains open for future work, namely the implementation of significance test. While it is not the case that there is no work on significance tests in parsing (Yeh, 2000; Bikel, 2001), using them is unfortunately far from being standard in the statistical parsing community.

7.3.3 *Discussion*

Our experiments have shown that direct data-driven parsing of discontinuous constituents is indeed possible, and gives good results. We have seen that several techniques known from PCFG parsing can straight-forwardly be transferred, and that most of them have a comparable effect in PSRCG parsing.

Taking the unmodified grammar (through deterministic binarization) as a baseline, we have seen that only with certain markovization settings and with grammar annotation, we get a substantial improvement in result quality while maintaining a reasonable speed. Other techniques which extend the binarized grammar such that it generates more only increase the number of items and thereby the parsing time while having little or no positive effect on the result quality. This holds for many techniques which might seem a good idea to try, such as the unary top and bottom clauses in the binarization, binarization orders with less directionality than others, etc.

The major reason for this problematic behavior seems to be the combinatorial explosion caused by different predicate arities, resp. block degrees of non-terminals, combined with the fact that about 98% of all constituents in the original treebank are continuous. We simply suffer from a sparse data situation: While we can already learn a usable model even from the unmodified treebank grammar, we must very carefully choose in which direction we extend the binarized grammar. A small extension of the binarized grammar might add many relevant events, but it can also, at the same time, add a huge amount of irrelevant events to the model, blowing up the search space such that the only effect remains a higher parsing time.

Future research on data-driven parsing should therefore concentrate on methods for splitting the problem of learning a probability model

in several parts, for example in learning the block degrees separately from the labels themselves, or, as Levy (2005) suggests, learning ID separately from LP. The use of unsupervised methods seems to be in order; this has also already been remarked by Levy. Another interesting perspective is the combination of a grammar resulting from Boyd's transformation (see p. 102) with guided parsing (see p. 85), and the use of coarse-to-fine parsing (see p. 112).

7.4 RELATED WORK

How does our approach perform in comparison with previous work in data-driven parsing?

The results from LCFRS parsing are not directly comparable with PCFG parsing results, since LCFRS parsing is a harder task. However, we can compare them to our own PCFG parsing results. Furthermore, since the bracket scoring metric coincides for constituents without crossing branches, to a certain extent, a comparison is possible. In order to place our results in the context of previous work on parsing NeGra, we cite some of the results from the literature which were obtained using PCFG parsers with bracket scoring. Note that these results were obtained on sentences with a length of $\leqslant 40$ words and that those parser possibly would deliver better results if tested on shorter sentences. Kübler (2005) (table 1, plain PCFG) obtains 69.4, Dubey and Keller (2003) (table 5, sister-head PCFG model) 71.12, Rafferty and Manning (2008) (table 2, Stanford parser with markovization $v = 2$ and $h = 1$) 77.2, and Petrov and Klein (2007) (table 1, Berkeley parser) 80.1.

As for the works which aim at creating crossing branches, Plaehn (2004) obtains 73.16 labeled F_1 with his Probabilistic Discontinuous Phrase Structure Grammar approach, albeit only on sentences with a length of up to 15 words. On those sentences, we obtain 84.59. Levy (2005) does not provide any evaluation results of his work. van Cranenburgh et al. (2011) have also followed up on our work. They introduce an integration of our approach with Data-Oriented Parsing (DOP), producing results which slightly exceed ours.

As for results on parsing treebanks other than NeGra, Evang and Kallmeyer (2011) followed up on our work. They transform the Penn Treebank such that the trace nodes and co-indexation are converted

to crossing branches and parse them with our parser. Their results compare favorably to the ones from the literature.

7.5 CONCLUSION

We have seen that data-driven direct parsing of discontinuous constituents using Probabilistic Simple Range Concatenation Grammar (SRCG) is feasible and yields competitive results. When comparing this approach to PCFG parsing results, one has to keep in mind that LCFRS parse trees contain non-context-free information about discontinuities. Therefore, a correct parse with our grammar is actually more informative than a correct CFG parse, evaluated with respect to a transformation of NeGra into a context-free treebank where precisely this information gets lost.

A major difficulty of direct data-driven parsing of discontinuous constituents is that the discontinuous structures themselves are important (they occur in 75% of all sentences, not only in NeGra but also in the PTB), but not frequent enough to learn them with a straightforward Maximum-Likelihood approach (only about 2% of all constituents in NeGra are discontinuous). Combined with the linguistically undesired fact that in SRCG, by definition, we must distinguish predicates by their arities, we run into severe sparse data problems. In future work, as already noted in the discussion section, we therefore aim at finding a method of splitting the learning of the probability model in several parts.

8

PARSING NON-PROJECTIVE DEPENDENCIES

In this chapter, we use our parser for grammar-based parsing of the Czech Prague Dependency Treebank (PDT). In section 8.1, I present the evaluation procedures for dependency parsing. In section 8.2, I present the treebank-specific preprocessing. Section 8.3 is dedicated to the experiments and the discussion of their results. In section 8.4 I present related work, and section 8.5 concludes this chapter.

Material in this chapter has been published previouly in Maier and Kallmeyer (2010).

8.1 EVALUATION

We use the standard metrics which have been used in the literature for the evaluation of dependency parses, namely *labeled attachment score (LAS)* and *unlabeled attachment score (UAS)*, as well as the percentage of *completely correct sentences including labels (LCC)* and *completely correct sentences not including labels (UCC)*. UAS, resp. LAS is computed as the percentage of unlabeled, resp. labeled edges in the gold standard which occur in the intersection of unlabeled, resp. labeled edges of the gold standard and the parser output. More precisely, an unlabeled edge consists of a sentence number, a node label (i. e., an integer) and the label of the head of the node (i. e., also an integer). A labeled edge additionally contains the dependency relation label between the node and its head. Let K_L and K_U be the set of labeled, resp. unlabeled edges of the gold standard and let A_L and A_U be the set of labeled, resp. unlabeled edges of the parser output. LAS is computed as $\frac{|A_L \cap K_L|}{|K_L|} * 100$, UAS is computed as $\frac{|A_U \cap K_U|}{|K_U|} * 100$. LCC, resp. UCC is computed as the percentage of sentences for which the sets of labeled, resp. unlabeled edges in gold standard and parser output are equal.

8.2 TREEBANK-SPECIFIC PREPROCESSING

We perform experiments with the PDT, release 2.0 (Hajič et al., 2000).

Annotation Format

The annotation of the PDT has already been presented in section 4.3.1. It has three annotation layers. The first layer contains morphological information (and the sentence itself) (*m-layer*), the second layer contains the dependency structures themselves (*a-layer*), and the third layer (*tectogrammatical layer* or *t-layer*) contains further non-local relations. The treebank annotation is provided in a stand-off XML format, i. e., all annotation layers are provided in separate files. One annotation file, resp. one set of annotation files contains several sentences. The linking of the files is done with the XML ID, resp. IDREF directives.

We thus read POS tagged sentences from an m-layer file and the corresponding dependency structures from the corresponding a-layer file simultaneously and reconstruct the dependency structures in a format suitable for the parser. Since using the complete morphological information for every word would result in very sparse data, we use only the first two letters of the morph tag as POS tag.[1]

Punctuation

In the PDT, punctuation is attached to the dependency structures (mainly explained by Hajičová et al. (1999) on pp. 188). We rely on this attachment and perform no further modification.

8.3 EXPERIMENTS

We now apply the parser to the actual data.

8.3.1 *Data*

Data Sets

The data of the PDT 2.0 is readily partitioned into a training set, a development set, and a test set. We discard the development set and use the other two for training and parsing. Since, as it turns out, dependency parsing experiments are more demanding in terms of computa-

1 Collins et al. (1999) only use the first letter. We use two letters to bring our data closer to the properties of our constituency data.

	PDT20
sentence len. limit	20
number of sentences	13,219
number of nodes	145,454
av. sentence length	11.00
av. edges	9.07
sentences with no gaps	10,923 (82.63%)
av. gap degree per sent.	0.18
nodes with no gaps	142,706 (98.11%)
av. gap degree per node	0.02
max. gap degree	2
well-nested sentences	13,211 (99.94%)

Table 17: PDT: Data Set

tional resources, we limit ourselves to sentences of a maximum length of 20 words. In the training set, we take the first five sections of the PDT training set and exclude all sentences longer than 20 words. For testing, we take the PDT test set, remove all sentences longer than 20 words, and then shorten it such that we get a 90/10 ratio, just as for the constituency parsing experiments. Table 17 shows the properties of the resulting data set (training and test set combined). Note the similarity of the ratios of sentences, resp. nodes with gaps and the ratio of well-nested sentences to the corresponding ratios of the constituency data in table 9, p. 196.

