UNIVERSITY OF OSLO
Department of Informatics

Context-free approximation of large unification grammars

A walk in the forest

Master's Thesis

Johan Benum Evensberget

December 12, 2011

I want to thank my advisor, Stephan Oepen, who has been a pleasure to work with from day one, providing helpful comments on writing, navigation in the big web of science, stimulating, inspirational and encouraging discussions and help for seeing trees for just forests.

I want to thank Emanuele Lapponi, Brendan Lee, Arne Skjærholt, André Lynum and Gisle Ytrestøl for fruitful discussions of both technical and theoretical nature, and help with proof-reading.

Thanks.

Contents

List of Tables **ii**

List of Figures **iii**

1 Introduction **1**

1.1 Overview 2

2 Background **5**

2.1 Syntactic Analysis 5

2.2 Graphs 7

2.3 Formal language theory 8

2.4 Context free grammars 8

2.5 Unification based grammars 10

2.6 HPSG 14

3 Context-free approximation **17**

3.1 Motivation 17

3.2 Approximation of formal grammars 19

3.3 Context free approximation of unification based grammars 21

3.4 Refining reconstructable approximations 23

3.5 Related work 26

4 A framework for data-driven approximation **29**

4.1 Practical considerations 29

4.2 Revisiting sound versus unsound approximation 30

4.3 Resources 32

4.4 Implementation and extraction procedure 33

4.5 Feature selection 38

4.6 Measures on Grammars 40

4.7 Extraction experiments 42

5 Standalone parsing with approximations **47**
5.1 Parsing and evaluation tools 48
5.2 Experiment setup 49
5.3 Parsing with BitPar 50
5.4 Investigating binary parallelization 52
5.5 A meta-comparison of parsers 53
5.6 Outlook and Reflections 55

6 Conclusion **59**
6.1 Future work 60

A Feature paths for internal annotation **63**

B The WikiWoods subset **65**

Bibliography **67**

List of Tables

3.1 Comparison of CFG and UBG 21

4.1 Static grammar measures with varying degree of annotation 42
4.2 Simulated grammar coverage. 43
4.3 Lexical collapsing. 44
4.4 Lexical generalization. 45
4.5 WikiWoods. 45

5.1 WS13 as parsed by PET 50
5.2 WS13 as parsed by BitPar with different annotations 51
5.3 WS13 as parsed by BitPar with large grammars 52
5.4 WS13 as parsed by BitPar with different annotations 53
5.5 WS13 as parsed by BitPar with different annotations 55

A.1 Listing of features used for internal annotation 63

List of Figures

2.1 Graphs and trees 7
2.2 Context free derivations 9
2.3 PCFG derivations 10
2.4 Feature structure in AVM and graph notation 11
2.5 Type hierarchy 13
2.6 Examples of unification 13
2.7 Head-feature principle 15
2.8 Example HPSG rule schemata 15
2.9 Example HPSG-derivation 16

3.1 Venn diagram of string language approximation 20
3.2 HPSG parse tree and corresponding rules 23
3.3 Feature annotation 24
3.4 Original and lexically collapsed parse trees for “She plays Eliante.” 25

4.1 Derivational morphology rules 31
4.2 Annotating feature structures 34
4.3 Lexical collapsing 35
4.4 Extracting CFG-rules 36
4.5 ERG Derivation for “The actress plays Eliante” 38
4.6 “The actress plays Eliante” with annotated heads 39
4.7 Grammar measures 41

5.1 Augmented-CKY 54
5.2 Unary rule expansion 55

CHAPTER 1

Introduction

Ikke tusend ord
sig prenter, som én gernings spor.

HENRIK IBSEN

In the last fifty years, more and more sophisticated methods of symbolic modeling of human language have been proposed and implemented on computers. While methods and ideals have varied, the common goal has always been to give an adequate model of human language, such that a computer can "understand", in a strong or a weak sense, and make use of ordinary natural language, and also "generate" natural language responses.

Context-free grammars, as proposed by Chomsky in the 1950s (1956), are a comparatively simple such model, but large-scale implementation of high quality context-free grammars proved to be unfeasible. In the decades to follow, new and more advanced computational models were proposed, dealing with the shortcomings of their predecessors. However, as the expressiveness and so-called linguistic adequacy of the models increased, so did the computational complexity and the effort needed to implement efficient algorithms for both the analysis and the development of the models, and of course the computational aspects of the model itself.

Head-driven phrase-structure grammar, HPSG, is one of the more advanced frameworks of such symbolic modeling. While very attractive from a modeling point of view, the computational effort needed to work with the model is significant. One of the fastest implementations of HPSG parsing will typically (on today's hardware) use several seconds on the analysis of a sentence. As is common with most of models of language, longer sentences take considerably longer time to process.

In the case of HPSG parsing; very long sentences with a high level of ambiguity might take a prohibitive amount of time to process, and in most practical approaches both time and memory limits have to be stipulated. At present, as much as 10% of sentences in typical corpora of text cannot be analyzed due to these limits, even with the comparatively generous limit on parsing time of 60 seconds. On these sentences, the parser is in fact *slower* than a human reader, which can be a critical point in many scenarios. For example, in interactive dialouge situations the computer must analyze the user's input and generate a proper response in a very short time, preferably at least as fast as another human would do.

In the last decade, context-free *approximations* have been studied as a way to speed up processing. In this scheme one could get the best of two worlds, the linguistic adequacy and expressiveness of the formalism from HPSG, and the ease of implementation and efficiency of context-free grammars. However, there are both theoretical and practical hurdles that have proved to make the approximation process difficult, and no scheme of approximation has seen widespread use so far.

On the theoretical side, context-free grammars prove to be a formalism that cannot express all the properties of HPSG. These two formalisms generate "formal languages" of different complexity. Therefore, it is impossible, in both the general and most practical cases, to create a context-free grammar that is equivalent to a HPSG. However, by "cherry-picking" the most interesting properties of the original HPSG grammar, a context-free approximation modeling the most important properties of an HPSG can be extracted.

On the practical side, the right properties have to be identified. This requires considerable experimental effort. Even though good information sources are available, obtaining good approximations still require that the correct properties are selected. Then one need to find a delicate balance between coverage, processing speed and accuracy, where each of these parameters could weigh differently depending on use scenarios.

Early efforts have focused chiefly on so-called "sound" approximations, meaning that the approximation is a true superlanguage of the original grammar. However, these grammars can often become so large that it is hard to put them into practical use. Furthermore true sound approximations can in practice be difficult to extract, especially with modern HPSG grammars which can license infinitely sized lexica.

1.1 Overview

In this work, we focus on "unsound" approximations, which need not be a true superlanguage of the original grammar. These grammars are typically much smaller than sound versions, and when extracted from treebanked material one could also augment these with a notion of probability, yielding a probabilistic context-free grammar. These PCFGs can also serve as a stochastic model of the underlying approximation, facilitating, among other things, parse selection. A special class of approximations are the so-called "reconstructable" approximations, which have the elegant property that an entire derivation in such a grammar specifies exactly one derivation in the original grammar if that derivation is well-formed with respect to the constraints in the original grammar. This means that all the information that was lost to facilitate a context-free approximation could be reconstructed at a later point if so desired.

We implement a flexible system for extracting approximations and reporting on various properties of the obtained grammars. In addition we also put these grammars to work as a stand-alone parsing system, giving a speed up of two orders of magnitude with only a modest loss in accuracy.

In chapter 2 we give a short review of the key points of the various theoretical models and frameworks we build upon in this work, starting with syntactic analysis in general, before moving on to a quick review of graphs and formal languages. Context-free grammars are discussed both as a formal mathematical object, but also how they can model linguistic structure, especially syntax.

We then give an account of unification-based grammars, and how they provide a more suitable formalism for fine-grained linguistic description of language, followed by a short review of typed feature-structures which is the underpinning formalism of HPSG, which is presented in some detail in the last section.

Chapter 3 discusses in detail the motivation behind context-free approximations, and reviews several scenarios where such approximations might be useful. Then an account of purely theoretical properties on formal-language approximation is given, followed by a discussion of the main task at hand, context-free approximation of unification-based grammars, detailing both the feasibility of the process and discussing important techniques to facilitate high quality results. The chapter rounds off with a review of related work.

In chapter 4 we move onwards from the theoretical side of the question and detail practical considerations of the approximation, with special focus on sound and unsound approaches. Next, the resources this thesis builds upon are presented, before the implementation and extraction procedure are detailed. Considerations pertaining to feature selection is laid out, including a discussion of one of the more important sources of features.

Next, we discuss both static and dynamic measures on the resulting approximations, which might serve as indicators of how an approximation can perform. Then we report results from extracting approximations with different configurations; the resulting grammars range from some few thousand production rules for the smallest, to nearly half a million for the largest grammars. We also resolve a practical "impedance mismatch" in the tokenization assumptions by the various formalisms with a process we call *lexical generalization*.

Chapter 5 is a case study where the grammars obtained in the previous chapter are put to the task of stand-alone parsing. We begin with detailing various evaluation criteria commonly used in parsing, before comparing both run-time performance and accuracy of the approximations, versus a high-performance UBG parser. These results show that one approximation strategy *lexical collapsing* can be applied to give a substantial increase in both performance and accuracy, which we also motivate theoretically. Our "best" configuration has an average parse time of 142 ms per sentence, while the original high-performance UBG parser uses 3.6 seconds in average. However, some accuracy is lost, at 37% versus 46% measures with an *exact match* metric. While this may sound grim, the error reduction ratio in moving from an accuracy of 46% over 37% is only 1.2.

A high-performance renowned, off-the-shelf, CFG parser is employed, but we also implement our own parser, showing that some of the assumptions made in the aforementioned parser might not be beneficial in all cases. The detailed algorithm is laid out. Our parser performs roughly four times better in one configuration, and can in addition give n-best outputs. We then present a "meta-comparison" of the performance of several recent and differing approaches to the same task. We continue with an outlook of future work, and discuss alternate evaluation schemes that may be more reflecting of the true usefulness of an approximation in this use scenario.

CHAPTER 2

Background

It's difficult to be rigorous about whether a machine really 'knows', 'thinks', etc., because we're hard put to define these things. We understand human mental processes only slightly better than a fish understands swimming.

The Little Thoughts of Thinking Machines
JOHN MCCARTHY (1927-2011)

This chapter aims to introduce a selection of relevant theoretical and practical aspects that we will build upon in this thesis. To set the scene of what this work is all about, we will give a mountain-top overview on syntactic analysis in general, and proceed to briefly discuss the most important points of formal language theory, graphs and trees, context free grammars and unification grammars. In this chapter, we aim to establish terminology, notation and conventions, and not to give a thorough introduction or formal definition.

2.1 Syntactic Analysis

Syntactic Analysis, often called parsing, is one of the major tasks in natural language processing. The overall aim is to recover the syntactical relationships between words and phrases in sentences, such as identifying the main verb phrase or figuring out what the subject of the sentence is. Recovering these relations is important for many tasks, such as information extraction, machine translation and text simplification, and brings us arguably one step closer to understanding the sentence's meaning.

At center stage in syntactic analysis lies a notion of a formal grammar[1], which is a collection of rules that describe how words and phrases can or cannot relate to each other, and what those relations actually are. The term "parsing" are both used for the analysis of *natural languages*, like French and English, and artificial languages,like programming languages, which are usually formally specified and unambiguous.

[1]Some parsers, especially dependence parsers, solely employ a statistic "oracle" to do the parsing. Here a formal grammar is not explicit in the traditional sense as discrete rules, but is indirect in the form of a statistical model of the data that was used to estimate the parameters of that model.

Natural language seems to resist any formalization, and it is hard, if not impossible, to not have an “impedance mismatch” between human language and computational models for it. It is common to use the term *coverage* to describe how much of a language a formalism can adequately describe. Typically there will always be some holes in coverage, arising from sources such as unknown words in the input, unmodeled syntactical constructions, or even because the parser is unable to process a sentence in a timely fashion. Holes in coverage are usually called *undergeneration*.

On the other hand, *overgeneration* can present itself as a problem just as tricky as undergeneration. A parser can assign incorrect analyses or accept ill-formed input, allowing syntactical constructions between phrases that a human would deem incorrect, such as the wrong agreement between verbs and subjects or allowing sentences with wrong word order. *Spurious ambiguity*, on the other hand, is when a sentence is given several different syntactic analyses which all have an equal semantic interpretation.

Syntactic analysis is difficult. First we need to find a formal grammar that can represent the syntactical relations we are interested in. One way of getting a grammar is to write a set of rules in an accompanying formalism directly. This approach to parsing is often called *rule-based* or *grammar-based*. The needed effort can be quite high, especially if broad coverage and low overgeneration are both wanted. Grammar rules can become quite dense and they can have a complex interplay between them; adding one rule to address undergeneration for one phenomenon can result in overgeneration in another part of the grammar.

Another way is to look at a collection of inputs and desired outputs, or in other words a *treebank*. This approach is usually called *treebank-based* or data-driven. This requires a sizable treebank, which should be annotated with the syntactical relations we want. We can then observe the grammar indirectly by looking at all the trees, and use that data to infer what the grammar should look like. We can easily construct a minimal grammar so that all the trees will be covered, but more advanced methods are often necessary to get better performing grammars.

In the field of parsing, one often distinguishes between “deep” and “shallow” approach to syntactic analysis. Shallow parsers aim to recover a coarse set of syntactical relations, such as subjects, objects and modification, while deep grammars aim to give a more detailed account on syntactic relations and processes like noun subcategorization, locative inversion, passive constructions and so on. Furthermore, and arguably the most interesting for “downstream” users, several deep grammar formalisms can present a semantic representation of the input and rules governing semantic composition, and especially what happens to it in the more complex constructions, can be made part of the formalism itself.

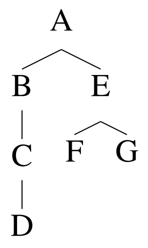
When a formalism has been agreed upon and correct parsers have been implemented, a difficult part of the problem can be attacked; selecting the *correct* analysis for a given sentence. Natural language is inherently highly ambiguous and a sentence can have possible analyses numbering in the thousands. A parser ideally aims to select the correct analysis among those, or at the very least give a smaller number of likely correct candidates. This process is called *disambiguation* or *parse selection*. Using a probabilistic model is a popular way to achieve this, where the statistically most probable analysis is selected. Sometimes this procedure can involve multiple stages and external models, where the parser first constructs a representation of all possible analyses and ranks them with its internal model, and then the most promising analyses are then inspected by another external system. This setup is called *reranking*. This can be useful as more advanced, but more expensive, stochastical models can be used.

Phrasal attachment, especially free modifiers such as prepositional phrases, are one of the biggest sources of ambiguity. For instance, in the sentence "He saw a man with a mirror", could have a short PP attachment; the man is carrying a mirror, or a long PP attachment; the mirror is being used to look with. This is normally called *structural ambiguity*. Word-level or *lexical* ambiguity, where a word in the input could have several word-classes is also a major factor. For example, "Man" can in principle be both a noun and a verb. Often these ambiguities will only be local, and the rest of the sentence imposes syntactical constraints that eliminate all the other readings, but it is not uncommon that complete analyses can be derived with both readings of an individual lexical item. We allow ourselves to use the classical example; "Time flies like an arrow, but fruit flies like a banana".

2.2 Graphs

Graphs are a very common and useful tool to represent structured relationships. A graph consists of *vertices* or nodes, and *edges*. An edge connects two vertices together. Loops are allowed, in other words, an edge could go from a node and into the very same node. Edges may be directed, that is, the connection is asymmetric. It is common to *label* both edges and vertices. The *degree* of a vertex is the number of edges connected to it. When edges are directed the degree is split into indegree and outdegree, representing incoming edges and outgoing edges respectively. A *path* in a graph describes a list of edges, or vertices[2], one could traverse to get from one vertex to another.

Trees are a subset of graphs and have directed edges and some restrictions on how the edges can connect nodes: (a) One node, the *root*, has an indegree of 0, in other words, no edges point to it. (b) Every node must be *connected* or reachable from the root through any number of other nodes. (c) All nodes, except the root, must have an indegree of 1. The node on the other end of the incoming edge is usually named the parent node, and all the nodes of the parent's outgoing edges are named children or daughters. Nodes without any daughters are *leaves*. See figure 2.1 for an example of graphs.



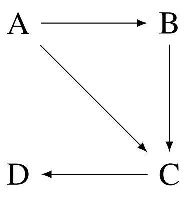


Figure 2.1: Two graphs. Only the left one is a tree. The tree nodes D, F and G are leaves, and A is the root.

[2]A path of vertices would only be unambiguous in a simple graph, a graph with no more than one outgoing edge from each node.

2.3 Formal language theory

A *formal language* is a mathematical object commonly used in the study of computation. We shall use $\mathscr{L}$ to refer to formal languages. A formal language is a set of "words"[3] which are strings made up from an alphabet of symbols. One of the ways to describe a formal language is with a *formal grammar*, which will typically be referred to with G in this work. A formal grammar is a set of rules and restrictions which describe the possible *derivations* or rule applications of a grammar, and make up a procedure of deciding whether some string s is in the language it describes, or in other words, if $s \in \mathscr{L}(G)$. We say that G generates S if that is the case. We shall largely follow the usual style of Hopcroft and Ullman (1979) and write grammars as $G = \langle N, T, P, S \rangle$. T is a set of terminal symbols. N is a set of non-terminal symbols, and S is a special start-symbol which all derivations must start from. It is required that $N \cap T = \varnothing$. P is a set of rules, which describe the derivations proper. Rules in P rewrite non-terminals to strings of terminals and other non-terminals. The traditional idea is that the notion of size of formal grammars should be finite, but interesting formal grammars should generate infinite languages[4]. Therefore P and the symbols N and T must all be finite sets. The language $\mathscr{L}(G)$ is the set of strings of terminals in T we can obtain by repeatedly deriving from S, which denotes the starting point for all derivations. Note that $S \in N$. We shall use uppercase letters for non-terminals, lower-case letters for terminals and Greek letters for any string of terminals and non-terminals, including the empty string ϵ.