Grammar Extraction

Given the grammars produced by the grammar extraction algorithm as presented in section 5.2.2, the parser does not produce dependency structures as in definition 2.11, p. 21. This is because it does not extract a clause which corresponds to the root node of the dependency structure, due to the fact that the root node has no head, i. e., no incoming edge. In order to nevertheless extract a clause for this node, we pretend that every root node has an incoming edge labeled TOP and extract the corresponding clause as usual.[2] For the dependency struc-

2 In fact, TOP corresponds to the virtual root node in our constituency treebanks.

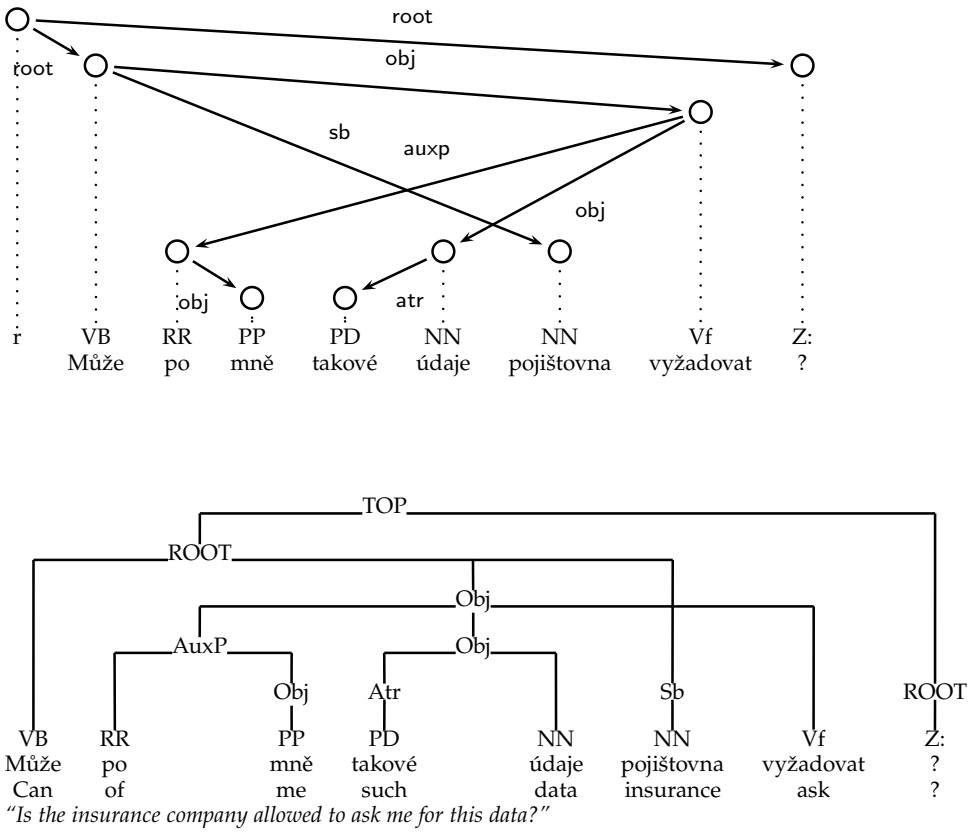


Figure 65: Dependency parser output

ture in figure 65, this would be the clause $\text{TOP}(X_1X_2) \rightarrow \text{ROOT}(X_1)$ $\text{ROOT}(X_2)$. As for the properties of the simple RCG we extract from the data set, we obtain a 3-SRCG with 11,099 clauses and 50 unique LHS predicate labels of clauses.

8.3.2 *Parsing Results*

Output Conversion

We need an additional conversion procedure for the parser output in order to obtain proper dependency structures. Figure 65 shows an example. The upper structure is the one we want to obtain from the lower one. The conversion is done as follows. As the terminals/POS tags of the target dependency structure, we take the terminals/POS tags of the parser output structure. Since a single non-terminal node

in the parser output structure represents exactly a single dependency relation, and since every dependency relation has exactly one head, every non-terminal must have exactly one (pre-)terminal child, which is the head of the corresponding dependency relation. To build the target structure, we traverse the parser output structure. For each node v we encounter, we build a dependency relation r, where

- as the label of r, we choose the label of v,
- as the head of r, we choose the (pre-)terminal child of the parent node of v, and
- as the dependent of r, we choose the (pre-)terminal child of v itself.

Inversely to the extension of the grammar extraction algorithm, we eventually discard the incoming TOP edge to the root node of the resulting dependency structure.

We perform again experiments with different settings in order to determine their effect. Since the grammars we extract from the dependency treebanks have rather different properties than the grammars we have seen in the last chapter, we must proceed slightly differently.

Experimental Results

Since for constituency parsing, we have achieved the biggest improvement by markovizing the binarized grammar, we first investigate the effect of different markovization settings and compare them with DETERM. We use HEADOUTWARD combined with BINARYTOP and BINARYBOTTOM since those options resulted in the fastest parsing speed for constituents. Table 18 shows the results, together with the number of clauses and symbols of the respective binarized grammars and the average number of items produced during parsing. With DETERM, we obtain 60.43/50.29 (UAS/LAS) and 18.83/11.82 (UCC/LCC); the corresponding binarized grammar contains 31,832 clauses and 20,839 symbols, and the parser produces 483,696 items in average.

The results lie much closer together than for constituency parsing. There is no substantial improvement with any type of markovization. In order to see if a change of the binarization settings is more effective, we also compare the different settings HEADOUTWARD, HEADOUTWARDKM, OPTIMAL, R-TO-L, L-TO-R, and DETERM. Table 19 shows

v	h	UAS	LAS	UCC	LCC	CL.	SYMB.	ITEMS
1	1	59.89	50.32	17.77	11.49	8,014	597	190,715
1	2	60.54	**50.85**	18.17	**12.14**	13,171	2,663	230,521
1	∞	**60.76**	50.58	**18.91**	11.82	20,300	8,920	133,367
2	1	57.81	48.12	17.93	11.49	14,112	2,085	620,470
2	2	59.51	49.80	18.09	**12.14**	19,542	5,865	831,272
2	∞	59.83	49.41	18.58	11.49	26,066	12,862	783,123
3	1	58.19	48.21	17.60	11.41	18,352	4,050	989,794
3	2	59.33	49.57	18.09	12.06	23,496	8,314	1,192,442
3	∞	59.88	49.40	18.74	11.49	29,637	14,951	1,147,816

Table 18: PDT20: Markovizations

the corresponding results. Just as with the different markovizations, there is a variance of less than one point in the attachment scores. The biggest difference in UAS is 60.76 for HEADOUTWARD vs. 60.17 for L-To-R; for LAS, it is 50.07 for HEADOUTWARDKM vs. 50.58 for HEADOUTWARD.

To get an impression where the results are situated in comparison with a state-of-the-art parser, we parse our data with the MST parser (McDonald et al., 2005) (see p. 124). Using the standard settings with the non-projective decoder, the results are 81.26/72.36 UAS/LAS and 35.70/21.76 UCC/LCC, i. e., almost 20 points higher than our results. Furthermore, training and parsing together required less than five minutes for all 13,219 sentences, while with `rparse` using a $v = 1, h = \infty$ markovization, the total required time was 2.5 hours.

8.3.3 *Discussion*

The most immediate observation that we can make is that on grammars extracted from dependency structures, markovization does not have the positive effect we have observed with the grammars extracted from constituency structures. In fact, markovization barely changes the result at all with respect to DETERM. It is even the case that the best results are obtained with the markovized grammar which most closely resembles the original grammar ($v = 1, h \geqslant 2$). The sizes of

	UAS	LAS	UCC	LCC
HEADOUTWARD	**60.76**	**50.58**	**18.91**	11.82
HEADOUTWARDKM	60.11	50.07	18.50	**11.90**
OPTIMAL	60.28	50.21	**18.91**	11.82
R-TO-L	60.60	50.53	**18.91**	11.65
L-TO-R	60.17	50.10	18.74	11.74
DETERM	60.43	50.29	18.83	11.82

Table 19: PDT20: Binarizations

the markovized grammars vary much, though, and cause a very high parsing time, particularly with high vertical markovizations. The average time for sentence of length 20 is 40.71 seconds, while for the a bigger deterministically binarized grammar extracted from NeGra (over sentences with a maximum length of 30), sentences of the same length only take slightly over a second to parse. The different binarization orders also barely cause an effect.

The fact that the results are almost identical across all different binarizations and markovizations indicates that there is a systemic problem with the way the grammar is extracted. At least for the baseline approach, the advantage of grammar-based parsing of non-projective dependencies, namely, the ability of going beyond edge-factored models, is outweighed by the disadvantage of much worse results and a much higher parsing time (cf. our comparison experiment with the MST parser, and the next section). A possible remedy can be found in the literature. By adapting the extended dependency-to-lexicalized-tree conversion of Collins et al. (1999), especially with respect to their way of choosing the node labels of the converted tree, one could most likely get better results. Another remedy could consist of incorporating structure-global information, e. g., through k-best parsing. In any case, one would then still have to deal with the high complexity of full PSRCG parsing.

8.4 RELATED WORK

As already mentioned, there is ample work on non-projective dependency parsing in the literature. However, to my knowledge, none of it

is grammar-based. The results which have been achieved on the (complete) Prague Dependency Treebank with state-of-the-art parsers are similar to the ones we obtained on our reduced data set with the MST parser: Nivre and Nilsson (2005) achieve UAS/LAS 80.1/72.8 and UCC/LCC 31.8/22.4 with the pseudo-projective (cf. p. 5.4.1) variant of the MALT parser; McDonald and Pereira (2006) report an accuracy of 85.2, resp. 35.9% completely correct sentences for the best setting of the MST parser on the complete PDT with the entire predefined test and training sets, using the POS tag reduction of Collins et al. (1999). No results on the PDT are reported for the easy-first parser of Goldberg and Elhadad (2010). However, the results they report for English data (Penn Treebank (PTB) section 23 and CoNLL English test set) lie between the MST parser results and the MALT parser results for this data.[3]

8.5 CONCLUSION

This chapter was dedicated to the presentation of the first results of grammar-based non-projective dependency parsing in the literature.

The general conclusion from our experiments is that even though there are good formal reasons for using Simple Range Concatenation Grammar (SRCG), resp. Linear Context-Free Rewriting System (LCFRS) for grammar-based non-projective dependency parsing (see the reasoning of Kuhlmann and Satta (2009)), in practice, this is a rather difficult enterprise. The cost of parsing in terms of time and space requirements is much higher than with other current dependency parsers and the results are not even in the vicinity. The reason is that a straight-forward MLE-based probability model fails to capture features of dependency structures which are global to structures: The contexts modeled by SRCG clauses is too local. A modification of the extraction algorithm with respect to how the labels are chosen could possibly be more successful. In any case, parsing would still be much more complex than with other state-of-the-art parsers.

Apart from the tree modifications of Collins et al. (1999), in future work, optimizations for constituency parsing should be tested for their effectiveness for dependency parsing. This holds particularly for Directed Treebank Refinement (Ule, 2003). Another way to better results

3 Note that in none of this works, a sentence length limit is imposed.

could be the reranking of k-best lists, in order to incorporate structure-global features. In principle, nothing stands in the way of computing such lists and adapting a reranker.[4] However, we would still face the high cost of parsing full SRCG. Last, since the well-nestedness constraint captures dependency structures equally well as constituency structures, just as constituency parsing, grammar-based dependency parsing would profit from a parser for a restricted variant of well-nestedness.

4 While with the algorithm of Pauls and Klein (2009), we face the same issues as with standard A*-parsing (see section 6.2.2), i. e., it should be feasible and fast. Computing a complete chart and extracting the k-best list afterwards (Huang and Chiang, 2005) is expensive but should not entail any other complications.

9 CONCLUSION

In previous chapters, I have already delivered detailed conclusions. In this chapter, I will summarize the main contributions of this thesis. I have aimed

A. at clarifying the status of discontinuous structures in both dependency and constituency treebank annotation,

B. at providing symbolic and probabilistic parsing techniques for formalisms which can model them, and

C. at confirming that the probabilistic parsing techniques are usable for data-driven parsing.

DISCONTINUOUS STRUCTURES IN TREEBANK ANNOTATION In chapter 5, I have analyzed annotation of constituency treebanks and dependency treebanks which account for discontinuities. I have introduced a grammar extraction algorithm for Simple Range Concatenation Grammar (SRCG) for discontinuous constituency treebanks and non-projective dependency treebanks. I have then shown for both how their degree of discontinuity can be characterized in terms of the two measures of *gap degree* and *well-nestedness*. In particular, I have shown that those measures which are known from recent literature on Dependency Grammar (DG) have a comparable descriptive value for constituency data. Furthermore, I have investigated to what extent treebank annotation exhibits synchronous rewriting and found that synchronous rewriting is a feature of treebank annotation which can be linguistically justified.

SYMBOLIC PARSING BEYOND CONTEXT-FREE GRAMMAR In chapter 3, I have presented an Earley-style parser for Range Concatenation Grammar (RCG), an incremental parsing strategy for SRCG and an experimental evaluation of both. Those parsers close important gaps in the literature. Furthermore, they have already proven useful in TuLiPA, a symbolic parser, which uses RCG or SRCG as pivot for-

malisms for parsing Tree-Adjoining Grammar (TAG) and variants of TAG.[1]

PROBABILISTIC PARSING BEYOND CONTEXT-FREE GRAMMAR In order to place my work on probabilistic parsing and data-driven parsing in the relevant context of the literature, in chapter 4 I have presented an introduction to probabilistic parsing and an overview of the corresponding literature, especially of those works which are concerned with the reconstruction of discontinuities. Out of this presentation, I have motivated the use of Probabilistic Simple Range Concatenation Grammar (PSRCG) for data-driven parsing. Furthermore, I have introduced relevant work in the field of data-driven dependency parsing and provided motivation for switching to a grammar-based approach.

In chapter 6, I have presented new techniques for direct data-driven parsing of discontinuous constituents using PSRCG with real-world sized data sets. These include binarization and markovization methods, a CYK style algorithm for weighted deductive parsing of PSRCG, and outside estimates for parse items used to speed up parsing, some of them allowing for true A^* parsing. The development of the techniques has brought together recent developments from different areas, such as research on parsing German and research on Linear Context-Free Rewriting System (LCFRS).

All techniques have been implemented in a single system. The implementation, which is called `rparse`, is freely available.[2] `rparse` is the first efficient parser for unrestricted PSRCG, resp. Probabilistic LCFRS. In fact, it is the first system which has successfully been used for direct data-driven parsing of discontinuous constituents with real-world sized data sets and also the first grammar-based parser for non-projective dependencies.

PARSING OF DISCONTINUOUS STRUCTURES In the chapters 7 and 8, I have applied my parser to discontinuous constituents and non-projective dependencies.

In chapter 8, I have applied my parser to the non-projective dependency structures of the Czech Prague Dependency Treebank. While the results do not reach the state of the art, they are the first reported

1 See `https://sourcesup.cru.fr/tulipa/`.

2 See `http://www.wolfgang-maier.net/rparse/`.

results for grammar-based parsing of non-projective dependencies. I have presented several ideas which could help to improve grammar-based data-driven parsing of non-projective constituents.

In chapter 7, I have applied my parser to the German NeGra treebank, a constituency treebanks with a direct annotation of discontinuities. The experiments have been evaluated with different evaluation measures. A discussion of the experimental results has been presented which, one the one hand, clarifies the interaction of the different parser parameters, and on the other hand highlights the differences between a PSRCG model and a Probabilistic Context-Free Grammar (PCFG) model. The experiments have proven that data-driven parsing with PSRCG is feasible at a reasonable parsing speed. An evaluation with bracket scoring has furthermore shown that the output quality of the parser lies in the range of the output quality of current PCFG parsers, while delivering more informative parses which contain discontinuous constituents. This shows that SRCG has a further potential and is worth to be explored, not only in parsing, but also beyond.

A

HEAD RULES FOR NEGRA

The following table lists the rules for head finding in NeGra (see sections 6.1.3 and 7.2.4). The rules are are minor modifications of the rules in the code of the Stanford parser (Klein and Manning, 2003c), as available from `http://nlp.stanford.edu/software/lex-parser.shtml`.

LABEL	DIRECTION	POTENTIAL HEADS
S	right-to-left	VVFIN VVIMP
S	right-to-left	VP CVP
S	right-to-left	VMFIN VAFIN VAIMP
S	right-to-left	S CS
VP	right-to-left	VVINF VVIZU VVPP
VP	right-to-left	VZ VAINF VMINF VMPP VAPP PP
VZ	right-to-left	VVINF VAINF VMINF VVFIN VVIZU
VZ	left-to-right	PRTZU APPR PTKZU
NP	right-to-left	NN NE MPN NP CNP PN CAR
AP	right-to-left	ADJD ADJA CAP AA ADV
PP	left-to-right	KOKOM APPR PROAV
CO	left-to-right	
AVP	right-to-left	ADV AVP ADJD PROAV PP
AA	right-to-left	ADJD ADJA
CNP	right-to-left	NN NE MPN NP CNP PN CAR
CAP	right-to-left	ADJD ADJA CAP AA ADV
CPP	right-to-left	APPR PROAV PP CPP
CS	right-to-left	S CS
CVP	right-to-left	VZ
CVZ	right-to-left	VZ
CAVP	right-to-left	ADV AVP ADJD PWAV APPR PTKVZ
MPN	right-to-left	NE FM CARD

NM	right-to-left	CARD NN
CAC	right-to-left	APPR AVP
CH	right-to-left	
MTA	right-to-left	ADJA ADJD NN
CCP	right-to-left	AVP
DL	left-to-right	
ISU	right-to-left	
QL	right-to-left	
-	right-to-left	PP
CD	right-to-left	CD
NN	right-to-left	NN
NR	right-to-left	NR
VROOT	left-to-right	.,

B

ACRONYMS

B.1 FORMALISMS

Many formslisms and frameworks have been mentioned in this thesis. I list them in the following, together with the acronyms I have used. Unless noted otherwise, the acronym of the probabilistic version of a formalisms is the acronym of the symbolic variant, with a "P" prepended; furthermore, the acronym for the languages produced by a formalism is the acronym of the formalism, with the "G" (for *grammar*) or "S" (for *system*) swapped with an "L" (for *language*), also unless noted otherwise.

CCFG	Coupled Context-Free Grammar
CCG	Combinatory Categorial Grammar
CFG	Context-Free Grammar
DG	Dependency Grammar
DPSG	Discontinuous Phrase Structure Grammar
FTAG	Feature Structure Based Tree-Adjoining Grammar
GCFG	Generalized Context-Free Grammar
GF	Grammatical Framework
GPSG	Generalized Phrase Structure Grammar
HG	Head Grammar
HPSG	Head-Driven Phrase Structure Grammar
ID/LP	Immediate Dominance/Linear Precedence
ITG	Inversion Transduction Grammar
LIG	Linear Indexed Grammar

LFG	Lexical Functional Grammar
LMG	Literal Movement Grammar
IG	Indexed Grammar
LCFRS	Linear Context-Free Rewriting System
LTAG	Lexicalized Tree-Adjoining Grammar
MCFG	Multiple Context-Free Grammar
MCTAG	Multi-Component Tree-Adjoining Grammar
MG	Minimalist Grammar
MTT	Meaning Text Theory
NRCG	Negative Range Concatenation Grammar
OSRCG	Ordered Simple Range Concatenation Grammar
PMCFG	Parallel Mulitple Context-Free Grammar[1]
PSG	Phrase Structure Grammar
RCG	Range Concatenation Grammar
SDTG	Syntax-Directed Transduction Grammars
SRCG	Simple Range Concatenation Grammar
TSG	Tree Substitution Grammar
TT-MCTAG	Multi-Component Tree-Adjoining Grammar with Tree Tuples
TAG	Tree-Adjoining Grammar
TIG	Tree Insertion Grammar
UCFG	Unordered Context-Free Grammar

1 Parallel Multiple Context-Free Grammar shares the acronym with Probabilistic Multiple Context-Free Grammar. In this work, however, PMCFG is only used once in the sense of Parallel MCFG (see p. 87), in all other places, it means Probabilistic MCFG.

VTAG	Vector Tree-Adjoining Grammar
WCDG	Weighted Constraint Dependency Grammar
WOSRCG	Wellnested Ordered Simple Range Concatenation Grammar
XDG	eXtensible Dependency Grammar

B.2 TREEBANKS

The following list contains all treebanks which have been mentioned, together with their acronyms.

BTB	BulTreebank
NeGra	NeGra Treebank
NeGra-Dep	NeGra dependency treebank (Daum et al., 2004)
PDT	Prague Dependency Treebank
PTB	Penn Treebank
TIGER	TIGER Treebank
TIGER-DP	TIGER dependency treebank (Forst et al., 2004)
TIGER-Dep	TIGER dependency treebank (Daum et al., 2004)
TüBa-D/Z	Tübingen Treebank of Written German

B.3 OTHER ACRONYMS

The following list contains other acronyms which have been used.

CNF	Chomsky Normal Form
CSP	Constraint Satisfaction Problem
DAG	Directed Acyclic Graph
DOP	Data-Oriented Parsing

FOM Figure-of-Merit

GES Generative Enumerative Syntax

MCS Mild Context-Sensitivity

MTS Model-Theoretic Syntax

POS Part-of-Speech

LHS left-hand side

RHS right-hand side

TA Thread Automata

VPP Valid Prefix Property

WDP Weighted Deductive Parsing

BIBLIOGRAPHY

Abeillé, A., Clément, L., and Toussenel, F. (2003). Building a treebank for French. In Abeillé, A., editor, *Treebanks*, chapter 10, pages 165–187. Kluwer, Dordrecht.