Formal grammars can be partitioned into *classes* by placing restrictions on the form of rules. For example, a left-linear grammar can only have rules of the form $A \rightarrow a$, $A \rightarrow a B$ and $A \rightarrow \epsilon$. It cannot generate the same class of languages as context-sensitive grammars, which have rules of the form $\alpha A \gamma \rightarrow \alpha \beta \gamma$. In other words, we cannot "remold" context-sensitive grammars as left-linear grammars and have them generate exactly the same language. Grammars can be weakly or strongly equivalent. Two grammars G and F are weakly equivalent if $\mathscr{L}(G) = \mathscr{L}(F)$. G and F are strongly equivalent if they also generate the same derivation trees for all words in the language. Left-linear grammars and right-linear grammars ($A \rightarrow B a$) are strongly equivalent, in the sense that they both generate the same class of languages and derivation trees. The restrictions on the form of rules define the expressivity of the grammar, or in informal terms, how complex the languages those grammars describe are. The classic example of these classes is the *Chomsky Hierarchy* (Chomsky, 1956). Usually more restricted grammars can be implemented more easily and efficiently.

2.4 Context free grammars

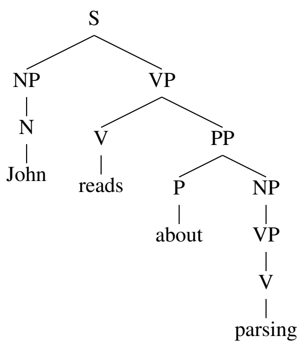
Context free grammars are ubiquitous in syntactic analysis, and present several attractive features; (a) they let us specify recursive structures which fits nicely with intuitive views on how syntax works, (b) they yield a straightforward representation of syntactic relationships and (c) they can be processed efficiently. From a formal perspective the terminal symbols would

[3]Here the term "word" is not meant in a linguistic sense, but follows from the metaphor.

[4]Otherwise we could just enumerate all words in the language one by one.

usually model words, while words of the formal language would model proper sentences. The non-terminals model syntactic categories, like NP and VP.

Context free grammars are formally on the form $\langle N, T, P, S\rangle$ and fits the definition of a formal grammar above. The productions in P are restricted to have the form $A \rightarrow \alpha$, in other words the context around a non-terminal is not available to rules. A non-terminal could in principle be rewritten at any time when deriving from S. Usually, language membership is not the main point of interest when using CFGs; instead the derivation history normally takes center stage. This is done by creating a *tree* of rule applications, letting the right-hand side of a rule be the children, with the left hand side as the root. Upon a successful parse we have a tree, rooted in S, with leaf-nodes being terminals corresponding to the words in the sentence. In such parse trees nodes with a terminal as a child are often called *preterminals*. See figure 2.2 for an example of a derivation tree.



$S \rightarrow NP\ VP$

$NP \rightarrow N$

$NP \rightarrow VP$

$VP \rightarrow\ V$

$VP \rightarrow V\ PP$

$PP \rightarrow P\ NP$

$N \rightarrow$ john

$V \rightarrow$ reads

$V \rightarrow$ parsing

$P \rightarrow$ about

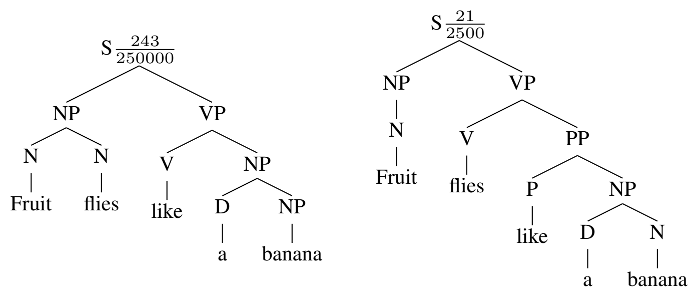
Figure 2.2: A small derivation tree and accompanying rules.

Context free grammars can be processed quickly, or more precisely in sub-exponential time. The main insight is that local ambiguities do not need to be factored in other contexts. The well-known CKY algorithm can parse any context free grammar[5] in $\mathcal{O}(N^3|G|^2)$ time, where N is the length of the input and $|G|$ is the size of the grammar (Kasami, 1965; Younger, 1967).

The key insight here is to use the fact that context free grammars are what the name suggest; we do not have to take anything but the immediate context into consideration. All the possible ways of deriving a category for a sub-sequence of the input do not need to be taken into consideration when this category is used as one of the children in another rule application. In other words, we can represent multiple derivations of a sub-sequence with the same root category as a single entity. This is commonly referred to as *ambiguity packing*. This makes it possible to find and represent an exponential, in the length of the input, number of parse trees in only polynomial time and space.

Most parsers with packing work in two phases: First the *parse forest* is created. Here one would usually apply ambiguity packing. Next the forest is *unpacked*, that is, all the possible parse trees are recovered, either exhaustively, or guided by a stochastic model in such a way that only the most interesting, and hopefully the correct, trees are unpacked.

[5]Grammar rewriting into so-called Chomsky Normal Form (CNF) might be necessary, but without loss of generality.



$NP \rightarrow D\ N$ 0.6
$NP \rightarrow N\ N$ 0.3
$NP \rightarrow N$ 0.1
$VP \rightarrow V\ NP$ 0.6
$VP \rightarrow V\ PP$ 0.4
$N \rightarrow$ fruit 0.2
$N \rightarrow$ flies 0.3
$N \rightarrow$ banana 0.5
$V \rightarrow$ flies 0.7
$V \rightarrow$ like 0.3

Figure 2.3: An example of two trees built from the *same terminal symbols* with different total probabilities, given the grammar fragment. The “missing” rules from the grammar fragment all have a probability of 1. The rightmost derivation has the highest probability.

In contrast to specifying the rules explicitly, a CFG can be implicitly observed by looking at a treebank. For each subtree of depth one $[A\ [B\ \cdots\ C]]$ in a treebank we add the rule $A \rightarrow B\ \cdots\ C$ to the set of rules. This will result in a minimum grammar that covers all the observed data. Having a treebank also allows us to view a grammar in a probabilistic view. PCFGs, probabilistic context free grammars associate each rule $A \rightarrow \alpha$ with a probability. It is necessary to conditionalize on A giving $p(\alpha|A)$, which can be easily observed using frequencies from the treebank. Furthermore we assume that derivations are independent, in other words, PCFGs can be thought of as a Markov process. The total probability of a derivation tree is then the product of all the derivation probabilities for its constituents. This is perhaps not very linguistically motivated, but this makes it possible to keep the $\mathcal{O}(N^3|G|^2)$ run-time of parsing algorithms. Packing becomes slightly more complex when dealing with PCFGs, as the trees now has a probability measure as well. Still the most probable parse can be found without too much trouble by employing a Viterbi scheme. This probabilistic extension to CFGs facilitate parse selection, by ordering candidates by total probability, typically where the most probable derivation is taken as the “correct”. See figure 2.3 for an example of PCFG derivations.

In summary, CFGs employ a rule system which can specify recursive structure. It is worthwhile to take note that CFGs can only create derivation *trees*. Graphs in general is outside the scope of context free grammars. CFGs are arguably suitable for dealing with coarse linguistic notions, and can be processed quickly. A probabilistic extension makes parse selection possible.

2.5 Unification based grammars

While context free grammars are easy to process, they leave a lot to be desired from a grammar engineering position. It is straightforward to obtain a grammar from a treebank, but the expressivity of CFGs is very limiting if we wish to write the rules ourselves, especially if we want to make a precise account of several syntactic processes.

$$\begin{bmatrix} \textit{phrase} & \\ \text{HEAD} & \textit{verb}\begin{bmatrix}\text{AGR}\ \boxed{1}\ \ \textit{3pl}\end{bmatrix} \\ \text{SPR} & \begin{bmatrix}\text{HEAD} & \textit{noun}\begin{bmatrix}\text{AGR}\ \boxed{1}\end{bmatrix}\end{bmatrix} \\ \text{COMPS} & \textit{nil}\end{bmatrix}$$

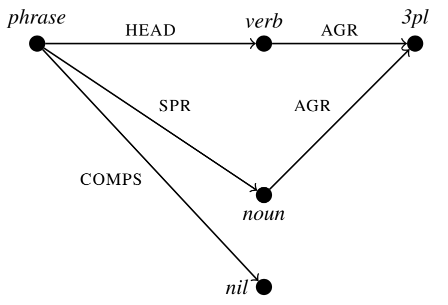


Figure 2.4: A small feature structure drawn as an attribute-value matrix and as a acyclic directed graph. In the AVM, features are written in SMALL CAPS and value types in *italics*. Numbered boxes denote reentrancies. Note how the reentrant features point to the very same node in the graph.

A large part of the problem is that the symbols of a CFG are atomic, and present no inner structure at all. In order to deal with agreement for example, we would need to split the usual VP and NP categories into several new categories, each with any possible value of agreement. Then we could create rules that would only match two phrases together if they have a compatible agreement value. However all other rules would also be affected. A rule that originally was looking for any VP will now need to be modified, as the original VP category has been split into several new categories. We need to reduplicate all rules looking for just VP into new rules for all possible agreement values we split VP in.

This approach to encoding variation of category sub-distinctions quickly leads to a combinatorial explosion; we need to cross multiply all distinctions over most if not all of the grammar. If we want to make several syntactical distinctions instead of just having coarse categories, the resulting CFG could consist of millions of rules and be very hard if not practically impossible to write and maintain.

Unification grammar (UBG) is a formalism that allows succinct specification of fine grained linguistic distinctions. While the symbols of a CFG are just atomic, the UBG "symbols" are *feature structures*. We use the common representation of feature structures as directed graphs with labeled edges and nodes. The edge labels denote *features* while node labels denotes *values*. Furthermore several features can describe the same node. We say that those features are *re-entrant*. See figure 2.4 for an example of a feature structure. A context-free grammar require that its symbols form a finite set, but UBGs can deal with a (countable) infinite number of possible feature structures.

Feature structures are just half the story. The critical point that makes UBG able to work with an infinite set of symbols is *unification*. Informally, unification can be thought of as two-way pattern matching. When some feature structures A and B are unified, the result, C should contain all the information in A and that in B, but only if A and B are compatible. Geometrically we are superimposing the two graphs onto each other. C will have all the features A and B has, valued appropriately, and all reentrancies from both as well. If A and B are not compatible their unification is undefined, which is also dubbed *unification failure*.

Feature structures allows the representation of structured categories and unification provides the means to specify the result of applying derivation rules and the flow of information is to a great extent liberated from a strict context-free perspective. This means that the grammar writer can specify general *rule schemas* which would then be instantiated by unifying the daughters of the rule into the rule schema. This makes it possible to write general rules for general syntactic processes, and let, for example, selectional constraints from the syntactic head be carried over into the rule by unification and reentrancies. For example both verbs and prepositions can have complements, but they place different constraints on what their complements should look like. However the syntactic process of licensing complements works the same in both cases, and one would want to use the same rule schema for all head-complement constructions. Unification based grammars give the grammar writer a much more expressive formalism to write with. Typical unification-based precision grammars usually have just a dozen or at the most a few hundred rules.

Typed feature structures

Although several unification based formalisms exits, we shall not venture further than typed feature structures, as defined in Copestake (2002), in this thesis. Here the feature structures are *typed*, that is, all the possible values a feature can have are types in a *type hierarchy*. We shall give a cursory informal definition below. Carpenter (1992) gives a very thorough treatment about several theories of typed feature structures in general.

The type hierarchy describes the subtyping relation between all the possible types. It is a tree-like structure[6] where the children of a node is said to *inherit*, or be a more specific type than their parent. We say that types "higher" in the tree are *general* and types lower in the tree are *specific*. Types specify *constraints* on both what features are appropriate and what type of values those features can have. A feature structure that conforms to all the constraints *satisfies* the type. There are some important structural restrictions on the hierarchy: There is only one greatest type or root type, called `top`, which is the supertype of all possible well-formed feature structures, and it has no features. In many ways, `top` can be thought of as the *empty* feature structure. Every node must be directly or indirectly connected to `top`. No cyclic sub-typing is allowed in the hierarchy. See figure 2.5 for an example type hierarchy.

Types themselves have corresponding unique most general feature structures satisfying them, an indirect representation that presents a much more intuitive work surface for the grammar (or thesis) writer, and we will allow ourselves to refer directly to this feature structure metonymically. The type-hierarchy itself is a bounded complete partial order on the subtype relation. That means that there is a unique `bottom` type, which is used to denote unification failure. Furthermore every two types in the hierarchy must have exactly one infimum or meet, in other words the greatest lower bound is unique. This is called the greatest lower bound constraint. The infimum of two types denotes the result of their type unification. The greatest lower bound constraint, together with `top` and `bottom`, makes the type hierarchy a complete lattice $\langle \top, \bot, < \rangle$.

However, a type can have several direct supertypes or parents. In other words, a form of multiple inheritance is allowed. Inheritance is strictly monotonic, so a type must be able to satisfy all the constraints of all its parents. Subtypes can also not be more general than

[6]It is not a tree, because a node can have an in-degree higher than one. It is however our opinion that the parent-children metaphor is very useful to get an intuition on what's going on, so we will allow ourselves to use it.

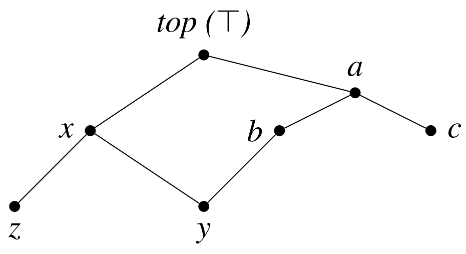


Figure 2.5: A simple type hierarchy. The more general types are higher, and more specific types are lower.

$$\begin{bmatrix} \text{F} & x \\ \text{G} & \begin{bmatrix} \text{I} & a \end{bmatrix} \end{bmatrix} \sqcap \begin{bmatrix} \text{F} & z \\ \text{H} & b \end{bmatrix} = \begin{bmatrix} \text{F} & z \\ \text{G} & \begin{bmatrix} \text{I} & a \end{bmatrix} \\ \text{H} & b \end{bmatrix}$$

$$\begin{bmatrix} \text{F}\,\boxed{1} & a \\ \text{G}\,\boxed{1} & \end{bmatrix} \sqcap \begin{bmatrix} \text{G} & x \\ \text{H} & b \end{bmatrix} = \begin{bmatrix} \text{F}\,\boxed{1} & y \\ \text{G}\,\boxed{1} & \\ \text{H} & b \end{bmatrix}$$

$$\begin{bmatrix} \text{F} & x \\ \text{G} & \begin{bmatrix} \text{I} & x \end{bmatrix} \end{bmatrix} \sqcap \begin{bmatrix} \text{F} & x \\ \text{G} & \begin{bmatrix} \text{I} & c \end{bmatrix} \end{bmatrix} = \perp$$

Figure 2.6: Examples of unifications assuming the type-hierarchy in figure 2.5. The third unification fails, as $\perp$ is the subtype of x and c.

their parents, that is, for all the inherited type constraints, the subtype must have either the same or a more specific constraint. Subtypes can of course add new type constraints. We can interpret this as follows; if there is a type $\mathcal{B}$ inheriting from $\mathcal{A}$ then every feature structure B that satisfies the constraints of $\mathcal{B}$ must also satisfy the constraints of $\mathcal{A}$. Loosely speaking, everything that is a $\mathcal{B}$ is also an $\mathcal{A}$.

We explain unification by introducing *subsumption*. A feature structure A subsumes another one, B, if all the features in A exists in B and are valued such that they are equal or no more general than in A. Any reentrant feature in A must also be reentrant in B. In other words, A is less specific than B, but the information in A and B is not incompatible.

The unification of two typed feature structures, A and B, is the smallest, or least specific, feature structure that is subsumed by both A and B. If the constrains of the type hierarchy cannot permit such a feature structure, the unification *fails*. In the theoretical sense, we have reached the end of the lattice `bottom` and since this is the least possible type it can be regarded as a “singularity”. It subsumes everything, so unification and subsumption involving bottom will always result in bottom. In the practical sense bottom is regarded as failure, and the feature structures are regarded as incompatible. For instance, if a unification fails when doing chart parsing, that hypothesis is discarded and nothing is added to the chart. See figure 2.5 for some examples of unification.

2.6 HPSG

Head-Driven Phrase Structure Grammar (HPSG), canonically defined in Pollard and Sag (1994) building on Pollard and Sag (1987), and also discussed in Sag, Wasow, and Bender (2003), is a theory of syntax. In this section we shall present its most relevant features for our work. HPSG is a generative grammar based on a typed feature structure formalism. The overall philosophical idea of generative grammars is that they can be formally specified as a set of rules, and thus be implemented on a computer. This fits nicely with the ideas of formal grammar and formal languages. Generative grammars should generate "correct" sentences, and disallow malformed ones.

HPSG relies heavily on the notion of *heads*. A linguistic head is the "main" word in a phrase, and it is usually the head word which controls the syntactic function and possible other constituents in phrases. Nouns head noun phrases, and verbs head verb phrases and sentences. Heads can license sub-categorized phrases, a classical example is intransitive, transitive and ditransitive verbs, which take a varying number of syntactically dependent arguments, so-called complements. Verbs and nouns usually take a so-called specifier argument as well, typically subject phrases and determiner phrases, respectively.

Many generative grammars are transformational, meaning that the nodes in phrase structure trees can "move" around, and transformational theories of grammar usually posit several levels of phrase structure with intricate movement rules. HPSG however, is non-transformational, in the sense that there is only one phrase structure and no nodes move.

On an abstract level, HPSG is a set of constraints which are specified from several sources: The type hierarchy imposes constraints on the form of the admissible, i.e. the so-called well-typed and sort-resolved feature structures. Lexical heads constrain the phrases they can combine with, and general grammar *principles* constrain the grammaticality of sentences and how phrases can compose. A sentence is well formed in HPSG if there is a representation satisfying all the constraints.

Most syntactic constructions in HPSG have a head daughter, and possible non-head daughters[7]. It is customary to write HPSG derivations as trees, very much alike the context-free derivation trees discussed above. The head daughter is found at the HD-DTR feature, further non-head daughters are listed at the NH-DTR feature. These map directly onto the daughters of a tree node. If it is not clear from context, the branch corresponding to the head-daughter should be labeled. Note however, that the tree still specifies *one* feature-structure. In figure 2.9 we give an example of a small HPSG derivation. The daughter nodes in a construction do not need to be linearly ordered, nothing in the formalism requires that a HPSG derivation takes the form of a projective tree. In other words, the derivation tree can have crossing branches. However, most computational implementations of HPSG systems do require projective trees, as this facilitates implementation by building upon, and extending, classic context free parsers.

HPSG is strongly lexicalized, and most of the constraints stem from the lexicon, which usually has highly structured entries. The grammar principles are made as general as possible, helped to a great extent by type constraints and unification. The idea is that the constraints from the lexical head serve to restrict what kind of phrases the lexical head licenses. These

[7]Almost all constructions are headed, but coordination constructions murky the picture somewhat.

$$\begin{bmatrix} \textit{phrase} & \\ \text{HEAD} & \boxed{1} \\ \text{HD-DTR} & \begin{bmatrix} \text{HEAD} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figure 2.7: The head-feature principle in AVM form. Note how the HEAD features are re-entrant. HD-DTR denotes the head daughter of the phrase.

constraints are typically "carried over", from daughters to parents by use of re-entrant features. Theoretical HPSG deals with general grammar principles, which most implementations codify as abstract rule schemata. In a bottom up parsing scheme, the applicability of a grammar rule only requires that the constituents on the right hand side are "present", that is, they are derivable from the input words. A rule schemata however, also requires that its possible constituents are unified "into" it, one at a time, and the rule is only applicable if these unifications succeed, i.e. if the resulting feature structure is admissible. In other words, the constituents need to be compatible with *each other* and the rule schema in addition to being derivable from the input. See figure 2.8 for an example of some schemata.

One of the most central principles that stipulate this is the *Head-feature* principle which requires that the HEAD feature of the head-daughter in a phrase is projected onto the HEAD feature of the mother node in headed-phrases. For instance, a transitive verb licenses a complement phrase. A verb-phrase can then be constructed with the verb and a noun-phrase. The head of this new phrase will still be of type verb, and furthermore, all the constraints the verb may have will still be present in the verb phrase. For instance, if the lexical form of the verb specified a first person plural agreement, a specifier to the verb-phrase must follow that constraint. We can write the head-feature principle more succinctly in AVM form as in figure 2.7. In addition, we can observe the effects of the principle in figure 2.9.

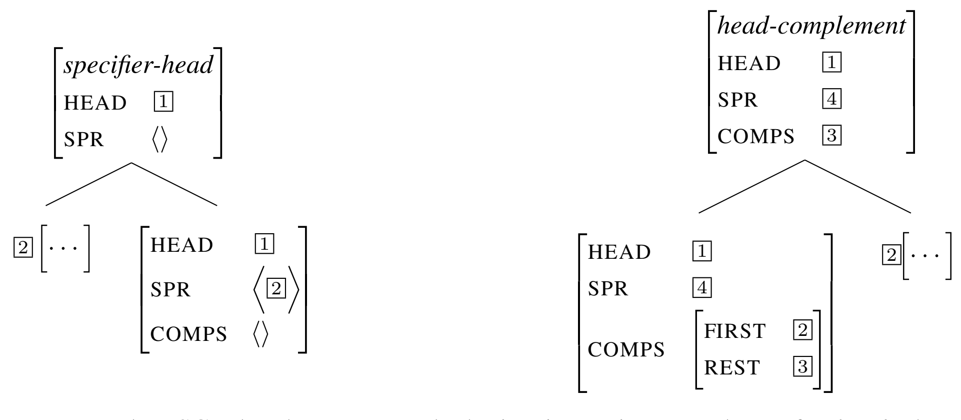


Figure 2.8: Two general HPSG rule schemata, namely the *head-complement* and *specifier-head* schemas. Note how the HEAD-feature of the head-daughter is reentrant with the HEAD-feature of the resulting feature structure. The constraints the head puts on is specifier or complement is expressed with reentrancies, which at first may seem discontinuous, but note that these schemas just are one feature structure, and the tree-branches correspond to the HD-DTR and NH-DTR features. The head-complement schema "picks off" one complement at a time, hence the reentrancy with the rest of the complement list. In the case of a singleton list the resulting COMPS list would be empty. In the other case the list would start with the next element of the original list.

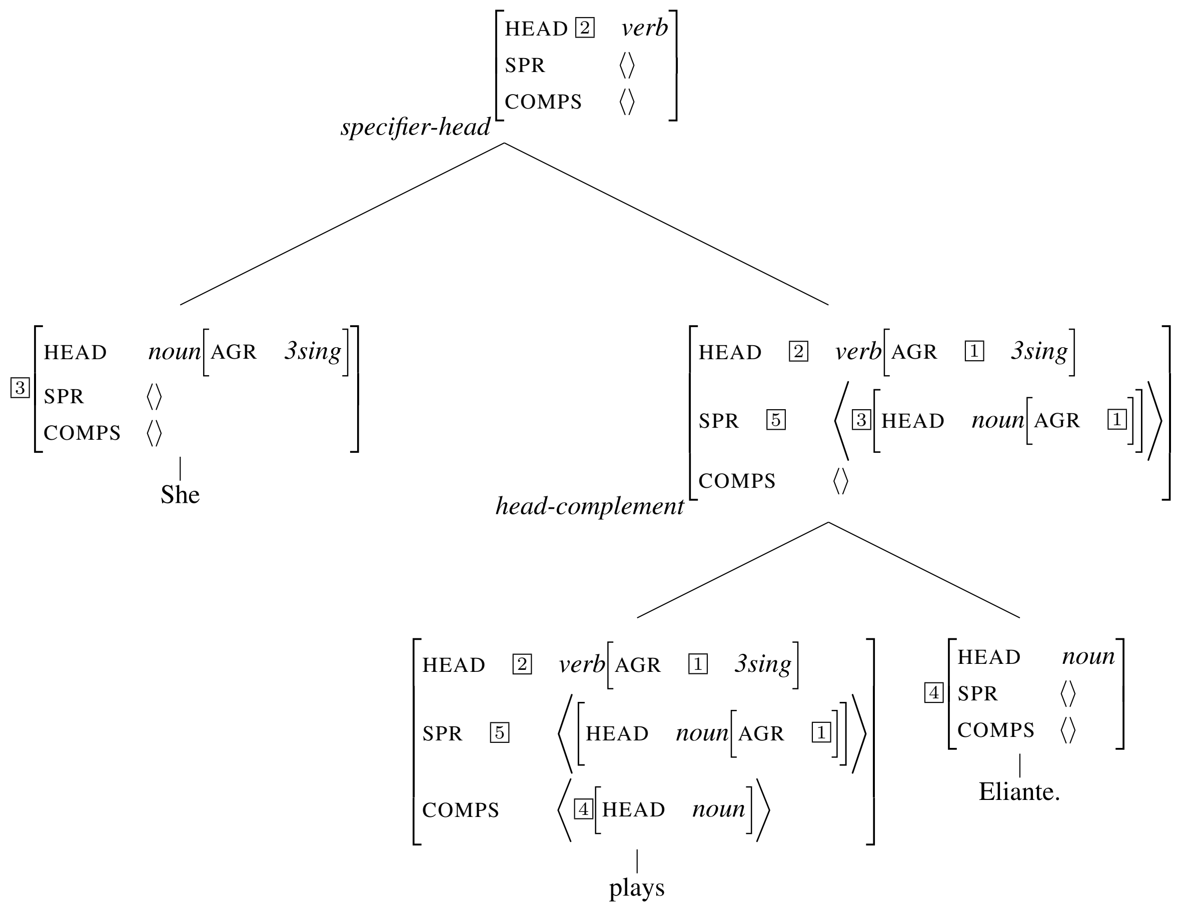


Figure 2.9: A simplified example of an HPSG derivation for the sentence “She plays Eliante.”, built using the schemas above. Note that ⟨ ⟩ denote lists. Lists themselves can be encoded directly in feature structures, where the non-empty list has two features, FIRST of type $\top$ and REST of type *list*. The non-empty list and the empty list, *nil*, are the subtypes of *list*.

CHAPTER 3

Context-free approximation

The harmonicas play
the skeleton keys and the rain

BOB DYLAN

This chapter is about this thesis' main theme: Context-free approximation of unification-based grammars. The motivation and uses for such approximations will be discussed. We shall give a brief discussion about theoretical aspects of formal approximations and we continue with a review of related work on context-free approximation. We shall then introduce the techniques we use in this work, both for the approximation itself, and for the evaluation of the resulting grammars.

3.1 Motivation

Processing unification- based grammars is expensive in both time and space. While the formalism is very attractive from the grammarians viewpoint, processing time can constrain the usability of UBGs, not only in on-line settings where very high responsiveness is required, but also for off-line processing. Long sentences with a lot of lexical ambiguity can take a long time, and use a lot of memory, and for practical purposes both processing time and memory use have to be set to reasonable limits. This can account for sizable coverage loss in several applications.

The classical UBG parser functions roughly like a regular chart based context free parser, but with modifications to accomodate feature structures and unification. Most parsers employ a pure bottom-up strategy, as top-down filtering can be difficult to apply, especially in highly lexicalized formalisms such as HPSG. Operations dealing directly with feature structures, unification and subsumption, tend to make up most of the processing time. The chart-based paradigm necessitates non-destructive unification, in other words, every time a unification is performed, potentially a whole new feature structure is created. Feature structures tend to grow quite large, and the chart can contain hundreds or thousands of edges, each containing a feature structure with hundreds of nodes. A large and diverse collection of strategies have been developed in the past decades to drive parsing time down, working on many stages of the parsing, and with several different philosophies behind them.

For example, in order to reduce unnecessary copying of feature structures, unifiers more suitable to chart-based processing, such as the Tomabechi-algorithm (Tomabechi, 1992) have been applied. The adaptation of local ambiguity packing techniques for UBGs, using subsumption, almost brings back the dynamic programming techniques that make CFGs solvable in polynomial time (Oepen & Carroll, 2000). The storage of feature structures themselves benefit greatly from structure sharing techniques, explored in Malouf, Carroll, and Copestake (2000).

The parser will unify many feature structures as it explores hypotheses, typically most of these unifications will fail. It is therefore worthwhile to detect unification failures as early as possible. Determinig whether a rule-application can be made requires several unifications to take place. Each possible daughter are unified with the rule schema in turn, a rule application is only regarded as succesful when all the daughters are unified with the schema and if the result is valid. However, the daughters will present different levels of constraints, and the order in which these unifications are made can be an important point of optimization. Kay (1989) and Bouma and van Noord (1993) argue for head-driven parsing, where the head daughter is unified first. However, further exploration show that this is not always the case. Oepen and Callmeier (2000) explore so-called key-driven parsing, where the key daughter, often the head, but not always, is the first candidate for unification.

Some unification failures can be determined statically, or in other words, by just analysing the structure of the grammar itself. Type unification itself can be precompiled (Kiefer, Krieger, Carroll, & Malouf, 1999). Unification tends to fail at some critical paths in the feature structures, whereas other paths almost always succeed. The *QuickCheck* filter first checks if these critical paths are compatible before the actual unification begins (Kiefer et al., 1999). We shall discuss the *QuickCheck* filter in more detail later on.

The chart parser paradigm can be combined with other external tools, typically in settings where the search space the parser explores is reduced or constrained. In highly lexicalized theories, such as the lexical entries contain most of the grammatical constraints, With fine-grained distinctions, such as different types of subcategorization, lexical ambiguity rates can be very high. Disambiguation at the lexical level, typically called supertagging (Bangalore & Joshi, 1999), can bring dramatic performance improvements (Dridan, 2009; Matsuzaki, Miyao, & Tsujii, 2007). However, supertagging itself is a hard problem, and errors in tag assignment can reduce both parsing accuracy and coverage.

Other work does away with the chart based backing entirely and explores UBG parsing in a deterministic view. Ninomiya et al. (2011) and Ytrestøl (2011b) explore general shift-reduce parsing with deep grammars. The overall aim is to recover a “script” for unifications which can then be instantiated in a deterministic fashion. This setup greatly reduces the number of unifications the parser must perform to analyze a sentence, but can result in lower accuracy or coverage.

A context-free grammar approximating the UBG in question can be useful for all three of the above approaches. Uses generally fall into three cases; a) for predicting grammaticality, b) as an enabler for probabilistic models, either as a PCFG, or as an information source in other models[1], c) as a parser in its own right.

Context free approximations can be used to expand upon filtering before a unification is attempted, by first checking if the corresponding rule in the approximated grammar exists, a scheme that can be viewed as spiritually extending the *QuickCheck* filter. If the corresponding rule does not exist, the unification is assumed to fail, and the hypothesis discarded[2]. This can reduce the number of expensive failing unifications.

This idea can be taken one step further; we can first completely parse the input sentence with the approximated grammar, and then use the resulting parse forest[3] as an indirect top-down filter on the unification tasks of the main parser. The unification of two edges is only performed if it exists in the parse forest of the approximated grammar. This can further reduce the number of failing unifications, but also, and perhaps more importantly the number of passive edges in the chart that are not part of any complete derivation can be reduced, resulting in not only speedups but also less memory usage.

A context-free approximation can also be useful to refine the results of a supertagger. Most taggers use a linear sequence model, which can give poor performance on long-distance dependencies. A sequence of tag assignments can be tested for grammaticality with the approximated grammar, and ungrammatical tag sequences are then taken to be incorrect. This setup can increase the tag accuracy with a modest increase in tagging time (Matsuzaki et al., 2007; Zhang, Matsuzaki, & Tsujii, 2009). Supertagging is usually approached in a sequence labeling context and can therefore suffer from long-range-dependency errors, which make the use of CFG approximation a possible way to reduce that kind of errors. Another approach is to let a PCFG rank tag sequences itself, by taking the pre-terminal yield the n-best parses as the supertag sequences.

In corpus or treebank driven methods we can also collect rule frequencies and obtain a probabilistic context free grammar. This can be seen as an indirect model of the unification based grammar and used in ranking the items on an UBG-parser agenda (Cramer & Zhang, 2010), for UBG parse selection (Kiefer, Krieger, & Prescher, 2002), either on its own, by taking the most probable parse tree, or as an information source in other parse selection models. Secondly, the PCFG enables another form of parse replay where we only reconstruct the PCFG trees with highest probability. In this approach the responsibility for exploring parse trees is shifted entirely over to the context free approximation.

3.2 Approximation of formal grammars

In this section we shall establish some terminology on how formal grammars are approximating each other. First we will look directly at the string languages the grammars generate. In the section below, G is the formal grammar we want to approximate and $F_1 \cdots F$ are forms of grammars approximating G and $\sim$ is the approximation relation.

[1]The infinite nature of feature structures is problematic for many probabilistic modeling used today.

[2]We shall elaborate more on the correctness of such an assumption later in this chapter.

[3]Or in other words, the parse chart where all the edges that do not contribute to a valid reading have been removed.

$$\begin{aligned}
&\mathscr{L}(G) \supset \mathscr{L}(F_1)\\
&\mathscr{L}(G) \subset \mathscr{L}(F_2)\\
&\mathscr{L}(G) \not\subseteq \mathscr{L}(F_3)\\
&\mathscr{L}(G) \cap \mathscr{L}(F_4) = \varnothing
\end{aligned}$$

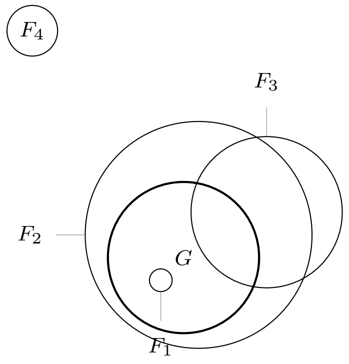


Figure 3.1: Venn diagram of how the string languages of grammars can relate in the universe Σ^*

It is useful to look at "forms" of approximating grammars by looking at the relationships between the string languages, $\mathscr{L}(\cdot)$, of the approximation and the original grammar. These sets can relate to each other in the ways described in figure 3.2. It is reasonable to define the approximation relation such that $F \sim G$ if $\exists g \in \mathscr{L}(G) : g \in \mathscr{L}(F)$, In other words F_4 is not deemed as an approximation of G, as under this definition an approximation has to share at least one word with the grammar it approximates. An F_1 grammar generates a sublanguage of G. In UBG approximation, F_2 and F_3 grammars are the most interesting cases. F_2 is a *sound approximation*, meaning that we can safely conclude that a word not in F_2 is not in G either. However F_2 generates a superlanguage of G, so the reverse property does not hold. F_3 is an *unsound approximation*, where words not in F_3 still can be in G, but words in F_3 might not be in G. The superfluous elements, or the overgenerating part of an approximating grammar; $\mathscr{L}(F) \setminus \mathscr{L}(G)$ will be written as $\mathcal{S}(F)$.

In practice we want approximations where $\mathcal{S}(F)$ is as small as possible. Intuitively F_2 may seem the better approximation scheme, as we can safely discard words not in F_2 from further processing. However the soundness requirement can make $\mathcal{S}(F)$ too big to be useful; the resulting sound grammar is so permissive that almost all possible strings are in F_2. On the other hand, if we still want a small $\mathcal{S}(F)$ the grammar can get so big that the time spent processing F_2 does not make up for the gains we get with the knowledge that a word is not in F_2 and thereby not in G either.

F_3 grammars are the least restricted type of approximations, where no formal guarantees actually hold. In practice however, we can create pretty tight approximations in this scheme, and get benefits in processing. As an added bonus, F_3 grammars can be created with arguably more intuitive and possibly faster algorithms than what F_2 grammars would require[4].

F_3 grammars are typically constructed from treebanks, which lends itself to PCFG-estimation, where a straight-forward approach is to use treebank frequencies directly. Given a treebank F_2 grammars can be estimated, but would require somewhat more involved techniques to deal with the productions that are not present in the data used to estimate it.

[4]In essence, fixpoint iteration on grammar rules is not needed.

3.3 Context free approximation of unification based grammars

So far, we have only studied the string languages of approximating grammars. However, the derivation history and structure is the part we typically are most interested in when doing syntactic analysis. The previous section discussed to what degree approximations can be *weakly equivalent* to each other, but to approximate the derivation history we need to look at some sense of *strong equivalence*, i.e. how rule applications in the original grammar correspond to rule applications in an approximation of it. In this section, and those to come, the core of this thesis will be explored.

	CFG	UBG
Symbols	atomic: $A\ B\ C\quad v, w$	structured: $\mathcal{TFS}$
Productions	$A \rightarrow B\ C \in P$	$A \in P\ \wedge\ A \sqcap B \sqcap C \in \mathcal{TFS}$
Cardinality of symbols	*finite*	*infinite*
Parsing time	*polynomial*	*exponential*

Table 3.1: A table summarizing some high level aspects of CFG and UBG.

Unification-based grammars differ from context-free grammars in some key points tabulated in figure 3.1. Most notably, unification based grammars can have an infinite set of symbols, namely the feature structures. This means that unification-based grammars and context-free grammars, which require a finite set of symbols, do not generate the same class of languages, and thus context-free grammars can, in the general case, only create an approximately equivalent grammar. It follows that UBGs with only a *finite* number of feature structures can be written as CFGs.