Aho, A. V. (1968). Indexed Grammars – An extension of Context-Free Grammars. *Journal of the ACM*, 15(4):647–671.

Angelov, K. (2009). Incremental parsing with Parallel Multiple Context-Free Grammars. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 69–76, Athens, Greece. Association for Computational Linguistics.

Baker, J. K. (1979). Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA.

Bangalore, S., Boullier, P., Nasr, A., Rambow, O., and Sagot, B. (2009). MICA: A probabilistic dependency parser based on Tree Insertion Grammars (application note). In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 185–188, Boulder, CO. Association for Computational Linguistics.

Barthélemy, F., Boullier, P., Deschamp, P., and Éric Villemonte de la Clergerie (2001). Guided parsing of Range Concatenation Languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 42–49, Toulouse, France. Association for Computational Linguistics.

Barton Jr., G. E. (1985). On the complexity of ID/LP parsing. *Computational Linguistics*, 11(4):205–218.

Becker, T., Joshi, A. K., and Rambow, O. (1991). Long-distance scrambling and Tree-Adjoining Grammars. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational*

Linguistics, pages 21–26, Berlin, Germany. Association for Computational Linguistics.

Becker, T., Rambow, O., and Niv, M. (1992). The derivational generative power of formal systems or scrambling is beyond LCFRS. IRCS report 92-38, University of Pennsylvania, Philadelphia, PA.

Beeri, C. and Ramakrishnan, R. (1991). On the power of magic. *Journal of Logic Programming*, 10(3-4):255–299.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton.

Bertsch, E. and Nederhof, M.-J. (2001). On the complexity of some extensions of RCG parsing. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, pages 66–77, Beijing, China.

Bies, A., Ferguson, M., Katz, K., McIntyre, R., Tredinnick, V., Kim, G., Marcinkiewicz, M. A., and Schasberger, B. (1995). Bracketing guidelines for the Treebank II Style Penn Treebank Project. `ftp://ftp.cis.upenn.edu/pub/treebank/doc/manual/`, December 4, 2010.

Bikel, D. M. (2001). Randomized parsing evaluation comparator (statistical significance tester for `evalb` output). `http://www.cis.upenn.edu/~dbikel/software.html#comparator`, February 19, 2012.

Bikel, D. M. and Chiang, D. (2000). Two statistical parsing models applied to the Chinese treebank. In *Proceedings of the Second Workshop on Chinese Language Processing at ACL 2000*, pages 1–6, Hong Kong. Association for Computational Linguistics.

Bille, P. (2005). A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239.

Billott, S. and Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, BC. Association for Computational Linguistics.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A

procedure for quantitatively comparing the syntactic coverage of English grammars. In Price, P., editor, *Fourth DARPA Speech and Natural Language Workshop*, pages 306–311, San Mateo. Morgan Kaufmann.

Blevins, J. P. (1990). *Syntactic complexity: Evidence for discontinuity and multidomination*. PhD thesis, University of Massachusetts, Amherst, MA.

Bockhorst, J. and Craven, M. (2001). Refining the structure of a stochastic Context-Free Grammar. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 1315–1320, Seattle, WA. Morgan Kaufmann.

Bod, R. (1993). Using an annotated corpus as a stochastic grammar. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 37–44, Utrecht, The Netherlands. Association for Computational Linguistics.

Bod, R. (1995). *Enriching Linguistics with Statistics: Performance Models of Natural Language*. Number 1995-14 in University of Amsterdam ILLC Dissertation Series. Academische Pers, Amsterdam, The Netherlands.

Bod, R. and Scha, R. (1996). Data-oriented language processing: An overview. Technical Report LP-96-13, Departement of Computational Linguistics, University of Amsterdam, Amsterdam, The Netherlands.

Bod, R., Scha, R., and Sima'an, K., editors (2003). *Data-Oriented Parsing*. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, CA.

Bodirsky, M., Kuhlmann, M., and Möhl, M. (2005). Well-nested drawings as models of syntactic structure. In *Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*, pages 195–203, Edinburgh, UK.

Booth, T. L. and Thomson, R. A. (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450.

Boullier, P. (1996). Another facet of LIG parsing. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 87–94, Santa Cruz, CA. Association for Computational Linguistics.

Boullier, P. (1998). Proposal for a Natural Language Processing syntactic backbone. Research Report 3342, INRIA-Rocquencourt, Rocquencourt, France.

Boullier, P. (1999). Chinese numbers, MIX, scrambling, and Range Concatenation Grammars. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 53–60, Bergen, Norway. Association for Computational Linguistics.

Boullier, P. (2000a). A cubic time extension of Context-Free Grammars. *Grammars*, 3:111–131.

Boullier, P. (2000b). Range Concatenation Grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, pages 53–64.

Boullier, P. and Deschamp, P. (1988). *Le système SYNTAX™ – manuel d'utilisation et de mise en oeuvre sous UNIX™*. `http://syntax.gforge.inria.fr/syntax3.8-manual.pdf`, January 4, 2012.

Boullier, P. and Sagot, B. (2009). Parsing directed acyclic graphs with Range Concatenation Grammars. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 254–265. Association for Computational Linguistics.

Boyd, A. (2007). Discontinuity revisited: An improved conversion to context-free representations. In *Proceedings of The Linguistic Annotation Workshop (LAW) at ACL 2007*, pages 41–44, Prague, Czech Republic. Association for Computational Linguistics.

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER Treebank. In Hinrichs, E. W. and Simov, K., editors, *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories*, pages 24–42, Sozopol, Bulgaria.

Brants, T. (1997). The NeGra Export format. CLAUS Report 98, Computational Linguistics Department, Saarland University, Saarbrücken, Germany.

Bresnan, J., editor (1982). *The mental representation of grammatical relations*. MIT Press, Cambridge.

Buch-Kromann, M. (2009). *Discontinuous Grammar – A dependency-based model of human parsing and language learning*. VDM Verlag.

Bunt, H. (1991). Parsing with Discontinuous Phrase Structure Grammar. In Tomita, M., editor, *Current Issues in Parsing Technology*, pages 49–63. Kluwer, Dordrecht.

Bunt, H. (1996). Formal tools for describing and processing discontinuous constituency structure. In Bunt, H. and van Horck, A., editors, *Discontinuous Constituency*, volume 6 of *Natural Language Processing*, pages 63–83. Mouton de Gruyter, Berlin.

Bunt, H., Thesingh, J., and van der Sloot, K. (1987). Discontinuous constituents in trees, rules and parsing. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, pages 203–210, Copenhagen, Denmark. Association for Computational Linguistics.

Burden, H. and Ljunglöf, P. (2005). Parsing Linear Context-Free Rewriting Systems. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 11–17, Vancouver, BC. Association for Computational Linguistics.

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J., and Way, A. (2004). Treebank-based acquisition of wide-coverage, probabilistic LFG resources: Project overview, results and evaluation. In *Workshop Beyond shallow analyses – Formalisms and statistical modeling for deep analyses at IJCNLP '04*, Sanya City, Hainan, China.

Butt, M., King, T., Niño, M.-E., and Segond, F. (1999). *A Grammar Writer's Cookbook*, volume 95 of *CSLI Lecture Notes*. CSLI Publications, Stanford, CA.

Cahill, A. (2004). *Parsing with Automatically Acquired, Wide-Coverage, Robust, Probabilistic LFG Approximations*. PhD thesis, Dublin City University, Dublin, Ireland.

Campbell, R. (2004). Using linguistic principles to recover empty categories. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 645–652, Barcelona, Spain. Association for Computational Linguistics.

Caraballo, S. A. and Charniak, E. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Champollion, L. (2011). Lexicalized Non-Local MCTAG with Dominance Links is NP-Complete. *Journal of Logic, Language and Information*, 20:343–359.

Charniak, E. (1996). Tree-bank grammars. Technical Report CS-96-02, Department of Computer Science, Brown University, Providence, RI.

Charniak, E. (1997). Statistical parsing with a Context-Free Grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 598–603.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139, Seattle, WA. Association for Computational Linguistics.

Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., and Vu, T. (2006). Multilevel coarse-to-fine PCFG parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 168–175, New York, NY. Association for Computational Linguistics.

Chen-Main, J. (2006). *On the generation and linearization of multi-dominance structures*. PhD thesis, Johns Hopkins University, Baltimore, MD.

Chen-Main, J. and Joshi, A. (2010). Unavoidable ill-nestedness in natural language and the adequacy of tree local-MCTAG induced dependency structures. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+10)*, New Haven, CT.

Chen-Main, J. and Joshi, A. (2012). A dependency perspective on the adequacy of Tree Local Multi-component Tree Adjoining Grammar. Manuscript.

Chi, Z. and Geman, S. (1998). Estimation of Probabilistic Context-Free Grammars. *Computational Linguistics*, 24(2):299–305.

Chiang, D. (2003). Statistical parsing with an automatically extracted Tree Adjoining Grammar. In Bod et al. (2003), pages 299–316.

Chiang, D. (2004). Uses and abuses of intersected languages. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pages 200–315.

Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IEEE Transactions*, 2(3):113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague, Paris.

Chomsky, N. (1981). *Lectures on Government and Binding: The Pisa Lectures*. Mouton de Gruyter, Berlin.

Chomsky, N. (1995). *The Minimalist Program*. MIT Press, Cambridge.

Chu, Y. and Liu, T. (1965). On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Clark, S., Hockenmaier, J., and Steedman, M. (2002). Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 327–334, Philadelphia, PA. Association for Computational Linguistics.

Cocke, J. and Schwartz, J. T. (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain. Association for Computational Linguistics.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, San Francisco, CA. Morgan Kaufmann.

Collins, M. and Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 263–270, Philadelphia, PA. Association for Computational Linguistics.

Collins, M., Hajič, J., Ramshaw, L., and Tillmann, C. (1999). A statistical parser for Czech. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 505–512, College Park, MD. Association for Computational Linguistics.

Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Corazza, A., Lavelli, A., and Satta, G. (2008). Measuring parsing difficulty across treebanks. Manuscript retrieved from `http://www.dei.unipd.it/~satta/publ/paper/ecc.pdf`, September 1, 2011.