However, UBGs still typically employ phrase structure rules, and are in many ways a grammar of trees. This class of tree-structured UBG-grammars can intuitively be viewed as context free grammars, but with a countable infinite numbers of non-terminals. At a minimum, a context free grammar can be created from a UBG just by reducing the set of symbols a UBG operate with down to a finite number. The intuitive way to do this is to *restrict* feature structures, that is, delete nodes and edges so that the grammar only describes a finite number of them. This can be done by creating a "white-list" of features we wish to keep, as in (Krieger, 2004), or "black-list" of features known to have unbounded subgraphs, as in Kiefer and Krieger (2000).

There are two main ways of creating the approximation, *grammar-driven* and *data-driven*. In a grammar-driven approach, algorithms work directly on the grammar. The grammar rules are instantiated, first with lexical entries, and then the resulting feature structures are appropriately restricted. Then these restricted feature structures are again used as daughters for new rule instantiations. This proceeds until a fixpoint has been reached, i.e. no "new" feature-structure can be found (after restriction!) by instantiating an old one with a grammar rule. This approach will create a sound, F_2, approximation of the original grammar.

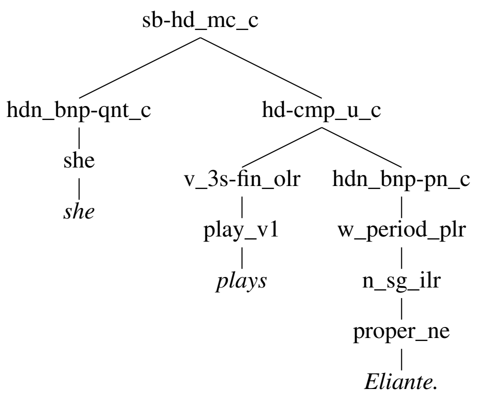
Data-driven approximations do not work on the grammar directly, but work with actual rule applications, either passive items from a parse chart, or on treebanked material. Each rule-application, $A \rightarrow B \cdots C$, will then have its symbols appropriately restricted. The resulting rule is then a permissible context-free rule, and the approximated grammar is then induced from the restricted items in the treebank in the regular fashion: The derivation trees are split into its local productions on the form $A \rightarrow B \cdots C$, and then these productions and accompanying symbols are added to the grammar. When all local productions have been considered, the minimal context free grammar that covers the passive items is yielded[5].

Some practical matters make an approximation more feasible. Context-free grammars are inherently local, with all productions being on the form $\alpha A \gamma \rightarrow \alpha\beta\gamma$. This eliminates any lookaround when applying rules. In other words, the applicability of a rule is only determined by its immediate daughters, which requires an approximation scheme to carry over all relevant information in each context-free symbol. In the case of HPSG this need not very problematic because of the so-called Locality Principle, which requires that only the information immediately available in a feature structure can decide if it can be the daughter of a rule. In other words, no rule can "look" at the daughters of its immediate daughter, and so on. This fits neatly with the limited lookaround we find in CFGs, and makes an approximation more feasible by having the necessary information already locally present in the feature structures one-by-one. Approximation schemes do not have to search any sub-trees to find required information, which means that the feature structures can be approximated in isolation. The algorithm need only work on one rule application at a time.

With the strong equivalent sense in mind, the most interesting approximations are the *reconstructable* ones. That is, the resulting derivation from the approximation specifies exactly *one unique* $\mathcal{TFS}$ if it is well typed. If not, the derivation is ungrammatical, and thus a part of $\mathcal{S}(F)$. This means that we can not only decide if a derivation from the approximating grammar is grammatical, we can also recreate the feature structure corresponding to it, and all the information that was discarded to facilitate the approximation can be deterministically recreated at will. The only requirement on the approximation grammar is that each symbol in the grammar can only have one corresponding UBG rule. That is, only the identity of the rule is needed, the actual feature structure itself is not necessary. Given these rule identities the corresponding $\mathcal{TFS}$ can be found merely by applying the rules in the order the derivation history specifies. This is usually called *deterministic parse replay*.

As it turns out, the smallest reconstructable approximation for $\mathcal{TFS}$ is just the type of the root node in the feature structure, which denotes either a lexical entry or a rule. This "naïve" approximation serves as a baseline for our experiments, which is not only suitable theoretically, but also practically, as this approximation scheme is used in several applications, including the systems of Ytrestøl (2011b) and Cramer and Zhang (2010).

[5] It is possible, when working directly on passive items from a chart, that "useless" symbols are created. Useless symbols are those symbols that cannot take part in any *complete* reading, i.e. starting from S and ending in terminals. Useless rules are rules with one or more useless symbols in them. These symbols needlessly inflate grammar size. However, the chart can first be pruned so only edges that are part in a complete reading are left, which would remove any useless symbols. It is also possible, should the pruning be impractical, to remove all useless symbols and rules from any CFG by static analysis (Hopcroft & Ullman, 1979).



```
sb-hd_mc_c      →  hdn_bnp-qnt_c hd-cmp_u_c
hd-cmp_u_c      →  v_3s-fin_olr hdn_bnp-pn_c
v_3s-fin_olr    →  play_v1
n_sg_ilr        →  generic_proper_ne
w_period_plr    →  n_sg_ilr
hdn_bnp-pn_c    →  w_period_plr
hdn_bnp-qnt_c   →  she
she             →  she
play_v1         →  plays
proper_ne       →  Eliante.
```

Figure 3.2: A small example of an HPSG derivation, and the CFG rules that the naïve approximation would generate.

3.4 Refining reconstructable approximations

The main task of the approximation method we explore here can now be made clear; we expand the least reconstructable approximation in an attempt to create a better, for some values of the word, context free grammar. The baseline method is problematic in the case of HPSG, as HPSG employs very general rule schemas. The resulting least approximation is so permissive that just about anything can be assigned a valid analysis in it. Not even basic syntactic categories like VP, PP and NP will be present in such an approximation. See figure 3.4 for an example of how such a scheme would approximate a simple derivation. In this section, different refinements to this baseline will be discussed.

The starting point is a very uninformed grammar. A better approximation would require adding more information to the grammar, either refining the symbols themselves (and indirectly also the rules), or estimating a probabilistic context-free grammar, and thus augmenting the system with a notion of probability of derivations and rules. Note that these approaches are not incompatible. Symbol refinement can be done in many ways, including internal annotation with values from the corresponding $\mathcal{TFS}$, external annotation from the context around a symbol, neighboring nodes in the derivation tree, or by rewriting the trees themselves.

External annotation is a classic technique in treebank-oriented parsing. Here, limited context is added to the symbols of a context-free grammar. An annotation that resembles the HEAD-feature principle is *lexicalization*, where properties of the word, e.g. either its surface form[6], a normalized version of that, such as a lemma or stemmed form, or its part of speech, is annotated into the usual syntactic category symbols. To let this information "flow" upwards through rule applications, the resulting category will also need to be annotated with the same information, which requires that the original rules in the grammar have a notion on which daughter is the syntactic head. Intuitively the surface word form is an important source of information, especially considering selectional preferences, licensing of other phrases and inflection and other morphological variation. Too aggressive lexicalization, however, can induce sparsity in the resulting grammar, which can give rise to parameter estimation difficulties.

[6] The actual string of characters.

$$\begin{bmatrix} \textit{hd\_imp\_c} & \\ \text{SYNSEM}\,|\,\text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{verb\_full} & \\ \text{AUX} & - \\ \text{VFORM} & \textit{imp\_vform}\end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \textit{valence} & \\ \text{SUBJ} & \textit{*ocons*} \\ \text{SPR} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix}\end{bmatrix} \\ \text{CONT} & \begin{bmatrix}\textit{mrs}\\ \ldots\end{bmatrix}\end{bmatrix}\end{bmatrix} \qquad \mathcal{A} = \begin{cases} \langle\ \rangle \\ \langle \text{SYNSEM.LOCAL.CAT.HEAD} \rangle \\ \langle \text{SYNSEM.LOCAL.CAT.VAL.SUBJ} \rangle \\ \langle \text{SYNSEM.LOCAL.CAT.HEAD.VFORM} \rangle \end{cases}$$

```
hd_imp_c::[HEAD]verb_full:[SUBJ]*ocons*:[VFORM]imp_vform
```

Figure 3.3: An example of a feature structure and an annotation-vector, $\mathcal{A}$, and the resulting annotated context free symbol. Here $\langle\ \rangle$ means the root type of the feature structure.

Another important annotation scheme is factoring in vertical and horizontal context from the derivation history, such as grandparenting, where the category of the grandparent node is annotated into the category of grandchildren node. Zhang and Krieger (2011) present, among other results, several related experiments with a varying degree of external annotations.

However, another possibly complementary source of information is in the feature structures themselves. Following Krieger (2004) symbols can be annotated with some, but not all, feature paths and their corresponding *type*. The features used in annotation form an *annotation-vector*. Should the feature not be present in a feature structure, the annotation defaults to $\perp$. Only types are annotated, that is, if there was a non-atomic feature structure as the value this feature path, only the type would be considered, the rest of the feature structure is ignored, unless "deeper" feature paths were present in the annotation vector. If the UBG formalism requires a finite number of types, which is reasonable, it follows directly that this annotation scheme also will generate a finite number of symbols. Thus we have a context free grammar even if a sound approximation was created.

The case of external and internal annotation is particularly interesting. One could imagine that external annotation, especially word surface lexicalization, captures many of the distinctions that are already present in the feature structures, with but the cost having to observe them indirectly from the string surface itself. In addition, the number of lexical forms can be very high, and create problems with data sparsity, not only in coverage, but also parameter estimation should a PCFG be made. However, the information added with external annotation do not need to overlap with the information available in the feature structures themselves. There is much more information directly in the words themselves, but it is hidden and not structured. For instance external annotation can model extralinguistic phenomena, like domain or genre effects observed in a particular corpus, ontological or world-knowledge effects and so on. This could be regarded as "overtraining" from a purely linguistic perspective, but could arguably, depending on the use case of the approximation, be regarded as beneficial.

A third way of annotation is to collapse the derivation history. Here several UBG derivations are folded into one context free symbol. To keep the reconstructability of the annotation, the identity and order of each contributing UBG derivation must be extractable. The annotation of internal structure, features and corresponding types, and external context, derivation

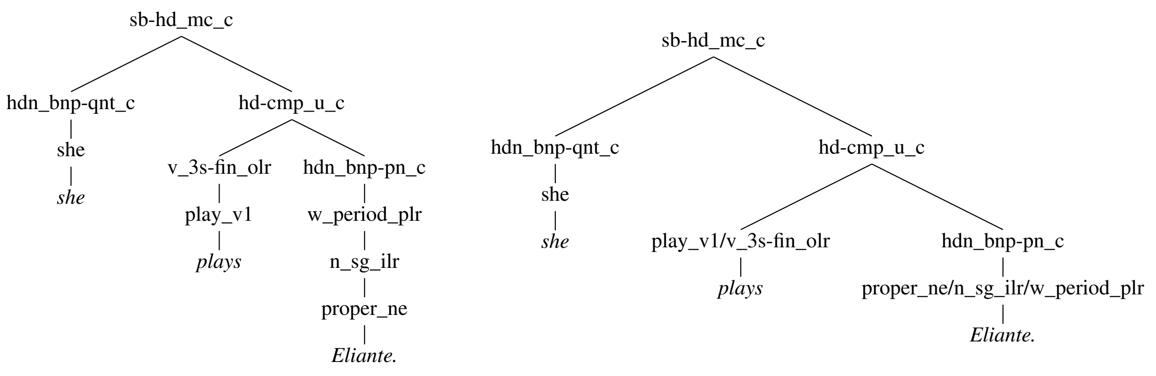


Figure 3.4: Original and lexically collapsed parse trees for “She plays Eliante.”

history and lexical heads, can work well for syntactic constructions, but can discard morphological information in an indirect fashion. In the domain of CFGs, the input token can only be seen from its immediate preterminal, and any information not carried over into the preterminal symbol is not available for further rules.

Typical unification based grammars do not contain a full form lexicon, but employ a morphological engine and a lexicon of lemmas. The morphological rules deal with both inflection and derivation, and these rules will add important information, such as verb tense or agreement constraints, to the feature structures of the lemmas. Because of the strong lexicality of HPSG, discarding this information can result in a very general and permissive approximation. The applicability of these morphological rules cannot be expressed adequately in the a feature-annotation scheme. The system does not only rely on feature structures, but also on the *word form* itself. Typically these rules will start from a lemma and add morphemes concatenatively until a surface form that corresponds to the input token has been found. To approximate this with CFG rules is possible, but infeasible, as the grammar would become extremely large.

Another way to capture this process, but still remain in the domain of CFGs, is apply folding and collapse the derivation trees corresponding to the lexical rules into one derivation, which is then taken to be the preterminal of a word. However, this can create a very high number of preterminal symbols, and can induce a lot of sparsity in the resulting approximation. In addition, lexical ambiguity rates would rise.

The above techniques dealt with symbol refinement. Another important approach is to estimate rule frequencies and create a probabilistic context-free grammar. Approximations with a high level of symbol refinement could potentially disallow a wrong derivation, if its symbols contain the necessary information. Probabilistic approximations however, could assign low probabilities to bad derivations and disfavouring it in the ordering of them (by probability), and instead favor a more probable, and hopefully correct, derivation. However, as stated above, these methods are not incompatible, but stand in an orthogonal relationship. The quality of PCFG-approximations will typically rise with some levels of symbol refinement. If the probabilities are not interesting to the task at hand, one could still stand to benefit from PCFGs, or at least the rule frequency data. Rule frequencies usually have a very long tail, and, for example, the rules occurring with a very low frequency, typically with only one or two occurrences, could be removed. This would create a much smaller grammar, potentially making it significantly faster to process, but at the expense of coverage.

One could, for instance, extend the top-down filtering approach, mentioned in section 3.1 above; where the unification of two edges with a probability under a certain threshold is disallowed. Tuning the threshold can control the amount of filtering. Too much filtering, however, can remove the reading of the original grammar that would score highest according to the parse-selection model of the UBG parser. Should the parser reach a state where no further actions are possible and no complete reading has been derived, the threshold can be set more defensively and parsing allowed to continue. In this setup, the coverage of the original grammar would stay unchanged.

3.5 Related work

Context-free approximation of unification grammars has been studied by several authors in the last decades, based on HPSG and several linguistic theories, such as GPSG[7](Gazdar, Klein, Pullum, & Sag, 1985), LFG (Kaplan & Bresnan, 1995) and PATR-II (Shieber, Uszkoreit, Pereira, Robinson, & Tyson, 1983).

Varying simplifying assumptions have been made. Under the assumption that the number of feature structures is finite, Goldstein (1988) creates a context-free grammar from a HPSG-like formalism. Moore (1999) describes the compilation of finitely valued feature structures, equal in expressive power to CFGs, into context-free grammars without left recursion, which is expanded upon by Rayner et al. (2001).

Carroll (1993) extracts a context-free grammar from ANLT, a GPSG-like formalism. Neumann and Flickinger (1999) create lexicalized tree substitution grammars from a HPSG-parsed corpus. In the context of LFG, where grammar rules may contain regular expressions, Cancedda and Samuelsson (2000) generate approximate rules without the use of regular expressions.

Kiefer and Krieger (2000, 2002) create a sound context-free approximation of both HPSG and PATR-II grammars, using fixpoint iteration on repeatedly instantiated grammar rules as discussed in section 3.3. However, the starting point of the fixpoint iteration is a collection of (abstracted) lexical items. This presents some problems in practical use. First, the resulting grammar is sound, but only on the part of the grammar that is "activated" by these items, that is grammar rules that are transitively derivable from input words used as "seeds" during the fixpoint iteration procedure. Kiefer and Krieger present empirical results on approximations obtained from relatively small corpora of lexical items. It thus unclear if a sound approximation is viable on large corpora. Furthermore, they do not perform experiments on how these "pseudo sound" grammars would perform on unseen data. Secondly, although it was not the case at the time of the work in Kiefer and Krieger (2000), modern precision grammars can contain an infinite variation of surface lexical forms. A true sound approximation of the entire grammar and lexicon might require modification of the original algorithm to be able to deal with the infinitely sized lexicon, if this is feasible in practice at all.

Krieger (2007, 2004) uses data-driven approaches and creates an unsound approximation on instantiated data. Their approximation is similar to this work, but Krieger works on the

[7]Often regarded as a precursor to HPSG.

passive edges of a parse chart, that is, completely instantiated grammar rules. This approach was taken to increase the robustness of the resulting grammar. As no filtering on the passive edges of the parse chart is performed, the resulting grammar may contain useless symbols and productions. These can, however, as Krieger notes, be easily removed post facto algorithmically. In addition, Krieger (2007) present another way of aggregating the resulting CFG productions obtained by the approximation, namely by rule-subsumption (see 4.4). Here, more general grammar rules are preferred over more specific ones. This gives a smaller grammar, but however, and surprisingly, the resulting grammars obtained under rule-subsumption are only slightly smaller than grammars obtained under rule-equivalence.

In Zhang and Krieger (2011), perhaps the work most closely related and contemporaneous to this thesis, treebank-driven approximations are explored. Here, both internal and external annotation is performed to create a probabilistic context-free grammar approximating both the original UBG, and its parse selection component. Zhang and Krieger present empirical data where the resulting PCFG approximated from the LinGO ERG is used as a standalone parser, both with varying levels of internal and external annotation, and with varying amounts of treebanked training material. In their work the main grammar refinement techniques center around external annotation, namely grand-parenting. Internal annotation is explored, but with a smaller scale of annotation vectors than this work. In Zhang and Krieger, efficiency and processing time gains are not the main focus, and the processing time of the different approximations are not reported. However, potential gains in robustness are explored.

Grandparenting gives a good increase in accuracy, but also introduces new challenges. First, grandparenting increases the size of the grammar in a relatively drastic way. This can create sparsity problems. Second, the ambiguity rate increases, and the resulting grammars can become very costly to process. Full parse forest construction might not be feasible in all cases, which can be problematic in scenarios where the top n most probable parse trees were wanted. Last, grand-parented grammars are not immediately suitable for purely bottom up use scenarios, for example in the work of Cramer and Zhang (2010) where probabilistic context-free approximations are used to control parser actions. Here, the approximation is used such that grandparent nodes are not known ahead of time.

In addition, they compare the performance of their approximation with state-of-the-art split-merge grammar refinement (Petrov et al., 2006). Their results suggest that such EM-driven techniques might not be immediately suitable for refining such grammars, due to the high level of granularity already present at the starting point. While still giving a good increase in performance, the availability of internal structure lends itself as a more suitable way of refining such grammars. Secondly, split-merge grammar refinement is a comparatively costly process, and potential gains won by employing this technique could be offset by using more training data, at a level where split-merge might be unfeasible in practice, and faster approximation techniques.