Corazza, A., Lavelli, A., Satta, G., and Zanoli, R. (2004). Analyzing an Italian treebank with state-of-the-art statistical parsers. In *Third Workshop on Treebanks and Linguistic Theories (TLT-2004)*, Tübingen, Germany.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2003). *Introduction to Algorithms*. MIT Press, Cambridge, 2nd edition. 4th printing.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, Athens, GA.

Crescenzi, P., Gildea, D., Marino, A., Rossi, G., and Satta, G. (2011). Optimal head-driven parsing complexity for Linear Context-Free Rewriting Systems. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Daniels, M. W. (2005). *Generalized ID/LP Grammar: A Formalism for Parsing Linearization-Based HPSG Grammars*. PhD thesis, The Ohio State University, Columbus, OH.

Daum, M., Foth, K., and Menzel, W. (2004). Automatic transformation of phrase treebanks to dependency trees. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation*, Lisbon, Portugal. European Language Resources Association (ELRA).

Debusmann, R., Duchier, D., and Kruijff, G.-J. (2004). Extensible Dependency Grammar: A new methodology. In *Proceedings of the Workshop on Recent Advances in Dependency Grammars at COLING 2004*, Geneva, Switzerland.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38. Series B.

Dienes, P. and Dubey, A. (2003a). Antecedent recovery: Experiments with a trace tagger. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 33–40, Sapporo, Japan. Association for Computational Linguistics.

Dienes, P. and Dubey, A. (2003b). Deep syntactic processing by combining shallow methods. In *Proceedings of the 41th Annual Meeting of the Association for Computational Linguistics*, pages 431–438, Sapporo, Japan. Association for Computational Linguistics.

Dikovsky, A. and Modina, L. (2000). Dependencies on the other side of the curtain. *Traitement automatique des langues*, 41(1):67–96.

Dubey, A. and Keller, F. (2003). Probabilistic parsing for German using sisterhead dependencies. In *Proceedings of the 41th Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan. Association for Computational Linguistics.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Eisner, J. (1996a). Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86. Association for Computational Linguistics.

Eisner, J. (1996b). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING 1996: The 16th International Conference on Computational Linguistics*, volume 1, pages 340–345, Copenhagen, Denmark.

Eisner, J. and Satta, G. (1999). Efficient parsing for Bilexical Context-Free Grammars and Head Automaton Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–464, College Park, MD. Association for Computational Linguistics.

Emms, M. (2008). Tree distance and some other variants of Evalb. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, pages 1373–1379, Marrakech, Morocco. European Language Resources Association (ELRA).

Engel, U. (1988). *Deutsche Grammatik*. Groos, Heidelberg.

Evang, K. (2011). Parsing discontinuous constituents in English. Master's thesis, University of Tübingen, Tübingen, Germany.

Evang, K. and Kallmeyer, L. (2011). PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT 2011)*, pages 104–116, Dublin, Ireland.

Forst, M., Bertomeu, N., Crysmann, B., Fouvry, F., Hansen-Schirra, S., and Kordoni, V. (2004). Towards a dependency-based gold standard for German parsers: The TiGer Dependency Bank. In *Proceedings of The 5th International Workshop on Linguistically Interpreted Corpora (LINC-04) at COLING 2004*, pages 31–38, Geneva, Switzerland.

Foth, K. A., Daum, M., and Menzel, W. (2004). Interactive grammar development with WCDG. In *The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, pages 122–125, Barcelona, Spain. Association for Computational Linguistics.

Frank, A. (2001). Treebank conversion for LTAG grammar extraction. In *Proceedings of The 3rd Workshop on Linguistically Interpreted Corpora (LINC-2001) at Annual Meeting of Societas Linguistica Europaea*, Leuven, Belgium.

Frank, R. (2002). *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, Cambridge.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615.

Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and Control*, 8:304–337.

Gallo, G., Longo, G., Pallottino, S., and Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201.

Gardner, P. P. and Giegerich, R. (2004). A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5:140–148.

Gazdar, G. (1988). Applicability of Indexed Grammars to Natural Languages. In *Natural Language Parsing and Linguistic Theories*, pages 69–94. Reidel, Dordrecht.

Gazdar, G., Klein, E., Pullum, G. K., and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Gerdes, K. and Kahane, S. (2001). Word order in German: A formal dependency grammar using a topological hierachy. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 220–227, Toulouse, France. Association for Computational Linguistics.

Gildea, D. (2001). Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, Pittsburgh, PA.

Gildea, D. (2010). Optimal parsing strategies for Linear Context-Free Rewriting Systems. In *Human Language Technologies: The 2010 Annual*

Conference of the North American Chapter of the Association for Computational Linguistics, pages 769–776, Los Angeles, CA. Association for Computational Linguistics.

Gildea, D. (2011). Grammar factorization by tree decomposition. *Computational Linguistics*, 37(1):231–248.

Goldberg, Y. and Elhadad, M. (2010). An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, CA. Association for Computational Linguistics.

Gómez-Rodríguez, C., Carroll, J., and Weir, D. (2011). Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3):541–586.

Gómez-Rodríguez, C., Kuhlmann, M., and Satta, G. (2010). Efficient parsing of well-nested Linear Context-Free Rewriting Systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 276–284, Los Angeles, CA. Association for Computational Linguistics.

Gómez-Rodríguez, C., Kuhlmann, M., Satta, G., and Weir, D. (2009a). Optimal reduction of rule length in Linear Context-Free Rewriting Systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 539–547, Boulder, CO. Association for Computational Linguistics.

Gómez-Rodríguez, C. and Nivre, J. (2010). A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden. Association for Computational Linguistics.

Gómez-Rodríguez, C. and Satta, G. (2009). An optimal-time binarization algorithm for Linear Context-Free Rewriting Systems with fan-out two. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 985–993, Singapore. Association for Computational Linguistics.

Gómez-Rodríguez, C., Weir, D., and Carroll, J. (2009b). Parsing mildly non-projective dependency structures. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 291–299, Athens, Greece. Association for Computational Linguistics.

Goodman, J. (1998). *Parsing Inside-Out*. PhD thesis, Computer Science Group, Harvard University, Cambridge, MA. Technical Report TR-07-98.

Goodman, J. (2003). Efficient parsing of DOP with PCFG-reductions. In Bod et al. (2003).

Gorn, S. (1967). Explicit definitions and linguistic dominoes. In *Systems and Computer Science*, London, ON. University of Toronto Press.

Groenink, A. V. (1996). Mild context-sensitivity and tuple-based generalizations of Context-Free Grammar. Technical Report CS-R9634-1996, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands.

Groenink, A. V. (1997). *Surface without Structure – Word order and tractability in Natural Language analysis*. PhD thesis, Utrecht University, Utrecht, The Netherlands.

Grove, K. (2010). mcfgcyk. `http://conf.ling.cornell.edu/kgrove/mcfgcky/`, February 3, 2011.

Hajič, J., Böhmová, A., Hajičová, E., and Vidová-Hladká, B. (2000). The Prague Dependency Treebank: A three-level annotation scenario. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 103–127. Kluwer, Amsterdam.

Hajičová, E., Kirschner, Z., and Sgall, P. (1999). A manual for analytic layer annotation of the Prague Dependency Treebank (English translation). Technical report, ÚFAL MFF UK, Prague, Czech Republic.

Hall, J. and Nivre, J. (2008a). A dependency-driven parser for German dependency and constituency representations. In Kübler and Penn (2008), pages 47–54.

Hall, J. and Nivre, J. (2008b). Parsing discontinuous phrase structure with grammatical functions. In Nordström, B. and Ranta, A., editors, *Advances in Natural Language Processing*, volume 5221 of *Lecture Notes in Computer Science*, pages 169–180. Springer, Gothenburg, Sweden.

Han, C.-h., Yoon, J., Kim, N., and Palmer, M. (2000). A feature-based lexicalized Tree Adjoining Grammar for Korean. Technical Report IRCS-00-04, University of Pennsylvania, Philadelphia, PA.

Harkema, H. (2001). *Parsing Minimalist Languages*. PhD thesis, University of California at Los Angeles, Los Angeles, CA.

Havelka, J. (2007). Beyond projectivity: Multilingual evaluation of constraints and measures on non-projective structures. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 608–615, Prague, Czech Republic. Association for Computational Linguistics.

Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

Hockenmaier, J. (2003). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, School of Informatics, University of Edinburgh, Edinburgh, UK.

Hockenmaier, J. and Steedman, M. (2002a). Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas, Spain. European Language Resources Association (ELRA).

Hockenmaier, J. and Steedman, M. (2002b). Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 335–342, Philadelphia, PA. Association for Computational Linguistics.

Höhle, T. (1986). Der Begriff "Mittelfeld" – Anmerkungen über die Theorie der topologischen Felder. In *Akten des Siebten Internationalen Germanistenkongresses 1985*, Göttingen, Germany.

Holan, T., Kuboň, V., Oliva, K., and Plátek, M. (1998). Two useful measures of word order complexity. In *Proceedings of the Workshop on*

Processing of Dependency-Based Grammars at COLING-ACL'98, pages 21–29, Montréal, Canada. Association for Computational Linguistics.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Hotz, G. and Pitsch, G. (1994). Fast uniform analysis of Coupled Context-Free Languages. In *Proceedings of the 21th International Colloquium on Automata, Languages and Programming (ICALP'94)*, volume 820 of *Lecture Notes in Computer Science*, pages 412–423, Berlin, Heidelberg. Springer.

Huang, L. and Chiang, D. (2005). Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Vancouver, BC. Association for Computational Linguistics.

Hudson, R. (1984). *Word Grammar*. Basil Blackwell, Oxford.

Huybregts, R. (1984). The weak inadequacy of context-free phrase structure grammars. In de Haan, G., Trommelen, M., and Zonneveld, W., editors, *Van periferie naar kern*, pages 81–91. Foris, Dordrecht.

Jijkoun, V. (2003). Finding non-local dependencies: Beyond pattern matching. In *The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics*, pages 37–43, Sapporo, Japan. Association for Computational Linguistics.

Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84.

Johnson, M. (1985). Parsing with discontinuous constituents. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 127–132, Chicago, IL. Association for Computational Linguistics.

Johnson, M. (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Johnson, M. (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages

136–143, Philadelphia, PA. Association for Computational Linguistics.