CHAPTER **4**

A framework for data-driven context-free approximation

Were a language ever completely "grammatical"
it would be a perfect engine of conceptual
expression. Unfortunately, or luckily, no
language is tyrannically consistent.
All grammars leak.

Language, 1921
EDWARD SAPIR (1884-1939)

The previous chapter dealt mostly with theoretical aspects of approximation, and the discussion should hopefully generalize to other related formalisms of unification-based grammars and linguistic theories. In this chapter we will discuss some practical matters on the approximation, such as feature selection, and present an implementation for approximation based on a collection of several existing open source tools and grammars. We conclude the chapter by presenting various measurements of the obtained context-free grammars and experiment with different levels of annotation and refinement techniques.

4.1 Practical considerations

Context-free approximations can be obtained in a large number of different ways, and it is not possible to give one single best solution that will fit for all use cases. When the approximation is used directly to aid processing of the UBG, as in contrast to using approximations as a "bridge" between the UBG and other tools or formalisms, the time spent on processing the approximation grammar comes of course in addition to the time spent processing the original grammar. It is therefore important that the time gained from the knowledge obtained by the approximation outweighs the time spent on the extra processing needed to accommodate it.

More annotation creates larger grammars that can become costlier to process and more difficult to extract using data-driven methods. However, the more annotated grammars will typically be closer to the original grammar and be able to reject more ungrammatical derivations or make better statistical inferences. A degree of flexibility in parameterization and empirical knowledge about the trade off CFG approximations may make is a practical prerequisite to successful utilization of the technique. Although several research results show that the technique has great promise in a large number of different scenarios, no off-the-shelf "drop-in" solutions do exist.

For a particular use case, a balance between the tightness of the approximation, the time spent on the approximation, the potential knowledge gained from the approximation and the potential for search and coverage errors must be found.

Tree-rewriting, in particular lexical collapsing, is one of the more dramatic techniques for getting tighter approximations. While it can give a big boost to the performance of the approximation, it can also introduce a lot of coverage problems. The cardinality of the "tag-set" or the preterminals of a typical UBG can be in the hundreds, the cardinality of the preterminals of a lexically collapsed grammar can be several thousand symbols.

4.2 Revisiting sound versus unsound approximation

The choice of sound versus unsound approximation is perhaps the most important one to consider when creating an approximation. Sound approximations are theoretically "cleaner", and some of their uses, in particular as a symbolic recognition filter, see 3.1, cannot give rise to errors, either search errors, since the approximation is not used to rank or reorder the UBG parser actions, or coverage errors, since the approximation is a true superset of the original grammar. A sound approximation used in this way is a safe optimization.

Recall however that sound approximations tend to have a much larger size than unsound approximations which can make them impractical, as too much time is spent on processing the approximation To our knowledge, sound approximations are not commonly used in HPSG processing with (very) large grammars, with the notable exception of Matsuzaki et al. (2007), where a sound, but somewhat light weight, recognition filter is employed to check whether a supertag sequence is parseable.

Several practical matters make unsound approximations more interesting: The approximation can be much tighter, using more feature annotations and possibly tree-rewriting, without losing practicality by creating too big a grammar. Furthermore the likelihood of coverage loss decreases with more training data, and high quality approximations can readily be obtained, especially from syntactically diverse corpora.

All grammars will show some level of overgeneration. One potential benefit of unsound approximation, when treebanked material is used as the data source, is that the overgenerating part of the grammar will not necessarily be carried over into the approximation grammar. Sound approximations could potentially include a lot of symbols and rules which stem from "overgenerating" ways of combining grammar rules. This overgeneration may not be easily observed when using the grammar on actual textual input, and thus not readily apparent to grammar engineers, and would only be made apparent when using fix-point methods to find the resulting sound approximation. Some grammars are bi-directional and hence can be tested for overgeneration by generating licensed sentences from a semantic representation. This technique can greatly reduce overgeneration, but it is unlikely that a large grammar does not have any overgeneration at all present. While the approximation typically will be more

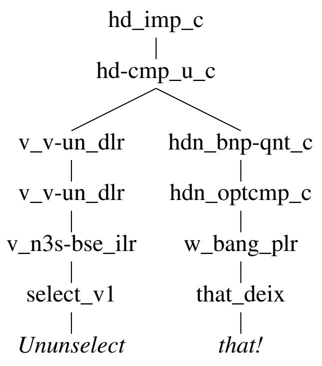


Figure 4.1: A derivation for the somewhat silly sentence "Ununselect that!". Note the two "v_v-un_dlr" nodes, which correspond to the "un-" prefixes in the verb. The "un-" prefix could, in principle, be repeated ad infinitum.

overgenerating than the original grammar, the crucial part of using treebank data is that the size of the approximating grammar is not needlessly inflated by including symbols sponsored by overgenerating derivations.

More "aggressive" uses of approximations, such as using probabilities from a PCFG to constrain parsing, can create search errors both with sound and unsound approximations. Since the potential for errors already have been introduced, in such settings one might as well take a step further and employ unsound approximations.

Approximating the lexicon

We have somewhat tacitly assumed that UBGs are tree-structured languages, and merely a simple symbol refinement on ordinary CFGs. However, in practice, most parsing systems use a morphological component in addition as mentioned in section 3.4. An important consequence of this is that such UBG grammars may license an *infinite* number of lexical surfaces[1], for instance by repeatedly using the deriving "re-" prefix to create new words. See figure 4.1 for an example of a morphological derivation.

This means that we cannot simply map each possible lexical surface to a context free terminal symbol if we want to create a sound grammar. The grammar licenses an infinite number of words, and a context-free grammar requires a finite number of terminal symbols. Of course, in practice the approximation procedure would never even terminate. Since the number of lexical surfaces is finite in corpus driven methods however, as the corpus itself is finite, the possible infinte lexicon do not strictly present itself as a theoretical obstacle for unsound approximations in general, as there are no formal coverage constraints. Highly complex lexicons and morphological components could of course still present problems in practice, especially if the approximation is naïvely mapping surfaces onto terminal symbols.

One could use a special feature structure that is subsumed by all possible lexical entries to handle unknown words. This could, however, be impractical in the sense that the approximation is made to be very loose. A possible improvement is to refine the unknown word

[1]Lemmas and derivational or inflectional morphemes.

handling by having several such feature structures, each with constraints tailored to model the syntactical properties of the unknown word; very much in the spirit of how unknown word handling is performed in modern UBGs today, which already contains such so-called generic word entries. One could imagine that unknown words in the input were mapped directly to these entries in a prepossessing stage before the input was parsed with the approximation.

While this will more or less elegantly create a true superset of the UBG language, it is incompatible with an important refinement technique, namely lexical collapsing as discussed in section 3.4. Just as there might be an unbounded number of lexical surfaces, by repeated derivational morphology; there is also an infinite accompanying number number of lexical rule applications. By collapsing all of these into one symbol, it follows that there could be an infinite number of these collapsed symbols, and thus collapsing is unsuitable when creating a sound approximation, at least without employing more complex backoff strategies.

4.3 Resources

DELPH-IN[2], the DEep Linguistic Processing with HPSG INitiative, is an international multi-site research collaboration focused on deep linguistic processing with grammars in the framework of HPSG, together with an accompanying semantic representation; Minimal Recursion Semantics, Copestake et al. (1999). Several tools exist in the DELPH-IN "ecosystem", and the system built in this thesis is expanding upon the LKB system, Copestake (1992), which is a framework for creating and parsing with typed feature-structure grammars. LKB employs (in part) $\mathscr{TDL}$, (Krieger & Schäfer, 1994), as a specification language. In addition we make heavy use of [incr tsdb()], (Oepen & Flickinger, 1998), a treebanking and grammar performance evaluation toolkit.

The LinGO ERG, Flickinger (2000), is a broad-coverage precision grammar based on HPSG for English, and will be the unification-based grammar we will be approximating in the experiments presented in this work. The ERG consists of around 200 syntactic rule schemata and 100 lexical rules, about one thousand lexical types, or so called "le-types", and has a hand-crafted lexicon with about 40000 stems. The result of several decades of effort, the ERG provides a fine-grained typed feature-structure representation of many linguistic phenomena and constructions. Each word has an le-type, which describes (almost) all the syntactic properties of the word, not only the coarse category such as part of speech, but also properties such as different forms of subcategorization.

The LinGO Redwoods Treebank, (Oepen, Toutanova, et al., 2002) is a treebank of about 45000, covering several different domains and genres. Redwoods differs from other classical treebanks, such as the Penn Treebank, in the way that all sentences are *licensed* with a valid analysis in the ERG. In other words, the treebank is made of manually disambiguated parse results, instead of being directly annotated. This means that the treebank is more transparent with respect to the theory of syntax it is based on. As a corollary, the treebank also contains sentences with no analysis, either because of a coverage hole, or because no good analysis could be found.

[2]See the preface in Oepen, Flickinger, Tsujii, and Uszkoreit (2002)

Corpora

In our experiments, the WeScience (Ytrestøl et al., 2009) part of Redwoods will be used as the starting point for a data source of our approximation experiments. The WeScience corpus consists of one hundred articles from the English part of Wikipedia. In addition, WikiWoods, (Flickinger, Oepen, & Ytrestøl, 2010), can serve as an additional, but somewhat noisy, data source. WikiWoods is a treebank consisting of the entire English Wikipedia as of July 2008, totaling *55 million sentences*, parsed with the ERG. A parse selection model, trained on the disambiguated part WeScience, has been employed to rank candidates automatically. The accuracy has been sampled and Flickinger et al. report scores of around 80% correct or "near-correct".

To measure performance figures of the obtained approximations we need to partition the data into at least two sections, usually dubbed training data and held-out or testing data. The approximation proper will be extracted from the training data and aspects of the resulting grammar can be measured on the held out data. Following tradition, the thirteenth section of WeScience (WS13), which consists of 1001 sentences will be used as the held-out dataset. Of those 1001 sentences, the ERG, as of version 1010, parses 887 sentences. The other hundred or so sentences are either ungrammatical, with respect to the grammar, or the parser could not find an analysis in a timely fashion. Of the 887 sentences that do have an analysis, 785 have been manually disambiguated and verified as correct readings. The rest have either not been disambiguated yet or marked as incorrect, that is, the grammar provides one or more analyses for the sentence, but not a correct one. In this case, a classical treebanking scheme could just annotate the sentence with the correct analysis, but in the grammar-supported Redwoods corpora; this would require that the grammar is changed as well.

The disambiguated part of the thirteenth section of WeScience, from now on referred to just as WS13, has an average sentence length of 14.5 words. This number however must be interpreted with care, as the distribution is somewhat bimodal with a $\sigma = 10.3$. This is mostly due to the genre of Wikipedia articles, where comparatively short sentences make up article titles and headings. See section 5.2 for a breakdown of sentence lengths.

4.4 Implementation and extraction procedure

This work implements a flexible system for creating context free approximations and automatically reporting on several varied measures of the obtained grammars. The system is implemented using both LKB and [incr tsdb()]. Profiles in the [incr tsdb()] system are used as the data source for the extraction, and we use [incr tsdb()] code directly to select the trees that will be used. This allows for easy portability to other DELPH-IN grammars, and furthermore one can select items from a profile using complex predicates if so desired. In addition, our system can *merge* CFGs, or in other words, compute the union of several context free grammars.

The approximation itself proceed in roughly the same way as the rule equivalence method used in (Krieger, 2004). Each derivation tree is visited top down, and for each derivation in the tree we check if any tree rewriting rule should be applied, and if so transform the tree accordingly. Then the feature structures in each local subtree are labeled with the current annotation vector, giving one context free symbol for each feature structure in the local subtree. Now, the labeled nodes in the derivation tree show only a finite amount of variation and can be mapped directly onto context-free rules, which then are collected and aggregated in order to collect the complete grammar and estimate the rewriting probabilities.

The top feature structure of each derivation tree, i.e. the final HPSG sign, is annotated in the regular fashion, but stored in a special vector. Finally the start-symbol S can be computed. S can rewrite to each of the symbols obtained from approximating the final signs. Since the system works on complete derivation trees, a *proper* CFG will be created; the resulting grammar will not have any useless rules or symbols, see section 3.3. The following figures explain core parts of the extraction procedure in detail:

```
Algorithm: Annotating feature structures
Data: $\mathcal{TFS}$ T, annotation vector $\mathcal{A}$
First, annotate the type of the complete feature structure.
string Symbol ← RootType(T);
Then, annotate additional feature paths and their values:
foreach FeaturePath FP in $\mathcal{A}$ do
    Symbol ← Symbol + FP + Type-of(Value-at(FP,T))
end
return Symbol;
```

Figure 4.2: The procedure to annotate feature structures with feature paths from an annotation vector. Note that $+$ here is meant as string-concatenation. Type-of and Value-at are functions to access the feature structures. Type-of returns the type of a feature structure. Value-at returns the value, i.e. another feature-structure at the end of a feature path.

```
Algorithm: Lexical collapsing
Data: Derivation D
root ← DerivationRoot(D);
daughters ← DerivationDaughters(D);
Only perform lexical collapsing if at a lexical rule.
As a sanity check the derivation can only have one daughter.
if LexicalRule(root) AND daughters.size() == 1 then
    collect all the daughters transitively into a list:
    trans-daughters ← CollectDaughters(First(daughters));
    find the terminal at the end of the chain:
    terminal ← FindTerminal(First(daughters));
    Annotate this derivation chain with all root-types:
    string Symbol ← RootType(root);
    foreach daughter t in trans-daughters do
        Symbol ← Symbol + RootType(t);
    end
    return ⟨symbol,terminal⟩;
else
    Otherwise return the derivation unmodified
    return D;
end
```

Figure 4.3: The procedure for performing lexical collapsing. If Derivation D is a product of applying a lexical rule, then a new collapsed tree is returned. Otherwise the derivation is unchanged

```
Algorithm: Extracting CFG-rules from an [incr tsdb()] profile
Data: TSDB profile prof, Predicate pred
Select items from the tsdb profile according to the predicate
Trees T ← TSDB-Select(prof,pred);
Create a list of context-free rules, initially empty
CFG-Rules R;
foreach Tree in T do
    Reconstruct the derivation so we have access
    to the feature structures
    DerivationTree DT ← Reconstruct(T);
    Now, traverse the derivation tree top-down left to right:
    foreach Derivation D in Traverse(DT) do
        First, check if lexical collapsing should be applied:
        if Collapse(D) then
            If collapsing, add the obtained rule to the grammar:
            $\langle Symbol, terminal \rangle$ ← LexicalCollapse(D);
            R ← R $\cup_{\equiv}$ $\langle Symbol \rightarrow terminal \rangle$;
        else
            Otherwise, we create the local tree and annotate that:
            mother ← DerivationRoot(D);
            daughters ← DerivationDaughters(D);
            Annotate the mother node:
            LHS ← Annotate(mother);
            Annotate each daughter in turn:
            RHS ← $\langle\ \rangle$ ;
            foreach Daughter d in daughters do
                RHS ← RHS $\oplus$ Annotate(d);
            end
            Finally create the context free rule and add it
            R ← R $\cup_{\equiv}$ $\langle LHS \rightarrow RHS \rangle$;
        end
    end
end
return R;
```

Figure 4.4: Extracting CFG-rules. Note that $\oplus$ means list concatenation. Internal annotation is not performed on the lexical collapsed projections. As we elaborate on below, lexical types describing most syntactical information are used in the LinGO ERG. When a derivation tree is collapsed any further annotation do not contribute new information, as this information is already present in the lexical types themselves.

We use the $\cup_{\equiv}$ here to denote the union of equivalent context free rules, i.e just the regular union, to distinguish from other possible ways of aggregating rules. Recall from section 3.5 that Kiefer and Krieger explore in several of their works other ways of aggregating rules, for instance under subsumption, $\cup_{\sqsubseteq}$.

To estimate a PCFG we need to keep track of frequency information as well. Then the probability of each rule can be estimated using maximum likelihood estimation. I.e. $P(A \rightarrow \alpha) = \frac{f(A\rightarrow\alpha)}{f(A\rightarrow\cdot)}$. The estimation is done after the CFG has been extracted in full. Furthermore the system can output grammars in several formats which might be usable for other parsers and tools, including the format used by the DELPH-IN parser PET (Callmeier, 2000) for PCFG parse-selection models, on BitPar (Schmid, 2004) format (see secton 5.3) as a rule and lexicon file and on an internal format where grammars and necessary component files, such as symbol-tables, can be serialized and deserialized.

The system can *merge* multiple grammars, both symbolically on rules and symbols, but also aggregate the new rule frequencies. This facilitates parallelization of the extraction procedure. When working with larger corpora, such as the WikiWoods, several component grammars, working on different subsets of the corpora can first be extracted, and then merged together to form the final result. Most processing time is spent on instantiating the feature-structures in the training data. The actual annotation and aggregation itself is very fast. The extraction itself is trivially parallel, while the merging could be parallelized in a $O(logn)$ scheme. However, as merging is a comparatively fast procedure, a sequential scheme is good enough in practice. The system implements a “coordinatior” which can automatically partition the training data and run the different extraction jobs and then finally merging the results back together giving the final result.

4.5 Feature selection

When creating approximations with internal annotation, it is important to use “good” feature paths. All features describe some kind of information, recall however, from section 2.5, that all values stand in a type hierarchy. By taking the subtype and supertype relations in the hierarchy into active use when writing grammars, the type constraints on features are often made to be so-called “underspecified”, which means that a type of a feature is specified with relatively general constraints. However, the abstract rule schemata often specify several reentrancies, which means that the instantiated rule also will inherit these. Since reentrancies enforce a token identity of the reentrant features in the resulting feature structures, these underspecified value will usually be unified with a more specific value from another part in the instantiated feature structure.

This means that a large part of the features in typical HPSG grammars can be viewed as specifying *constraints*. These features are important when creating an approximation. By using internal annotation with some of these features, an indirect model of the constraints can be reflected in the approximation. Another part of the features in typical grammars are used to concatenate information, typically semantic composition. These features very rarely provide any constraints, and unifications involving these almost always succeed. These types of features rarely contribute much to approximations, and since they rarely contribute any constraints, they are classical examples of features that are not chosen for internal annotation. Furthermore, whereas it is an open question whether the purely syntactical features in typical grammars actually do describe an unbounded number of feature structures, the features used to concatenate information clearly do. Therefore, one would typically want to make use of the constraining features and disregard the concatenative features when creating an approximation, as the latter features can rapidly create a very large amount of variation in the resulting approximation, and thus lead to coverage problems.

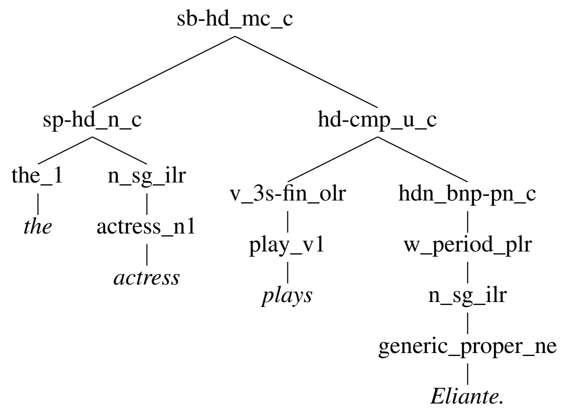


Figure 4.5: The English Resource Grammar derivation for the sentence “The actress plays Eliante”

For example, recall the ERG derivation for “She plays Eliante.” in figure 3.4. Here we use a somewhat different example; “The actress plays Eliante”. One of the rule schemata that appears here is the subject-head schema, here in the form of

$$sb\text{-}hd\_mc\_c \rightarrow sp\text{-}hd\_n\_c\ hd\text{-}cmp\_u\_c$$

which is also the context-free rule the naïve approximation would generate. The subject-head schema should in general only be applied to verbs, which will typically look for a subject phrase which is headed by a noun. But in the approximated context-free rule, none of this information is available: The types of the two constituents, *sp-hd_n_c* and *hd-cmp_u_c*, taken in isolation, only denote which rule schemas that were used to generate them. However several linguistic phrase types are modeled with the head-complement schema. In our case, the transitive verb "plays" allows a noun complement. But prepositions, for instance, can also have noun-phrase complements as well. Thus there is no way to tell if the *hd-cmp_u_c* here is a verb phrase or a preposition phrase, it can be a phrase resulting from any head-complement instantiation, in fact, this approximation could license subject-head constructions between noun phrases and prepositional phrases.

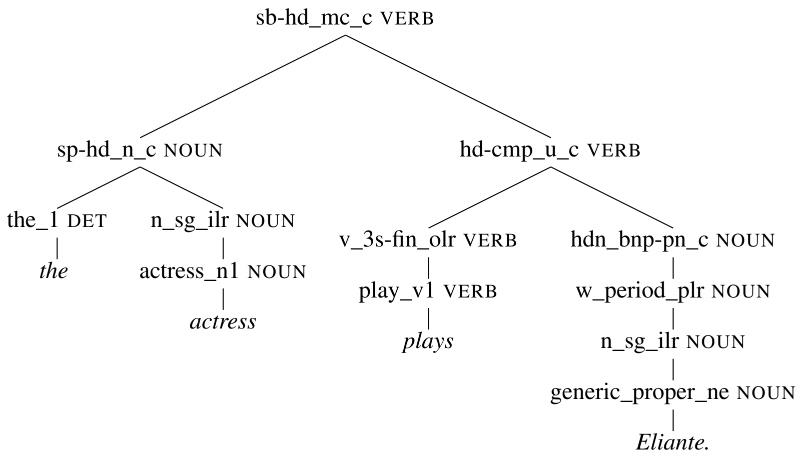


Figure 4.6: "The actress plays Eliante" with annotated heads

Internal annotation of some of the "constraint specifying" features, as discussed above, can capture some of the constraints that both the rule schemas and the feature structures specify. For example, if we specify the type of the HEAD feature in all feature structures, as shown in figure 4.6, the context-free rule in our approximation would become:

$$sb\text{-}hd\_mc\_c\text{:VERB} \rightarrow sp\text{-}hd\_n\_c\text{:NOUN}\ hd\text{-}cmp\_u\_c\text{:VERB}$$

Now the context-free rule is more informative, as part of the original constraints are annotated into the symbols of the rule. The approximation cannot create subject-head constructions between noun phrases and prepositional phrases, unless of course such a construction was part of the training data. However, type hierarchies are not available in context-free grammars. This means that heavy use of underspecification in the original grammar can create sparsity problems in the approximation. For example, the original grammar can specify a type hierarchy where *verb* is a supertype of *unaccusative-verb*. Recall from section 2.5, that when *unaccusative-verb* is a sub-type of *verb* it is a compatible type. When the grammar specifies a feature structure of type *verb*, a *unaccusative-verb* typed feature structure is compatible. This information is not carried over in the approximation; where context-free rules annotated with the type *verb* are incompatible with rules annotated with *unaccusative-verb*.

Therefore, the number of internal annotations used in the approximation must be balanced with the size of the available training data and the nature of the grammar itself. Some feature paths, typically HEAD, only show a small or moderate variation in types and can be good candidates for internal annotation. Other paths show a great deal of variation, for example features regarding semantic selectional preferences, which makes those features bad candidates for annotation in most cases.

The unification of two feature structures tends to fail more often at some critical paths in the structures. The so-called *QuickCheck* filter, (Malouf et al., 2000), takes advantage of this notion, and checks if the feature structures are compatible before the unification proper takes place. These critical paths can be found by using a special unifier which counts when the type-unification of a feature fails, but also continues the unification[3] so that additional "deeper" paths can be obtained.

When tuning the internal annotations used in the approximation, the *QuickCheck* filter serves as a major source of knowledge. The top-ranked feature paths are the "most informative" when it comes to specifying constraints. However, just taking the top n feature paths from *QuickCheck* does not necessarily create the best approximation. Several high-ranking features show too much variation of observed value types to be useful, and including these will create coverage problems.

Other highly ranked features are strictly subsuming less ranked features, and including both would have no effect. For instance *SS.LOCAL.CAT.VAL.SPR*, which denotes if the specifier list is empty or not is subsuming its own subfeature $\cdots$*.SPR.FIRST.LOCAL.CAT.HEAD* which denotes the *HEAD* feature of the first specifier. This trend appears in most of the valence and modifying constructions in typical grammars. The first one shows much less variation, but is also much less informative than the last. The upside to this is that one can include the latter if there is enough data, but can fall back to the first if the coverage turned out to be too low to be useful. The list of the feature paths used in this work is given in appendix A.

4.6 Measures on Grammars

Context-free grammars are finite objects, and it makes sense to quantify their size. See figure 4.7 for a description on the measurements we shall employ here. Recall from section 2.4, context-free grammars have *terminals* and *non-terminals* symbols and production rules.

The cardinalities of the set of symbols and rules vary with annotation. Holding the amount of data used in the approximation constant, longer annotation vectors will create bigger grammars. Typically, more annotated grammars will be able to make more fine-grained distinctions, but at the cost of coverage.

[3]Normally the unification will stop immediately when two incompatible paths are found, as the result will then always be $\perp$.

Below we define some measures on context-free grammars. Recall that CFGs take the form $\langle N, T, P, S\rangle$, where N, T are, respectively, sets of non-terminal and terminal symbols. P is a set of productions, rewriting rules from *one* non-terminal symbol into any string of terminal and non-terminal symbols. In addition to just terminal symbols, it is interesting to quantify how many words, (W), the grammar describes, that is a terminal symbol with an accompanying preterminal. For instance, while "flies" is just one terminal symbol, it could make up two distinct words, both as a noun and as a verb. Here a word is meant to be a *pair* of a terminal symbol and the preterminal symbol that produces it.

While these measures quantify size, they tell us nothing about coverage. Of course, to quantify coverage, we need more than one grammar. Therefore, instead of using all available data to approximate a grammar, a fraction of it is held out. This can then again be used to estimate another smaller grammar, and we can measure how well approximation, obtained from the now reduced data-set, covers the approximation obtained from the held-out part. For this to make sense, both approximations must be obtained with the same annotations and configuration.

Production Coverage (PC) and Lexicon Coverage (LC), as defined in figure 4.7 tell exactly how much these grammars overlap. Lexical Coverage and Tree Coverage, however, measure how well the approximation covers the held-out corpus in general. Lexical coverage (LT) puts a number to how many of the sentences in the held-out corpus that the approximation has matching terminal symbols to cover all the words, while Tree Coverage (TC) tells us how well the entire derivation trees are covered. Tree Coverage is particularly interesting when the approximation is used as a stand alone PCFG parser, as it gives an upper bound to the performance of parse selection.

N,P The cardinality of the sets N, P

W The number of *distinct* words and preterminal pairs.

C The ratio of sentences in some corpus that can be given *any* analysis by the grammar.

PC Production coverage $\frac{|P_T \cap P_H|}{|P_H|}$

LC Lexicon coverage $\frac{|W_T \cap W_H|}{|W_H|}$

LT Lexical coverage; the fraction of sentences in a corpus without lexical holes.

TC Tree coverage; the fraction of the *lexically covered*, sentences in some corpus covered by the production rules in T.

Figure 4.7: Measures on some aspects of grammars. Assuming two grammars T and H we quantify how well T covers H. Note that T N P PC LC are "static" measurements that are obtained just from the definitions of T and H, while C, LT and TC are measures obtained with a corpus as well, typically the held-out part of the experiment.

Evaluating the performance of an approximation, in vivo, for instance, by quantifying how much it can improve the performance of a task where it is employed, is a practical approach. However, there are still several "internal" measures that are readily available, and examining these can give more insight in how much an approximation can bring to the table. Smaller grammars might be processed faster, but not necessarily. Depending on implementation choices in the parser, either the number of symbols or the ambiguity rate can be the dominant factor in run-time. Caution should be used when predicting run-time solely on these measures, for instance more symbols can reduce ambiguity rate and increase parsing speed, see table 5.2.

4.7 Extraction experiments

In the experiments in this section, we use a fixed list of feature paths, and we use the top N features, from the list in appendix A, to annotate feature structures. Table 4.1 shows how grammars obtained from the data-driven method grow with various levels of annotation. In this first experiment, no tree rewriting is performed. More annotations will necessarily create bigger, or at least equal sized, grammars. We argue that the number of distinct word and preterminal-pairs is not particularly interesting. When parsing with a grammar, the terminal symbols are already laid out as the string of input words. Lexicon coverage, however, is an important measure. Data driven CFG-approximation will most likely obtain grammars with lexicons that do not cover the entire lexicon of the original grammar. Low lexical coverage can severly limit the usability of the approximation grammar. However, more sophisticated methods, instead of just mapping each input word to a terminal symbol can be used, as we will detail below.

The grammar with zero annotations, A = 0, is the smallest reconstructable approximation, which serves as a baseline for how context-free approximations could be refined. There are two main things to note here. First, with more annotations the grammar size increases, both in the number of non-terminal symbols and in the number of production rules. Since we are holding the amount of data constant the coverage on the held out part decreases.

Secondly, and maybe contra-intuitive, the number of words are constant throughout all sizes of annotation vectors. However, the ERG uses so-called lexical types to describe the syntactic properties of words. These `le`-types contain almost all the information of a lexical entry, apart from the surface form and associated properties of that, such as phonological onset. In addition some selectional preferences, especially phrasal verbs which select only certain particles, i.e. “walk [about]”. Therefore, additional feature annotations on the lexical type which do not pertain to lexeme-specific information do not split words into more subtypes, as the lexical type is already annotated, since that is the requirement for reconstructability.

A	W	N	P	W_h	N_h	P_h	C	PC	LC	LT	TC
0	20245	886	8650	3782	557	2791	36%	90%	69%	36%	86%
1	20245	1280	9921	3782	725	3021	36%	89%	69%	36%	85%
5	20245	2546	17626	3782	1242	4614	35%	83%	69%	36%	77%
10	20245	3600	20815	3782	1593	5085	35%	81%	69%	36%	74%
15	20245	4503	22443	3782	1858	5279	34%	79%	69%	36%	73%

Table 4.1: Number of Annotations (A), words (W) and non-terminal symbols (N), Productions (P), Coverage (C), Production coverage (PC), lexicon coverage (LC), lexical coverage (LT) and tree-coverage (TC), trained with WeScience 1-12 and tested on WeScience 13.

At a first glance, the lexical coverage of our approximation seems very low, at a poor 36%. This is due (in part) to the somewhat naïve assumptions on tokens that the approximation algorithm uses. No token normalization, such as down-casing or separation of punctuation marks into separate tokens, is performed. The ERG uses a somewhat idiosyncratic way of dealing with tokenization, and treats punctuation as morphological pseudo-affixes. That means that what would otherwise be known words can appear as unknown if for instance they are in the sentence final position with an affixed period. This necessitates a more careful design of how to map the surface forms into the approximation grammar. A coverage of about one third means that the approximation is impractical in almost all scenarios.

However, it is also possible that the approximation abstracts away from surface tokens, and uses a *lattice based* input format, where the lexical parsing has already been performed by another system, typically the morphological engine of the underlying UBG. While this has not been fully implemented, we simulate this approach by extracting the lexical production yield. These are all the possible analyses of a lexical item, including derivational and inflectional morphology and word level ambiguity. This is extracted from all the words in the held out corpus, and added into the grammar obtained from the training data. This simulation means that the ambiguity form the lexical parsing component is preserved, which means that this yields an accurate, though somewhat impractical, estimate of the grammar size and performance measures.

A	W	N	P	W_h	N_h	P_h	C	PC	LC	LT	TC
0	20245	886	8650	3782	557	2791	100%	90%	100%	100%	89%
1	20245	1280	9921	3782	725	3021	100%	89%	100%	100%	73%
5	20245	2546	17626	3782	1242	4614	99%	83%	100%	100%	57%
10	20245	3600	20815	3782	1593	5085	97%	81%	100%	100%	52%
15	20245	4503	22443	3782	1858	5279	95%	79%	100%	100%	50%

Table 4.2: Static measures for a lattice based simulated grammar with varying degrees of annotation.

Table 4.2 show a simulation on how such grammars coupled with a lexical parser would perform. Note how the tree-coverage drops dramatically with more annotations. Even with one annotation, which is still a very coarse approximation, the total possible correct coverage is only 73%. This reveals that using only WeScience 1-12 as training data might be too small to obtain reliable approximation grammars. However, even as these numbers might look grim, for most practical purposes for example in the case study in the next chapter, we can rewrite the parse trees obtained by the more annotated grammars into the smallest reconstructable form, in essence, *unannotating* the symbolss. This increases coverage to the level of the baseline unannotated grammar. Furthermore the regular coverage (C) is still quite high. Even though the correct parse tree might not be found by using the grammar , the approximation is still quite robust. This can be useful if only near-correct results are needed, and the approximation is employed mostly to increase the robustness of the original grammar.

With an original lexical coverage of about 36%, and therefore mostly only covering very short sentences, the rest of the presented experiments are performed on the simulation of lattice based input. We therefore drop the columns LC and LT from the tables, as these will now always be 100%.

Applying lexical collapsing is perhaps the most dramatic technique to refine approximations with. Table 4.3 shows how such grammars compare to only the annotated ones before. The number of words only increases slightly, which is expected, now we have several more collapsed preterminals, and hence the number of distinct word and preterminal pairs should increase. We see that grammars obtained with lexical collapsing describe a much bigger number of non-terminal symbols. The number of productions rises only somewhat however. It is therefore reasonable to assume that these grammars are sparse, that is the ratio of possible productions reachable, or "activated" by input terminals is pretty low. This is further supported by the drastic falloff in coverage. While the grammar annotated with 15 feature paths without lexical collapsing had a coverage of 95% the corresponding lexical collapsed one had only 68%, a coverage that is pushing the limits of practicality in most applications.

A	W	N	P	W_h	N_h	P_h	C	PC	TC
0	22260	3245	11787	4007	1093	3005	90%	82%	62%
1	22260	3411	12621	4007	1174	3173	88%	82%	61%
5	22260	4424	20142	4007	1622	4686	80%	78%	51%
10	22260	5096	22268	4007	1859	4973	75%	77%	45%
15	22260	5937	23837	4007	2114	5158	68%	75%	44%

Table 4.3: Grammar sizes and coverage with lexical collapsing.

Another technique is to normalize lexical surfaces, for instance removing punctuation. As we already simulate the availability of a lattice based input, and thus the word-forms with punctuation marks are already present, we perform a process we call *lexical generalization* instead. Here the morphological derivation rules pertaining to punctuation are ignored, and removed from the derivation tree altogether before the approximation begins. One might say that lexical production yield is taken "modulo" punctuation. Theoretically this could break reconstructability, as a complete derivation tree from the approximation could now specify several derivations in the original grammar, with and without punctuation rules. However, in practice the input words are still present and unmodified, and in order to decide the correct derivation, the morphological engine of the original grammar can be applied. In this case the approximation is "fed" generalized entries, and if the result were to be reconstructed, the corresponding punctuation rules are inserted again. In most cases punctuation rules are deterministic, in other words, there is no ambiguity attached to them. If punctuation is present in the input word form, the corresponding punctuation rules must be applied. Furthermore these rules must also be applied at specific points in the derivation tree, namely after regular morphological processing, but before syntactical processing.

Lexical generalization gives a significant increase in coverage. In addition the grammars are much smaller, both in the number of non-terminals and in the number of rules, which may boost processing speed. However, coverage still drops quickly with the more annotated grammars. The grammar annotated with 10 feature paths give a coverage of 86% which might be practical in some situations, but it is still a somewhat underwhelming number. Annotating

A	W	N	P	W_h	N_h	P_h	C	PC	TC
0	21962	1583	9213	3989	767	2640	98%	87%	**68%**
1	21962	1752	10032	3989	848	2806	98%	86%	66%
5	21962	2765	17564	3989	1269	4327	91%	80%	52%
10	21962	3437	19698	3989	1533	4620	86%	79%	48%
15	21962	4278	21285	3989	1788	4809	80%	67%	47%

Table 4.4: Grammar sizes and coverage with lexical collapsing *and* lexical generalization.

d	A	W	N	P	W_h	N_h	P_h	C	PC	TC
WS	10	21962	3437	19698	3989	1533	4620	86%	79%	48%
WS	33	21962	8283	34472	3989	2825	6383	46%	69%	34%
WW	10	459244	9105	163652	3989	1533	4620	97%	94%	80%
WW	33	459244	39962	406760	3989	2810	6333	97%	92%	59%

Table 4.5: Grammar sizes and coverage for an approximation with 10 and 33 internal annotations estimated from WeScience and a small part of WikiWoods. Column (d) denotes what data was used in the approximation, here "WS" is WeScience 1-12 and WW is the discussed WS + WikiWoods subset. Both lexical collapsing and lexical generalization is employed, in addition to simulated lattice-based input. WS-10, the approximation obtained from just WeScience is included for reference.

with 15 paths gives even less coverage. Note especially how tree coverage drops dramatically. However, in situations where the context-free symbols can be "unannotated", a total tree coverage of 68% can be achieved.