Joshi, A. (1985). How much context-sensitivity is necessary for characterizing structural descriptions? In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural language processing: Theoretical, computational and psychological perspectives*, pages 206–250. Cambridge University Press, New York.

Joshi, A. and Sarkar, A. (2003). Tree Adjoining Grammars and their application to statistical parsing. In Bod et al. (2003), pages 253–281.

Joshi, A. K. (1987). An introduction to Tree Adjoining Grammars. In Manaster-Ramer, A., editor, *Mathematics of Language*. John Benjamins, Amsterdam.

Joshi, A. K., Levy, L. S., and Takahashi, M. (1975). Tree Adjunct Grammars. *Journal of Computer and System Science*, 10(1):136–163.

Joshi, A. K. and Schabes, Y. (1997). Tree-Adjoining Grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York.

Kaeshammer, M. and Demberg, V. (2012). German and English treebanks and lexica for Tree-Adjoining Grammars. In *Proceedings of the Eighth International Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA). To appear.

Kahane, S., Nasr, A., and Rambow, O. (1998). Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 646–652, Montreal, QC. Association for Computational Linguistics.

Kallmeyer, L. (1999). *Tree Description Grammars and Underspecified Representations*. PhD thesis, University of Tübingen, Tübingen, Germany. Technical Report IRCS-99-08 at the Institute for Research in Cognitive Science, Philadelphia.

Kallmeyer, L. (2009). A declarative characterization of different types of Multicomponent Tree Adjoining Grammars. *Research on Language and Computation*, 7(1):55–99.

Kallmeyer, L. (2010a). On mildly context-sensitive non-linear rewriting. *Research on Language and Computation*, 8(4):341–363.

Kallmeyer, L. (2010b). *Parsing beyond Context-Free Grammar*. Springer, Heidelberg.

Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., and Dellert, J. (2008a). Developing a TT-MCTAG for German with an RCG-based Parser. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., Dellert, J., and Evang, K. (2008b). TuLiPA: Towards a multi-formalism parsing environment for grammar engineering. In *Proceedings of the Workshop on Grammar Engineering Across Frameworks (GEAF) at COLING 2008*, pages 1–8, Manchester, UK.

Kallmeyer, L. and Maier, W. (2009). An incremental Earley parser for Simple Range Concatenation Grammar. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 61–64, Paris, France. Association for Computational Linguistics.

Kallmeyer, L. and Maier, W. (2010). Data-driven parsing with Probabilistic Linear Context-Free Rewriting Systems. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 537–545, Beijing, China.

Kallmeyer, L. and Maier, W. (2012). Data-driven parsing with Probabilistic Linear Context-Free Rewriting Systems. *Computational Linguistics*, 39(1). Accepted for publication.

Kallmeyer, L., Maier, W., and Parmentier, Y. (2009). An Earley parsing algorithm for Range Concatenation Grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 9–12, Singapore. Association for Computational Linguistics.

Kallmeyer, L., Maier, W., and Parmentier, Y. (2009). Un Algorithme d'Analyse de Type Earley pour Grammaires à Concaténation d'Intervalles. In *Conférence sur le Traitement Automatique des Langues Naturelles - TALN'09*, Senlis France. ATALA.

Kallmeyer, L., Maier, W., and Satta, G. (2009). Synchronous rewriting in treebanks. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 69–72, Paris, France. Association for Computational Linguistics.

Kallmeyer, L. and Parmentier, Y. (2008). On the relation between Multicomponent Tree Adjoining Grammars with Tree Tuples (TT-MCTAG) and Range Concatenation Grammars (RCG). In *Second International Conference on Language and Automata Theory and Applications (LATA 2008), Revised Papers*, Lecture Notes in Computer Science, pages 263–274. Springer, Tarragona, Spain.

Kallmeyer, L. and Satta, G. (2009). A polynomial-time parsing algorithm for TT-MCTAG. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 994–1002, Singapore. Association for Computational Linguistics.

Kanazawa, M. (2009a). A prefix-correct Earley recognizer for Multiple Context-Free Grammars. In *Proceedings of the Ninth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+9)*, pages 49–56.

Kanazawa, M. (2009b). The pumping lemma for well-nested Multiple Context-Free Languages. In Diekert, V. and Nowotka, D., editors, *Developments in Language Theory, 13th International Conference (DLT 2009)*, volume 5583 of *Lecture Notes in Computer Science*, pages 312–325. Springer, Stuttgart, Germany.

Kanazawa, M. and Salvati, S. (2010). The copying power of well-nested Context-Free Grammars. In Dediu, A.-H., Fernau, H., and Martín-Vide, C., editors, *Fourth International Conference on Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 344–355, Berlin, Heidelberg. Springer.

Kaplan, R., Riezler, S., King, T. H., Maxwell, J. T., Vasserman, A., , and Crouch, R. (2004). Speed and accuracy in shallow and deep stochas-

tic parsing. In *naacl-hlt-04*, pages 97–104, Boston, MA. Association for Computational Linguistics.

Kasami, T. (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

Kato, Y., Seki, H., and Kasami, T. (2006). Stochastic multiple Context-Free Grammar for RNA pseudoknot modeling. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+8)*, pages 57–64. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2001). Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, Beijing, China.

Klein, D. and Manning, C. D. (2003a). A* parsing: Fast exact viterbi parse selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 40–47, Edmonton, AB. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2003b). Accurate unlexicalized parsing. In *Proceedings of the 41th Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2003c). Fast exact inference with a factored model for Natural Language parsing. In *Advances in Neural Information Processing Systems 15 (NIPS)*, pages 3–10, Vancouver, BC. MIT Press.

Klein, P. N. (1998). Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th annual European Symposium on Algorithms (ESA)*, pages 91–102.

Knuth, D. E. (1977). A generalization of Dijkstra's algorithm. *Information Processing Letters*, 1(6):1–5.

Kracht, M. (2003). *The Mathematics of Language*. Mouton de Gruyter, Berlin.

Kromann, M. T. (2003). The Danish Dependency Treebank and the DTAG treebank tool. In *Second Workshop on Treebanks and Linguistic Theories (TLT-2003)*, Växjö, Sweden.

Kruijff, G.-J. M. (2002). Formal and computational aspects of dependency grammar: History and development of DG. Technical report, FoLLI, the Association for Logic, Language and Information. ESSLLI 2002 Course Notes.

Kübler, S. (2005). How do treebank annotation schemes influence parsing results? Or how not to compare apples and oranges. In *Recent Advances in Natural Language Processing 2005 (RANLP 2005)*, pages 293–300, Borovets, Bulgaria.

Kübler, S., Hinrichs, E. W., and Maier, W. (2006). Is it really that difficult to parse German? In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 111–119, Sydney, Australia. Association for Computational Linguistics.

Kübler, S., Maier, W., Rehbein, I., and Versley, Y. (2008). How to compare treebanks. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, pages 2322–2329, Marrakech, Morocco. European Language Resources Association (ELRA).

Kübler, S. and Penn, G., editors (2008). *Proceedings of the Workshop on Parsing German*. Association for Computational Linguistics, Columbus, Ohio.

Kudo, T. and Matsumoto, Y. (2004). A boosting algorithm for classification of semi-structured text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 301–308, Barcelona, Spain. Association for Computational Linguistics.

Kuhlmann, M. (2007). *Dependency Structures and Lexicalized Grammars*. Doctoral dissertation, Saarland University, Saarbrücken, Germany.

Kuhlmann, M. and Möhl, M. (2007a). Mildly context-sensitive dependency languages. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Prague, Czech Republic. Association for Computational Linguistics.

Kuhlmann, M. and Möhl, M. (2007b). The string-generative capacity of regular dependency languages. In *Proceedings of the 12th Conference on Formal Grammar (FG-2007)*, Dublin, Ireland.

Kuhlmann, M. and Nivre, J. (2006). Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 507–514, Sydney, Australia. Association for Computational Linguistics.

Kuhlmann, M. and Satta, G. (2009). Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 478–486, Athens, Greece. Association for Computational Linguistics.

Kunze, J. (1975). *Abhängigkeitsgrammatik*, volume 12 of *Studia grammatica*. Akademie-Verlag, Berlin.

Lambek, J. (1958). The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Langer, H. (1998). Experimente mit verallgemeinerten lookahead-algorithmen. In Schröder, B., Lenders, W., Hess, W., and Portele, T., editors, *Computer, Linguistik und Phonetik zwischen Sprache und Sprechen. Tagungsband der 4. Konferenz zur Verarbeitung natürlicher Sprache - KONVENS-98*, pages 69–82, Frankfurt am Main. Peter Lang Verlag.

Lee, L. (2002). Fast Context-Free Grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Levine, R. D. and Meurers, D. W. (2006). Head-Driven Phrase Structure Grammar: Linguistic approach, formal foundations, and computational realization. In Brown, K., editor, *Encyclopedia of Language and Linguistics*. Elsevier, Oxford, second edition.

Levy, R. (2005). *Probabilistic Models of Word Order and Syntactic Discontinuity*. PhD thesis, Stanford University.

Levy, R. and Manning, C. (2003). Is it harder to parse Chinese or the Chinese treebank? In *Proceedings of the 41th Annual Meeting of the Association for Computational Linguistics*, pages 439–446, Sapporo, Japan. Association for Computational Linguistics.

Levy, R. and Manning, C. (2004). Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 327–334, Barcelona, Spain. Association for Computational Linguistics.

Lichte, T. (2007). An MCTAG with tuples for coherent constructions in German. In *Proceedings of the 12th Conference on Formal Grammar (FG-2007)*, Dublin, Ireland.

Lichte, T. (2012). Analyse kohärenter Konstruktionen mit TAG-Varianten. Manuscript.

Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1420–1427, Montreal, QC.

Ljunglöf, P. (2004). *Expressivity and Complexity of the Grammatical Framework*. PhD thesis, Göteborg University, Gothenburg, Sweden.

Ljunglöf, P. (2004). Grammatical Framework and Multiple Context-Free Grammars. In *Proceedings of the 9th Conference on Formal Grammar (FG-2004)*, Nancy, France.

Lobin, H. (1993). *Koordinationssyntax als prozedurales Phänomen*, volume 46 of *Studien zur deutschen Grammatik*. Narr, Tübingen.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Maier, W. (2004). n2cf – Umwandeln des TIGER-Korpus in eine kontextfreie Struktur. Term paper, University of Tübingen.