Adding more training data

WeScience contains about 8000 fully disambiguated sentences, and as such is a modestly sized treebank. The WikiWoods corpus contains millions of sentences. We can increase the coverage of the approximation by fueling it with more data. Table 4.5 describe measures of an approximation obtained from 433870 sentences, constituted of WeScience sections 1-12 and a subset of WikiWoods. The exact subset is described in appendix B.

More data gives much better coverage, but the approximation is now several times larger, having an order of magnitude more rules, and almost three times more non-terminal symbols. This can make the grammar unwieldy, and one could lose one the original attractive benefits of an approximation, namely processing speed. In several use cases however, these bigger grammars can perform much better than a medium-sized approximation extracted from just gold-standard data, not only by allowing much better coverage; more training data supports much more aggressive use of internal annotation, making much tighter approximations possible. Tighter approximations model the original grammar better, and may as such perform better. Note, in table 4.5, how WW-33 an approximation with more than three times more annotation only gives a very slight dip in production coverage over WW-10, when using much more training data. For reference we include a grammar extracted with the same 33 annotations on just WeScience.

The numbers presented in this section reflect general measures on the resulting approximations, and how these measures change when varying the level of internal annotation, using tree-rewriting techniques and varying the amount of data the approximation is drawn from. These measures should be independent of a use case, where more “in vivo” experiments need to be performed. However, these measures can still serve as a starting point on what kind of approximation and what level of refinement that should be selected for the task at hand.

CHAPTER 5

Standalone parsing with approximations

A case study

Finding alternative ways to process unification based grammars is a trending topic. As discussed in section 3.1, unification grammars allow a high degree of linguistic expressivity. The added capabilities of the formalism come at a cost of higher processing time. Context-free approximations can be useful in several scenarios where it is working in unison with a full-blown UBG parser. Another, perhaps more ambitious scenario is to let the approximation be used as a grammar in a separate regular context-free parser. While not relying exhaustively on context-free techniques, Ytrestøl (2011b) and Ninomiya et al. (2011) both explore processing of unification based grammars with shift-reduce parsers. The shift-reduce parser is augmented with an oracle that controls the actions the parser performs, and relatively coarse context-free approximations are in some configurations used to aid the oracle.

In this chapter we work on standalone parsing with the grammars obtained from the approximation techniques described in the previous chapters. In this scenario, the task of exploring the search space and applying or instantiating grammar rules is shifted entirely onto the PCFG parser. There are two main advantages to this: First, processing speed should be expected to greatly increase. Secondly, the robustness of the grammar and parsing system as a whole may increase. Precision grammars typically have two failure modes, the first one being technical issues arising from limited time and memory resources. However, precision grammars are often engineered with a high focus on not licensing ungrammatical structures, such as minor errors in agreement, or more serious errors like mismatching subcategorization frames. This can entail that using precision grammars on noisy or unedited text can lead to coverage losses. Context-free approximations may not only be used to increase processing speed, but by not annotating all constraining features, a more robust grammar can be obtained. While perhaps not linguistically adequate, by using this approach some meaningful interpretation of these sentences might be salvaged.

5.1 Parsing and evaluation tools

In the following experiments, we use BitPar (Schmid, 2004) to parse the sentences in the same test corpus as the previous chapter, WeScience Section 13. BitPar is a highly optimized PCFG parser. BitPar uses a classical CKY-style parsing scheme that is augmented to handle unary rule applications, resulting in a parser that handles both unary and binary rules. Grammar binarization is not neccesary, as the LinGO ERG is already maximally binary. BitPar uses bit-vector representations of possible grammar symbols internally in order to parallelize the inference of new possible edges. Here machine instructions, for instance binary-and on two machine words, are used to compute the intersection of two cells in the parsing chart. On 64-bit machines, this means that the parser could compute the intersection of two sets in one step, but only if the number of possible members in the set is 64 or less. If the universe of categories is bigger, more machine words must be allocated. The crucial point however; is that these bitfields are *ordered*. For example, even though only three members were present in a set, two 64-bit words would be necessary to encode this fact if the universe was bigger than 64 (and smaller than 128). Generally speaking, BitPar is a suitable parser for highly ambiguous probabilistic context-free grammars.

The parse trees returned by BitPar need to be compared in a meaningful fashion to the gold standard trees in WS13. Several evaluation metrics have been proposed for measuring the quality of a parser and a grammar. The two main areas one would usually want to quantify are the parsing time and the accuracy of the parsing results. Getting a meaningful number on parsing accuracy is a difficult task. The strictest metric is exact match (EX), as used in (Zhang et al., 2007) This quantifies how often the tree returned from the parser matches the gold standard *exactly*. We follow standard procedure, like Zhang and Krieger (2011), and decouple this measure from tagging accuracy (elaborated on below); the pre-terminal nodes are left out of the comparison. However, EX will count a tree with just one minor error as just as wrong as a completely meaningless tree. ParsEval scores (Black et al., 1991) are another classic metric. Here the trees are split into a set of labeled brackets, each describing the span and the category of a node in the tree. By splitting both the resulting parse tree and the gold standard tree into brackets, one can measure the overlap of these sets. Precision, $\frac{|B_{parse} \cap B_{gold}|}{|B_{parse}|}$ measures how accurate the parser labels tree nodes, while recall $\frac{|B_{parse} \cap B_{gold}|}{|B_{gold}|}$ measures how accurately the parser finds the correct tree nodes. It is often possible to optimize a system to favor one of these measures over the other. This can be very useful depending on the task. In parsing however we are usually more interested in a summarized metric. The F_1 score is the harmonic mean $\frac{2PR}{P+R}$ of precision and recall. The harmonic mean is chosen over the arithmetic mean to penalize eventual degenerate cases where one of the two component metrics is maximized at the penalty of the other. Creating all possible spans gives a perfect recall, and not creating any gives a perfect precision.

Before these measures are evaluated, the parse trees are normalized back to their original form, apart from rule applications deleted by lexical generalization. Lexical collapsing is undone, and feature annotation is removed. Hence, the resulting parse tree is now annotated by the smallest reconstructable form, namely the original grammar rule that "sponsored" the tree node. Recall, from section 3.3 the resulting tree now specifies *exactly* one feature structure iff. the grammar licenses it.

The F_1 score might be more informative than EX, in the way that it can give partial credit to partially correct parse trees. However, when interpreting the parse trees, some nodes are more critical than others. This is not reflected in the F_1 score. Dridan and Oepen (2011) discuss problems with syntactic evaluation metrics in general and propose an evaluation scheme more directly connected to interpretation models of a sentence. Some of the deleted constraints in our grammars are purely syntactic and do not posit constraints on the semantic interpretation of sentences. F_1 scores on syntactic bracketing can penalize this. For example, some rule schemata are codified into two phrase structure rules, typically rules pertaining to coordination or adjungation, where several "flavours" of punctuation might be present. These rules in the ERG are duplicated in both a "formal" and a "non-formal variant". This distinction might be interesting syntactically, but the main interpretation of the sentence is not affected. Furthermore, context-free grammars cannot easily separate these rules. Punctuation information is described in the ERG at the feature SYNSEM.PUNCT and its substructure. However, annotating with these feature structures would create a very sparse grammar. A semantic evaluation approach might give a better reflection on the usability of the obtained grammars, however we only present ParsEval and EX scores in this work due the unsolved problem of extracting semantics robustly from potentially malformed syntactic analyses.

Another useful metric is the tagging accuracy (TA). This measures how accurate the parser assigns preterminals, or tags, to the input words. Here the gold-standard tag assignment is compared to the tag the parser outputs. One could decompose this tagging into precision and recall, which especially lends itself if one wanted to further decompose the accuracy into a score per tag class; but we only report accuracy, which is a "flat" measure; the fraction of correctly assigned tags over the total. As discussed in section 3.1, resolving lexical ambiguity can greatly speed up the processing of unification-based grammars. By taking the preterminal yield of the highest ranking parse tree, the input words can be tagged. To avoid potential errors, one could take the yield of the n best parse trees. In that case, some lexical ambiguity might still be present, but the likelihood of having the correct tags still present increases.

5.2 Experiment setup

As in the previous chapter, WeScience 13 (WS13) is used as a held-out corpus. To get a starting point for later comparisons we present how PET, the main high-performance DELPH-IN parser, itself performs on WS13. We also detail how sentence lengths distributed in the corpus. Recall that WeScience is a corpus constructed from Wikipedia articles. We can see this fact being reflected in the high amount of very short sentences below five words, which correspond to article titles and subheadings. This means that the average sentence length is not a very informative number taken by itself. Table 5.1 details precision, recall and exact match scores on the trees yielded by PET. Here, only the best-scoring tree, as judged by a parse-selection model, trained on WeScience sections 1 to 12, is compared to the gold standard. The overall aim of this experiment is not to improve parse selection. PET yields an exact match of 46.6%[1,2], and it is unlikely that context-free approximations will be able to improve upon this. However, the average parsing time of 3.6 seconds per sentence is the main target for improvement in these experiments.

$\mu = 3.6$ seconds ; $\Sigma = 2846$ seconds

L>	N	P%	R%	EX %	TA%
5	216	94	94	90	97
10	78	87	87	64	96
15	122	87	86	45	96
20	130	86	86	43	97
25	104	86	86	26	95
30	78	81	82	17	95
35	41	81	81	9	97
40+	21	79	80	0	95
Total	785	87	87	46.6	96

Table 5.1: Length (L>), Total items (N), precision (P), recall (R), exact-match (EX) and tagging accuracy (TA) for WeScience 13 parsed and disambiguated by PET. Also included is the mean parsing time per sentence (μ) and the total parsing time to process the entire test set (Σ).

5.3 Parsing with BitPar

In this first round of experiments, we parse WS13 with BitPar. We use the same grammars as in the experiments in the previous chapter. Table 5.2 shows the results of measuring gold standard trees against the most probable parse tree. Note the big increase in performance with the lexically collapsed grammars. This big increase can be explained theoretically as follows: First note that here only the most probable parse tree is examined, and that lexical collapsing works only on unary production chains. Given the case that a category A is present in the most probable parse tree. In the case where A can be derived (transitively) from an input word w, the most probable way of doing so will always be part of the most probable complete parse tree. If A can be derived through $A \rightarrow B \rightarrow w$ but also through $A \rightarrow B \rightarrow C \rightarrow w$ only the first unary chain will be selected. Intuitively, the PCFG is not "forced" to apply all the necessary morphological processing rules. In the case where there exist several unary chains from a word resulting in a given category, only the most probable one would be selected. This fact is supported by the large increase in recall, where originally several morphological rules are missing from the best parse trees. When lexical collapsing is applied, the input words now project their *entire* morphological derivation.

None of these grammars can rival PET in accuracy, but processing time is more than two orders of magnitude faster. PET parses WS13 in about 45 minutes, while the fastest grammars here spend less than half a minute processing the same corpus. At an average parsing time of about 30 milliseconds, these grammars could be used in situations where real-time responses are required, such as interactive dialogue environments, or real-time refinement or analysis

[1]However, this comparison is (sadly) not strictly one-to-one to our experiments as we use gold-standard tokenization, which was not performed in the experiment this measure was taken from.

[2]Zhang and Krieger (2011) report an exact match score on 44% by PET, however here the parse selection model was trained on WS1-11, i.e. section 12 being held out. For reference, the parameters used in the parse-selection model was three levels of grandparenting and a feature frequency cutoff of 5.

A	C%	P%	R%	F1	EX%	TA%	μ	Σ
PET	100	84	84	84	46	96	3.6k	2846
0	100	65	59	62	11	89	385	303
1	100	71	64	68	12	89	115	90
5	99	74	68	71	15	89	76	50
10	97	77	73	75	20	91	42	33
15	95	76	73	75	20	91	41	30
0-LC	98	73	72	72	23	86	128	100
1-LC	98	77	77	77	23	87	36	28
5-LC	91	79	79	79	29	90	25	20
10-LC	86	81	81	81	32	92	24	19
15-LC	79	81	82	81	36	92	28	22

Table 5.2: One-best PCFG parse of WS13, for several grammars as discussed in the previous chapter. The EX is given on *covered sentences*. μ is reported in milliseconds, Σ in seconds.

of search results. Lexical collapsing radically improves accuracy, but at the cost of some coverage loss. Note also how the lexical collapsed variants, even though these grammars are bigger (see tables 4.4 and 4.2) ,they can still be processed much faster. This is because the ambiguity rates in these grammars are lower, and therefore the parser creates much “thinner” parse forests. The words in the input “activate” lesser parts of the grammar, and in purely bottom up approaches this results in less work for the parser.

We report exact match on the covered sentences only. Arguabely, this gives a somewhat more convoluted picture for “upstream” users of a parser system, since now the exact match ratio has to be multiplied with the coverage to give a measure on how often the system *delivers* the correct result. However, in these approxmations the parsing time is relatively cheap. One could imagine situations where input was parsed with the relatively accuracte grammars with only moderate coverage. If this succedes we now have a comparatively high probability on an exact match. If the parser did not find any analysis, one could fall back to a less annotated grammar. Now one might still get a result, but with a comparatively lower probability of an exact match.

As noted in the previous chapter, the coverage of the more annotated and lexical collapsed variants is somewhat disenchanting. By adding more training data the coverage increases, but the size of the grammar increases as well. In table 5.3 we report the accuracy and performance of the two bigger grammars presented in the previous chapter. The biggest grammar, WW-33, has almost full coverage, and a respectable accuracy of 37% exact matches. While these are perhaps more realistic candidates for stand-alone parsing, the original goal of low processing time is not met with satisfaction. The current performance of BitPar on WS-33, around half a second processing time per sentence, leaves something to be desired. While this configuration is still much faster than PET, the accuracy and coverage loss compared to PET might not justify the increased parsing speed of the approximation. Furthermore, in areas where processing time is tightly constrained, 500 milliseconds might still not be fast enough to be useful.

A	C%	P%	R%	F1	EX%	TA%	μ	Σ
PET	100	87	87	87	46	96	3.6k	2846
WS-10	86	81	81	81	32	92	24	19
WS-33	46	87	88	88	61	96	47	37
WW-10	97	83	84	84	32	92	329	259
WW-33	97	83	83	83	37	92	571	449

Table 5.3: One-best PCFG parse of WS13 with the larger grammars trained on a small subset of WikiWoods. Grammars with 10 and 33 annotations trained only on WeScience is included for reference. Both lexical collapsing and lexical generalization is performed. The reason for the "stellar" accuracy of WS-33 is the low coverage, mostly the short sentences, and thus easier to parse accurately, were covered.

5.4 Investigating binary parallelization

BitPar is suitable for highly ambiguous grammars. However, the parallelization it performs can require very long bitfields. If these are very sparse, the potential profit in speed one could gain by parallelization is lost, by having to have enough machine words to be able to codify the entire universe of possible symbols. WW-33 is not highly ambiguous, but does contain many constraints inherited by the high level of annotation. It therefore follows that WW-33 produces much thinner parse forests than the less annotated grammars.

We therefore implemented a regular augmented CKY parser without bitfields in Common Lisp, building on the codebase implemented in the previous chapter. Figures 5.1 and 5.2 present the basic algorithm in pseudo code. This parser, which certainly is not optimized to the level of BitPar, does still perform better on some of the grammars we investigate here. Table 5.4 shows a comparison of the two parsing systems where WW-33 is used as the grammar, and the sentences in WeScience 13 are parsed. The comparison here is between, two complete parsing systems, and not just binary parallelization. First, BitPar precomputes all possible unary production chains, and memoizes these. This means that doing the unary-rule expansion (see algorithm 5.2) is much faster for BitPar. Secondly, in the configuration used here, BitPar only produces the 1-best Viterbi parse. However our parser perform so-called full forest construction, which means that *all* possible parse trees might be extracted for a given input. This requires much more computation to keep track of the complete parse forest. Code to perform so-called "selective unpacking" (Carroll & Oepen, 2005; Zhang, Oepen, & Carroll, 2010) has been successfully adopted from the LKB, yielding efficient n-best output if so desired.

While the configurations used here are not valid as a direct comparsion between two equal parsers with and without binary parallelization, it still shows that parallelization is not an optimization that is suitable for all kinds of grammars, since our less optimized implementation outperforms BitPar by a large margin on WW-33.

We also tried to parse WS13 with the WW-10 grammar as discussed above, but this turned out to be prohibitively slow. WW-10 is a much more ambiguous grammar, and it might not be feasible in practice to perform full forest construction without any pruning. For example, one of the longer sentences in WS13, 35 words long, creates a massive 4.5 million edges. In both parsers, no pruning was performed. When using pruning, low edges with a relatively probability compared to other edges in the same span are removed from further processing. This can potentially lead to drastic improvements in parsing time, but the possibility of so called "search-errors" is now introduced. A search-error happens in situations where the parser does not find the most probable parse tree according to its own grammar, caused by one or more constituent edges having been pruned. With conservative thresholds however, search-errors occur relatively rarely.

A	C%	P%	R%	F1	EX%	TA%	μ	Σ
PET	100	87	87	87	46	96	3.6k	2846
BitPar-33	97	83	83	83	37	92	571	449
CKY+-33	97	83	83	83	37	92	142	112

Table 5.4: PCFG parsing times with our parser.

5.5 A meta-comparison of parsers

In the recent time, several stand-alone parsing systems have been proposed and tested with the LinGO ERG using the 13th section of WeScience as test data, all with gold standard tokenization. It is therefore tempting to compare aspects of the performance and accuracy of these different approaches, in such as relevant information is available. However, as these systems differ minutely in assumptions on input, this comparison should only be interpreted as a rough sketch of how a true head-to-head comparison might be, and the comparisoon is too coarse to give meaningful absolute numbers on the ranking of the systems compared here.

In table 5.5 we compare the published scores of CuteForce (Ytrestøl, 2011a) an oracle guided shift-reduce parser. Here we report the scores where CuteForce is using supertagged input, with a tag accuracy of 95%, and is working in so-called "unrestricted mode", meaning that neither CFG nor UB filtering is applied to the parser actions. Ytrestøl (2011a) report that 51% of the analyses returned by their parser are *valid*, that is, not conflicting with any of the original constraints in the grammar. In this configuration, CuteForce is trained on about 150000 derivations from WeScience and WikiWoods, roughly half of the training data used in our grammar. However, and crucially, the input is first tagged with a supertagger with an accuracy of 95%. This tagger is trained on 6.8 million sentences from the WikiWoods, and it is reasonable to assume that using such high quality supertags would also increase the performance in our system, where supertagging is not performed. One would expect that resolving the lexical ambiguity would give rise to much lower processing times. Furthermore our system perform with a tag accuracy of around 92% in almost all configurations. It would be interesting to see if the accuracy could rise to better values when high-quality supertagging is performed. The average parsing time of CuteForce and our parser have both been benchmarked on the same hardware.