Maier, W. (2006). Annotation schemes and their influence on parsing results. In *Proceedings of the Student Research Workshop at COLING/ACL 2006*, pages 19–24, Sydney, Australia. Association for Computational Linguistics.

Maier, W. (2010). Direct parsing of discontinuous constituents in German. In Seddah et al. (2010).

Maier, W. and Kallmeyer, L. (2010). Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+10)*, pages 119–126, New Haven, CT.

Maier, W. and Lichte, T. (2011). Characterizing discontinuity in constituent treebanks. In *Formal Grammar. 14th International Conference, FG 2009. Bordeaux, France, July 25-26, 2009. Revised Selected Papers*, volume 5591 of *Lecture Notes in Artificial Intelligence*, pages 167–182, Berlin, Heidelberg, New York. Springer-Verlag.

Maier, W. and Søgaard, A. (2008). Treebanks and mild context-sensitivity. In de Groote, P., editor, *Proceedings of the 13th Conference on Formal Grammar (FG-2008)*, pages 61–76, Hamburg, Germany. CSLI Publications.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge.

Marcus, M., Kim, G., Marcinkiewicz, M. A., Macintyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*, pages 114–119.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. Special Issue on Using Large Corpora: II.

Matsuzaki, T., Miyao, Y., and Tsujii, J. (2005). Probabilistic CFG with latent annotations. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics*, pages 75–82, Ann Arbor, MI. Association for Computational Linguistics.

McDonald, R. and Pereira, F. (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, Trento, Italy. Association for Computational Linguistics.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530, Vancouver, BC. Association for Computational Linguistics.

McDonald, R. and Satta, G. (2007). On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technology*, pages 121–132, Prague, Czech Republic. Association for Computational Linguistics.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. Suny Series in Linguistics. State University of New York Press.

Michaelis, J. (2001a). Derivational minimalism is mildly context-sensitive. In Moortgat, M., editor, *Selected papers of Logical Aspects of Computational Linguistics, Third International Conference (LACL'98)*, volume 2014 of *Lecture Notes in Computer Science*, pages 179–198, Grenoble, France. Springer.

Michaelis, J. (2001b). *On Formal Properties of Minimalist Grammars*. PhD thesis, Potsdam University, Potsdam, Germany.

Michaelis, J. (2001c). Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars. In de Groote, P., Morrill, G., and Retoré, C., editors, *Proceedings of Logical Aspects of Computational Linguistics, 4th International Conference (LACL'01)*, volume 2099 of *Lecture Notes in Computer Science*, pages 228–244. Springer, Le Croisic, France.

Michaelis, J. and Kracht, M. (1997). Semilinearity as a syntactic invariant. In Retoré, C., editor, *Selected Papers of Logical Aspects of Computational Linguistics, First International Conference (LACL'96)*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95, Nancy, France. Springer.

Miyao, Y. (2006). *From Linguistic Theory to Syntactic Analysis: Corpus-oriented Grammar Development and Feature Forest Model*. PhD thesis, University of Tokyo, Tokyo, Japan.

Miyao, Y. and Tsujii, J. (2008). Feature forest models for probabilistic HPSG parsing. *Computational Linguistics*, 34(1):35–80.

Nakanishi, R., Takada, K., and Seki, H. (1997). An efficient recognition algorithm for Multiple Context Free Languages. In *Fifth Meeting on Mathematics of Language*, pages 1–5, Schloss Dagstuhl, Germany. The Association for Mathematics of Language.

Nasr, A. and Rambow, O. (2004). A simple string-rewriting formalism for dependency grammar. In *Recent Advances in Dependency Grammar at COLING 04*, pages 17–24, Geneva, Switzerland.

Nederhof, M.-J. (1997). Solving the correct-prefix property for TAGs. In *Fifth Meeting on Mathematics of Language*, pages 124–130, Schloss Dagstuhl, Germany. The Association for Mathematics of Language.

Nederhof, M.-J. (1998). An alternative LR algorithm for TAGs. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 946–952, Montreal, QC. Association for Computational Linguistics.

Nederhof, M.-J. (1999). The computational complexity of the correct-prefix property for TAGs. *Computational Linguistics*, 25(3):345–360.

Nederhof, M.-J. (2003). Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):1–9.

Nederhof, M.-J. and Satta, G. (2006). Estimation of consistent Probabilistic Context-Free Grammars. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 343–350, New York, NY. Association for Computational Linguistics.

Neuhaus, P. and Bröker, N. (1997). The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 337–343, Madrid, Spain. Association for Computational Linguistics.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies*, pages 149–160, Nancy, France.

Nivre, J. (2006). Constraints on non-projective dependcy parsing. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–80, Trento, Italy. Association for Computational Linguistics.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics*, pages 99–106, Ann Arbor, MI. Association for Computational Linguistics.

Osenova, P. and Simov, K. (2004). BTB-TR05: BulTreebank Stylebook. Technical Report 05, BulTreeBank Project, Sofia, Bulgaria.

Parikh, R. (1966). On Context-Free Languages. *Journal of the Association for Computing Machinery*, 13:570–681.

Parmentier, Y. and Maier, W. (2008). Using constraints over finite sets of integers for Range Concatenation Grammar parsing. In Nordström, B. and Ranta, A., editors, *Advances in Natural Language Processing*, volume 5221 of *Lecture Notes in Computer Science*, pages 360–365, Gothenburg, Sweden. Springer.

Pauls, A. and Klein, D. (2009). K-best A* parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 958–966, Singapore. Association for Computational Linguistics.

Pereira, F. C. N. and Warren, D. H. D. (1983). Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge, MA. Association for Computational Linguistics.

Petrov, S. (2009). *Coarse-to-Fine Natural Language Processing*. PhD thesis, University of California at Berkeley, Berkeley, CA.

Petrov, S. (2010). Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Los Angeles, CA. Association for Computational Linguistics.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, NY. Association for Computational Linguistics.

Plaehn, O. (1999). Probabilistic parsing with Discontinuous Phrase Structure Grammar. Master's thesis, Saarland University, Saarbrücken, Germany.

Plaehn, O. (2004). Computing the most probable parse for a Discontinuous Phrase-Structure Grammar. In Bunt, H., Carroll, J., and Satta, G., editors, *New developments in parsing technology*, volume 23 of *Text, Speech And Language Technology*, pages 91–106. Kluwer.

Pollard, C. (1984). *Generalized Phrase Structure Grammars, Head Grammars and Natural Language*. Doctoral dissertation, Center for the Study of Language and Information, Stanford University, Stanford, CA.

Pollard, C. and Sag, I. (1994). *Head-Driven Phrase Structure Grammar*. CSLI Publications, Chicago.

Prescher, D. (2005). Inducing head-driven PCFGs with latent heads: Refining a tree-bank grammar for parsing. In Gama, J., editor, *Proceedings of the 16th European Conference on Machine Learning (ECML*

2005), volume 3720 of *Lecture Notes in Artificial Intelligence*, pages 292–304, Berlin, Heidelberg. Springer.

Prolo, C. A. (2000). An efficient LR parser generator for Tree Adjoining Grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, pages 207–218.

Prolo, C. A. (2003). *LR Parsing for Tree Adjoining Grammars and its Application to Corpus-based Natural Language Parsing*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

Pullum, G. K. (2007). The evolution of model-theoretic frameworks in linguistics. In *Model-Theoretic Syntax @ 10 at ESSLLI 2007*, pages 1–10, Dublin, Ireland.

Pullum, G. K. and Scholz, B. C. (2001). On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In de Groote, P., Morrill, G., and Retoré, C., editors, *Proceedings of Logical Aspects of Computational Linguistics, 4th International Conference (LACL'01)*, volume 2099 of *LNAI*, pages 17–43, Le Croisic, France. Springer.

Radzinski, D. (1991). Chinese number-names, tree adjoining languages, and mild context-sensitivity. *Computational Linguistics*, 17(3):277–299.

Rafferty, A. and Manning, C. D. (2008). Parsing three German treebanks: Lexicalized and unlexicalized baselines. In Kübler and Penn (2008), pages 40–46.

Rambow, O. (1994). *Formal and computational aspects of Natural Language syntax*. PhD thesis, University of Pennsylvania, Philadelphia, PA.

Rambow, O. and Satta, G. (1996). Synchronous models of language. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 116–123, Santa Cruz, CA. Association for Computational Linguistics.

Rambow, O. and Satta, G. (1999). Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223(1-2):87–120.

Ranta, A. (2004). Grammatical Framework, a typetheoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

Reape, M. (1991). Parsing bounded discontinuous constituency. In *Computational Linguistics in the Netherlands. Papers from the First CLIN-meeting*, Utrecht, The Netherlands.

Reape, M. (1994). Domain union and word order variation in German. In Nerbonne, J., Netter, K., and Pollard, C. J., editors, *German in Head-Driven Phrase Structure Grammar*, volume 46 of *CSLI Lecture Notes*, pages 151–198. CSLI Publications, Stanford, CA.

Rehbein, I. and van Genabith, J. (2007a). Evaluating evaluation measures. In *Proceedings of the 16th Nordic Conference of Computational Linguistics NODALIDA-2007*, pages 372–379, Tartu, Estonia.

Rehbein, I. and van Genabith, J. (2007b). Treebank annotation schemes and parser evaluation for German. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 630–639, Prague, Czech Republic. Association for Computational Linguistics.

Rehbein, I. and van Genabith, J. (2007c). Why is it so difficult to compare treebanks? TIGER and TüBa-D/Z revisited. In *Sixth International Workshop on Treebanks and Linguistic Theories*, pages 115–126, Bergen, Norway. Northern European Association for Language Technology (NEALT).

Resnik, P. (1992). Probabilistic Tree-Adjoining Grammars as a framework for natural language processing. In *Proceedings of COLING 1992: The 14th International Conference on Computational Linguistics*, pages 418–424, Nantes, France.

Richter, F. and Sailer, M. (1995). Remarks on linearization. Reflections on the treatment of LP-rules in HPSG in a Typed Feature Logic. Magisterarbeit, University of Tübingen, Tübingen, Germany.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell, J. T., and Johnson, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In

Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 271–278, Philadephia, PA. Association for Computational Linguistics.

Roark, B. (2001). *Robust Probabilistic Predictive Syntactic Processing*. PhD thesis, Brown University, Providence, RI.

Rosenkrantz, D. J. and Lewis, P. M. (1970). Deterministic left-corner parsing. In *Proceedings of the 11th Annual Symposium on Switching and Automata Theory (SWAT 1970)*, pages 139–152, Washington, DC. IEEE Computer Society.

Sag, I. A. and Wasow, T. (1999). *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, CA.

Sagot, B. (2005). Linguistic facts as predicates over ranges of the sentence. In Blache, P., Stabler, E., Busquets, J., and Moot, R., editors, *Proceedings of Logical Aspects of Computational Linguistics, 5th International Conference (LACL 2005)*, volume 3492 of *Lecture Notes in Computer Science*, pages 271–286, Bordeaux, France. Springer.

Sagot, B. and Satta, G. (2010). Optimal rank reduction for Linear Context-Free Rewriting Systems with fan-out two. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 525–533, Uppsala, Sweden. Association for Computational Linguistics.

Sampson, G. and Babarczy, A. (2003). A test of the leaf-ancestor metric for parse accuracy. *Journal of Natural Language Engineering*, 9:365–380.

Sangati, F. and Zuidema, W. (2009). Unsupervised methods for head assignments. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 701–709, Athens, Greece. Association for Computational Linguistics.

Sarkar, A. (1998). Conditions on consistency of probabilistic Tree Adjoining Grammar. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 1164–1170, Nantes, France. Association for Computational Linguistics.

Sarkar, A. and Joshi, A. (1996). Coordination in Tree Adjoining Grammars: Formalization and implementation. In *Proceedings of COLING 1996: The 16th International Conference on Computational Linguistics*, pages 610–615, Copenhagen, Denmark.

Satta, G. (1992). Recognition of Linear Context-Free Rewriting Systems. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 89–95, Newark, DE. Association for Computational Linguistics.

Satta, G. (1994). Tree-Adjoining Grammar parsing and Boolean matrix multiplication. *Computational Linguistics*, 20(2):173–192.

Scha, R. (1990). Taaltheorie en taaltechnologie; competence en performance. In de Kort, R. and Leerdam, G. L. J., editors, *Computertoepassingen in de Neerlandistiek*, pages 7–22. LVVN, Almere.

Schabes, Y. (1992). Stochastic lexicalized Tree-Adjoining Grammar. In *Proceedings of COLING 1992: The 14th International Conference on Computational Linguistics*, pages 425–432, Nantes, France.

Schabes, Y. and Joshi, A. K. (1988). An Earley-type parsing algorithm for Tree Adjoining Grammars. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 258–269, Buffalo, NY. Association for Computational Linguistics.

Schabes, Y. and Waters, R. C. (1995). Tree Insertion Grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–514.

Schluter, N. and van Genabith, J. (2008). Treebank-based acquisition of LFG parsing resources for French. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, pages 2909–2916, Marrakech, Morocco. European Language Resources Association (ELRA).

Schöning, U. (2001). *Theoretische Informatik – kurzgefasst*. Spektrum-Hochschultaschenbuch. Spektrum, Akademischer Verlag, Heidelberg, Berlin, 4th edition.

Schuler, W., AbdelRahman, S., Miller, T., and Schwartz, L. (2010). Broad-coverage parsing using human-like memory constraints. *Computational Linguistics*, 36(1):1–30.

Schulte, C. (2002). *Programming Constraint Services*, volume 2302 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Seddah, D., Kübler, S., and Tsarfaty, R., editors (2010). *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*. Association for Computational Linguistics, Los Angeles, CA.

Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On Multiple Context-Free Grammars. *Theoretical Computer Science*, 88(2):191–229.

Sekine, S. and Collins, M. J. (1997). EVALB bracket scoring program. `http://nlp.cs.nyu.edu/evalb/`, January 3, 2012.

Shen, L. and Joshi, A. K. (2005). Incremental LTAG parsing. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 811–818, Vancouver, BC. Association for Computational Linguistics.

Shieber, S. M. (1984). Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, 7(2):135–154.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1&2):3–36.

Sikkel, K. (1997). *Parsing Schemata*. Texts in Theoretical Computer Science. Springer, Berlin, Heidelberg, New York.

Sima'an, K. (1996). Computational complexity of probabilistic disambiguation by means of tree grammars. In *Proceedings of COLING 1996: The 16th International Conference on Computational Linguistics*, volume 2, pages 1175–1180, Copenhagen, Denmark.

Sima'an, K. (1999). *Learning efficient disambiguation*. Illc dissertation series 1999-02, Utrecht Institute of Linguistics OTS, Utrecht, The Netherlands.

Skut, W., Krenn, B., Brants, T., and Uszkoreit, H. (1997). An annotation scheme for free word order languages. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 88–95, Washington, DC.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 277–291, Tilburg, The Netherlands.

Søgaard, A. (2011). A $O(|G|n^6)$ time extension of inversion transduction grammars. *Machine Translation*. To appear.

Søgaard, A., Lichte, T., and Maier, W. (2007). The complexity of linguistically motivated extensions of Tree-Adjoining Grammar. In *Recent Advances in Natural Language Processing 2007 (RANLP 2007)*, pages 548–553, Borovets, Bulgaria.

Stabler, E. P. (1997). Derivational minimalism. In Retoré, C., editor, *Selected Papers of Logical Aspects of Computational Linguistics, First International Conference (LACL'96)*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95, Nancy, France. Springer.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge.

Suhre, O. (2000). Computational aspects of a grammar formalism for languages with freer word order. Diplomarbeit, University of Tübingen, Tübingen, Germany. Arbeitspapier des SFB 340, Nr. 154.

TEI Consortium, editor (2010). *TEI P5: Guidelines for Electronic Text Encoding and Interchange. Version 1.8.0, Last updated on November 5th 2010*, chapter 18 Feature Structures. TEI Consortium. `http://www.tei-c.org/release/doc/tei-p5-doc/en/html/FS.html`, November 15, 2010.

Telljohann, H., Hinrichs, E., Kübler, S., and Zinsmeister, H. (2006). Stylebook for the Tübingen Treebank of Written German (TüBa-D/Z). Technischer Bericht, Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany. Revidierte Fassung.

Tesnière, L. (1959). *Eléments de syntaxe structurale*. Klincksieck, Paris.

Tounsi, L., Attia, M., and van Genabith, J. (2009). Parsing Arabic using treebank-based LFG resources. In *LFG09, Proceedings of the 14th International LFG Conference*, CSLI Publications, pages 583–586, Cambridge, UK.

Tsarfaty, R. and Sima'an, K. (2007). Three-dimensional parametrization for parsing morphologically rich languages. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 156–167, Prague, Czech Republic. Association for Computational Linguistics.

Ule, T. (2003). Directed treebank refinement for PCFG parsing. In *Second Workshop on Treebanks and Linguistic Theories (TLT-2003)*, Växjö, Sweden.

Ule, T. (2006). *Treebank Refinement*. Dissertation, University of Tübingen, Tübingen, Germany.

Uszkoreit, H. (1986). Linear precedence in discontinuous constituents: Complex fronting in german. CSLI report CSLI-86-47, Center for the Study of Language and Information, Stanford University, Stanford, CA.

Valiant, L. G. (1975). General context-free recognition in less than cubic time. *Journal of Computer and System Science*, 10:191–229.

van Cranenburgh, A., Scha, R., and Sangati, F. (2011). Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2011)*, Dublin, Ireland.

Versley, Y. (2005). Parser evaluation across text types. In *Fourth Workshop on Treebanks and Linguistic Theories (TLT 2005)*, Barcelona, Spain.

Vijay-Shanker, K. and Joshi, A. K. (1985). Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, Chicago, IL. Association for Computational Linguistics.

Vijay-Shanker, K. and Joshi, A. K. (1988). Feature structure based Tree Adjoining Grammar. In *Proceedings of COLING 1988: The 12th International Conference on Computational Linguistics*, pages 714–719, Budapest, Hungary.

Vijay-Shanker, K., Weir, D., and Joshi, A. K. (1987). Characterising structural descriptions used by various formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA. Association for Computational Linguistics.

Vijayshanker, K. (1987). *A study of tree-adjoining grammars*. PhD thesis, University of Pennsylvania, Philadelphia, PA.

Villemonte de la Clergerie, E. (2002). Parsing Mildly Context-Sensitive Languages with Thread Automata. In *Proceedings of COLING 2002: The 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Vogel, C. and Bunt, H. (1992). Representing discontinuous constituency: Motivations and complexity. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, Seoul, South Korea.

Vogel, C. and Erjavec, T. (1994). Restricted Discontinuous Phrase Structure Grammar and its ramifications. In Martin-Vide, C., editor, *Current Issues in Mathematical Linguistics*, pages 131–140. Elsevier, Amsterdam.

Weir, D. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylviania, Philadelphia, PA.

Wu, D. (1997). Stochastic Inversion Transduction Grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404.

Wurmbrand, S. (2001). *Infinitives. Restructuring and Clause Structure*, volume 55 of *Studies in Generative Grammar*. de Gruyter, Berlin, New York.

Xia, F. (2001). *Investigating the Relationship between Grammars and Treebanks for natural languages*. Doctoral dissertation, University of Pennsylvania, Philadelphia, PA.

XTAG Research Group (2001). A Lexicalized Tree Adjoining Grammar for English. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA.

Yeh, A. S. (2000). More accurate tests for the statistical significance of result differences. In *Proceedings of COLING 2000: The 18th International Conference on Computational Linguistics*, pages 947–953, Saarbrücken, Germany.

Yli-Jyrä, A. (2003). Multiplanarity – a model for dependency structures in treebanks. In *Second Workshop on Treebanks and Linguistic Theories (TLT-2003)*, Växjö, Sweden.

Younger, D. H. (1967). Recognition and parsing of Context-Free Languages in time n^3. *Information and Control*, 10(2):189–208.

Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262.

Zwicky, A. (1986). Concatenation and liberation. In *Proceedings of the Twenty-Second Regional Meeting (General Session) of the Chicago Linguistics Society*, pages 65–74.

INDEX