We also include one grammar from Zhang and Krieger (2011), the "GP 2, FP 2" grammar trained on "WW00". Here a PCFG is constructed with two levels of grandparenting and two internal feature annotations; SYNSEM.LOCAL.CAT.HEAD and SYNSEM.LOCAL.CONJ. These features are also included our grammars, in fact they are also our two "top-ranked" candidates for internal annotation. This grammar was trained on "WW00" which is every section

```
Algorithm: Augmented CKY
Data: Input tokens I
Data: Grammar G
Prepare a chart structure to hold resulting (sub)-trees.
Each cell $\langle x, y \rangle$ correspond to a tree spanning the input from token number $x$ to $y$.
chart ← MakeArray((Length(I)+1)$^2$);
First, all productions $A \rightarrow w$ where $w$ corresponds to a token in the input are inserted into the chart:
foreach Word w in I, index from 0 do
    chart[index,index+1] ← $\{A\ :\ A \rightarrow\ w \in G\}$;
end
Next, all unary rules derivable from the symbols in each 1 element-span must be found:
foreach index i from 0 to Length(I) do
    chart[index,index+1] ← chart[index,index+1] ∪ UnaryRule-Expansion(chart[index,index+1]);
end
Now, the main parsing loop can be done. Here the algorithm fills in all possible trees in ever increasing spans:
for l from 1 to Length(I) - 1 do
    for i from 0 to Length(I) - L + 1 do
        for j from 1 to l do
            agenda ← ∅;
            left-cell ← chart[i, i + j];
            right-cell ← chart[(i + j),(i + l + 1)];
            Find all possible ways of combining symbols from the left and right cells:
            foreach B,C in $\{\langle B, C\rangle\ :\ B \in$ left-cell $\wedge\ C \in$ right-cell$\}$ do
                agenda ← agenda ∪ $\{A : A \rightarrow B\ C \in G\}$;
                chart[i, i + l + 1] ← UnaryRule-Expansion(agenda);
            end
        end
    end
end
The content of the cell spanning all the input is now the roots of all the parse trees that can be found with the input I. However, only the trees rooted in the start symbol are valid.
return $\{S : S \in$ chart[0,Length(I)+1] $\wedge\ S \in$ start-symbol(G)$\}$;
```

Figure 5.1: Augmented CKY-parsing. For brevity we have not detailed how locally equivalent analyses are packed. The key point in Augmented CKY is that the unary-rule expansion must be performed at each possible span; after all binary trees rooted in the span has been found, but before larger spans are explored.

```
**Algorithm:** Unary rule expansion
Data: Agenda
Data: Grammar G
result ← $\varnothing$;
while $A \leftarrow pop(Agenda) \neq \varnothing$ do
    If this is the first time a span with category A is derived then add all possible unary rules rewriting to
    A in the agenda
    if $A \notin result$ then
        result ← A ∪ result;
        foreach B in $\{B : B \rightarrow A \in G\}$ do
            Agenda ← B ∪ Agenda;
        end
    else
        Otherwise, the new derivation is packed into another derivation.
    end
end
return result;
```

Figure 5.2: Unary rule expansion. The algorithm proceeds to find the fixpoint of possible derivations given a grammar and an initial content, here provided by an "agenda" of tasks to perform.

of WikiWoods with "00" as a suffix. This amounts to about 480000 sentences, which is comparable to our selection of WikiWoods of 433870 sentences. Sadly Zhang and Krieger do not report processing times. Zhang and Krieger also report on bigger experiments that perform better in both F1 and EX, but "GP 2, FP 2" is the most comparable, in the amount of rules and training data, to our WW-33. While this comparison is not strictly head-to-head, as the corpora are different, and Zhang and Krieger treat tokenization different from our system, it shows that very high levels of internal annotation may perform at the same level as classical grandparenting techniques.

A	C%	P%	R%	F1	EX%	TA%	μ	Σ
PET	100	87	87	87	46	96	3.6k	2846
CKY+-33	97	83	83	83	37	92	142	112
CuteForce	99	-	-	82	36	95	15	-
Zhang and Krieger (2011)	-	80	79	80	32	93	-	-

Table 5.5: The performance of different parsing systems and grammars on WS13.

5.6 Outlook and Reflections

This case study shows that high-quality context free approximations can be used as stand-alone parsers for unification based precision grammars with a good degree of practicality. Parsing time can be sped up by over two orders of magnitude, but at the cost of a moderate dip in accuracy. However, the metrics used here were overly syntacto-centric. The LinGO ERG operates with a very fine level of detail in syntactic constructions. To permit practical approximations, many of these details need to be left out, and therefore the approximation based parsers can be somewhat penalized for giving a wrong syntactic analysis, while still presenting a correct semantic analysis. It is therefore our opinion that a semantic evaluation metric can be more telling on the performance of these stand-alone systems. Arguably, a detailed syntactic account of many processes is not of particular high interest when approximation-based parsers are used, and if the syntax should take center stage the original grammars and parsing systems are much more proficient performers.

It is reasonable to assume that the accuracy of the grammars presented in this section could be improved in several ways. First, the parse selection component of PET typically does not make use of the internal structure of nodes, and works directly at the top level, which is also what is available in the smallest reconstructable form. It is therefore possible to apply the original parse selection model to the parse forests that our parser constructs. It is reasonable to assume that this gives an increase in accuracy. However, it is hard to predict how this setup would compete with PET. The parse forest of the approximation would be likely to contain both grammatical and ungrammatical derivations, making the number of potential candidates the parse selection component must rank against each other higher. One could also use classical reranking techniques where the top n, typically in the low hundred, most probable parse trees are inspected by a separate model.

Another approach which might lead to accuracy gains is reconstruction. In this setup the complete derivations are reconstructed, the derivation is "replayed" in the original parser, in descending order of probability, and the first *valid* derivation, that is, a derivation not in conflict with the original grammar, is selected as correct. This is attractive from several angles. First, it is reasonable to assume that accuracy may improve. Secondly, semantics can easily be extracted, as the resulting derivation is well formed. Third, all the information that was lost to facilitate approximation is regained.

However, derivations that fail reconstruction might not be worthless. Especially the more annotated approximations, like WW-33, encode many of the most important HPSG principles. For example, the head value of complement, subject and specifier arguments are annotated, in addition to several features pertaining to adjungative constructions. This means that many of the constraints in the original UBG are reflected to a high degree in fine-grained approximations, and it is not unreasonable to assume that these "pseudo-derivations" might still be usable in many situations, especially where parse results are primarily used to mark phrasal boundaries or word dependencies, which could be an interesting information source in tasks like named-entity recognition, or negation scope resolution.

The meta-comparison in section 5.5 suggests that high levels of internal annotation might perform at the same level in accuracy as comparatively high levels of grandparenting. While Zhang and Krieger (2011) do not report processing times, it is reasonable to assume that the average ambiguity rate in "unparented" grammars is lower, and processing could be performed faster. Ytrestøl (2011a) reports very impressive processing times and equally impressive accuracy. In their parsing system, lexical ambiguity in the input is first resolved by a high-quality supertagger with an accuracy of about 95%, while our system starts from lexical surfaces directly, which creates denser parse-forests. It is not unreasonable to assume that processing time could be sped up significantly by incorporating supertagging. More crucially, our tagger only obtains a tag-accuracy at around a level of 92% in most configurations. One could then expect that the accuracy on syntactic constructions (F1,EX) could rise when the parser is using supertags with an accuracy of 95% as input.

Lattice-based input is another interesting notion. Here a separate morphological component constructs a lattice of all possible analyses of the input. This enables another training scheme, where the approximation disregards much of the morphological machinery and instead only works on feature structures that could take part in syntactical constructions. These could be annotated in the regular fashion. In this scheme the annotated "top" nodes of the lexical projections become the terminal symbols of the grammar. This obviates the difficulties arising from the possible infinite lexicon, and one could assume that this class of purely syntactic approximations might achieve better coverage.

Chapter 6
Conclusion

Where you've been is good and gone
All you keep is the getting there

Townes van Zandt (1944-1997)

In this work we have concentrated on context-free approximations of unification-based grammars. Such approximation has been proposed as one of the approaches to increase the practicality of deep language processing, and with the availability of very large corpora of automatically disambiguated deep grammar parse results, coupled with a smaller, but high quality manually disambiguated treebank, probabilistic context-free approximations have received renewed interest in the last few years.

In this work, such approximations are shown to be a very promising approach to the parsing of deep grammars, chiefly by increasing parsing speed. One of the approximations presented in this work achieved a substantial speed up over a high-performance UBG-parser, with an average parsing time of 147 milliseconds per sentence, where the original parser had an average of 3.6 seconds parsing time per sentence in the same corpus.

This speedup however, does come at a cost in accuracy; on the exact-match metric the original parser performs at 46%, while the aforementioned approximation obtains 37% exact matches. More advanced parse selection models could be applied to increase the accuracy of the approximation. This approximation yields only a very slight dip in coverage over the original, at 97% of the test corpus. It is possible to create more accurate approximations, but at the expense of coverage. Accuracy and coverage can be traded off against each other to suit the needs of the task at hand.

In chapter 3, several use scenarios where a high-quality context-free approximation might be useful were reviewed, and a theoretical perspective on grammar approximation was presented. Next, we discussed main approximation techniques, with a special focus on obtaining *reconstructable* approximations. Derivations in these grammars can be "replayed" in the original UBG; a complete approximate derivation specify one unique derivation in the original grammar. This means that the information that was lost in order to facilitate the approximation could be obtained at a later point if so desired.

We have implemented a flexible system for obtaining approximations in the framework of DELPH-IN tools. This system can parallelize grammar extraction to facilitate large-scale approximation experiments. Chapter 4 detailed this system and several practical considerations were discussed, with a particular focus on sound versus unsound approximation schemes and feature selection.

To resolve an "impedance mismatch" in the tokenization assumptions by the two formalisms we showed how this could be resolved with a process called *lexical generalization*, and how this procedure do not conflict with unique reconstructability if the original grammar satisfies weak conditions pertaining to tokenization.

In chapters 4 and 5 we showed theoretically and empirically how *lexical collapsing* is a necessary step to obtain high quality approximations. We also detailed static and dynamic measurements on the obtained approximations in chapter 4, with a particular focus on how one approximation overlap with another made with the same configuration on a held-out data-set, and how these measurements vary with the number of features used in internal annotation, and the use of lexical collapsing and generalization. Finally we presented approximations created with much larger amounts of training data.

Several of the grammars obtained in chapter 4 were examined as candidates for stand-alone parsing in chapter 5, where we reported parsing time and accuracy scores on various metrics. In a successful attempt to increase efficiency in some configurations, a CKY-style parser was implemented, giving a substantial performance boost over a highly optimized off-the-shelf parser. The most promising approximation was compared to other recent approaches by Zhang and Krieger (2011) and Ytrestøl (2011a). Possible approaches to improve both speed and parsing accuracy were discussed.

6.1 Future work

On the experimental side, we would like to reiterate that syntacto-centric evaluations like exact-match and ParsEval may not be immediately reflecting on the accuracy of the results obtained from context-free approximations. Precision-grammars typically contain a very fine-grained detail of linguistic description, where syntactic structures and processes take center stage. While syntacto-centric measures may give a good reflection on how various deep parsers and parse-selection models compare against each other, they might not be so suitable when cross-comparing deep and "pseudo-deep" parsers, such as PCFG approximations. The fine level of detail a deep grammar uses in syntactic description must necessarily be reduced in order to facilitate an approximation.

Using semantic evaluation techniques, as proposed in Dridan and Oepen (2011), could potentially give a better reflection on the true usability of context-free approximations in stand-alone parsing. Arguably, the finer points of syntax are not in center stage when context-free approximations are employed for stand-alone parsing. It is therefore more interesting to quantify how accurate the parser can recreate the semantic interpretation of a sentence.

Context-free approximations can be used, as Zhang and Krieger (2011) discuss in detail, to increase the robustness of deep grammar parsing systems. In our work, processing time has been the chief target, but deep grammars can also fail to analyze input in other cases than not being within time or memory limits; either through undergeneration in the grammar, or because of grammatical errors in the input itself. While having a strong notion of grammaticality is very useful in many cases, especially in natural language generation, it can be detrimental if unedited and noisy text were to be analyzed. In such situations, a context-free approximation that is more permissive than the original grammar could be used to parse such input. While the resulting so-called "pseudo-derivations" would not be valid derivations in the original grammar, they may still contain useful information. Extracting semantics robustly from such derivations is an ongoing research topic.

Lexical collapsing is one of the more important techniques to apply in order to obtain high quality approximations. However, even with lexical generalization, this technique can quickly introduce sparsity in the grammar. Modern UBGs can license an infinitely sized lexicon through repeated use of derivational morphology. Even though unseen lexical and morphological projections become rarer with more training data, they are still one of the "Achilles'-heels" of the system as a whole. A possible way around this is to not include the morphological component of the original grammar into the approximation, but instead use annotated "top" nodes of the lexical projections as terminal symbols in the approximation.

Many candidate feature paths, especially those describing selectional constraints on complements, are not immediately useful for direct annotation, as they rely too heavily on the type-hierarchy of the original grammar. One way to include these features and still obtain a practical grammar is to actively use the type hierarchy in the approximation procedure. By observing which types the value of the features at these paths usually take. If the variance in the observed types was too high to sufficiently model the type hierarchy, one could replace the more specific types with one of their more general types, thus creating a less fine-grained type hierarchy which might be more suitable for direct annotation. This is somewhat similar to aggregating the obtained context-free rules under *rule subsumption* instead of equivalence, as done in Krieger (2007). However, aggregating all rules under subsumption might create an approximation that is too permissive. Investigating various forms of rule aggregation, either directly under equivalence or with several degrees of subsumption is an interesting direction of future work.

The LinGO ERG, as of version 1010, operates with about 200 rule schemata. In this work, we have used the same annotation vector for all feature structures, but it might be interesting to differentiate the annotations based on what kind of rule schema that was used to construct each feature structure. For example, if one ordered the feature paths by unification failure rate, one could imagine that complement constructions and relative constructions would have different orderings of these paths. One problematic aspect of this notion however, is that all the necessary information need to be present in the symbols of the approximation. If one context-free rule is annotated with a feature path on the LHS that is not present in its RHS, the rule "adds" information in an underhand way.

Unification-based grammars are typically monotonic, and when only valid derivations are used as training data in the approximations used in this work, the resulting grammars are also modeling this monotonicity. This is facilitated by holding the annotation vector constant in all cases. In these grammars, malformed derivations can be constructed only when *not enough* of the constraints of the original grammar were modeled. However, if the annotation vector is not constant, “new” feature paths in the left-hand side of a rule could now add information that was not present in its right-hand side, nor in the structure of the original rule itself. This fact, in addition to dealing with the needed extra effort of specifying several annotation vectors, are one of the problems future studies on varying annotation vectors could address.

Another potentially interesting use of robust probabilistic context-free approximations is language modeling. In many tasks with a component of language generation, such as statistical machine translation, a language model is used to rank candidate realizations against each other. N-gram models are typically used in this task. While these have many desirable properties, such as computational efficiency and ease of implementation, and practically limitless amounts of training data available to them, they are not always suitable in modeling long-range dependencies, and have no direct concept of syntax. PCFG language models obtained from the approximation of deep grammars, especially when very large corpora of parse results are available could be an interesting alternative in the realization ranking task.

APPENDIX **A**

Feature paths for internal annotation

The features used for internal annotation in this work is taken in order from the following list. These path are exact, and not simplified as some of the paths in earlier figures may appear. These paths are taken from the file `qc.tdl` as it appears in the LinGO ERG as of version 1010, but are not necessarily listed in the order the paths occur in the file. Note also, that some paths subsume others and are left out in favor of the more specific one. This applies chiefly to valency.

```
synsem local cat head
synsem local conj
synsem local cat head aux
synsem local agr png pn
synsem local cat val spr first local cat head
synsem local cat val subj first local cat head
synsem local cat val comps first local cat head
synsem local cat head mod first local cat head
synsem local cat val comps first opt
synsem lex
synsem nonloc slash list first cat head
synsem local cat head tam mood
synsem local cat mc
inflectd
synsem local cat head inv
synsem nonloc slash list first cat head mod first local cat head
synsem local cat head vform
synsem modifd lperiph
synsem modifd rperiph
synsem modifd
synsem --sind
synsem local cat val subj first --sind
synsem local cat val subj first opt
synsem local cat val spr first opt
synsem local agr div
synsem --sind --tpc
synsem nonloc slash list first agr png pn
synsem local cat head mod first --sind
synsem local cat head tam tense
synsem local cat posthd
synsem local cat head prd
synsem nonloc que
synsem local cat hc-lex
```

Table A.1: Listing of features used for internal annotation

APPENDIX **B**

The WikiWoods subset

Some of the larger experiments discussed in this work draw from a section of the WikiWoods corpus, which consists of parsed and automatically disambiguated Wikipedia articles. In this work, these were parsed with ERG 1010. We use section 00110 in addition to every tenth section up to section 01290. To clear up any potential ambiguity around data selection, a complete listing follows:

00110	00370	00630
00120	00380	00640
00130	00390	00650
00140	00400	00660
00150	00410	00670
00160	00420	00680
00170	00430	00690
00180	00440	00700
00190	00450	00710
00200	00460	00720
00210	00470	00730
00220	00480	00740
00230	00490	00750
00240	00500	00760
00250	00510	00770
00260	00520	00780
00270	00530	00790
00280	00540	00800
00290	00550	00810
00300	00560	00820
00310	00570	00830
00320	00580	00840
00330	00590	00850
00340	00600	00860
00350	00610	00870
00360	00620	00880

00890	01030	01170
00900	01040	01180
00910	01050	01190
00920	01060	01200
00930	01070	01210
00940	01080	01220
00950	01090	01230
00960	01100	01240
00970	01110	01250
00980	01120	01260
00990	01130	01270
01000	01140	01280
01010	01150	01290
01020	01160	

Bibliography

Bangalore, S., & Joshi, A. K. (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, *25*(2), 237–265.

Black, E., Abney, S. P., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., et al. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings DARPA speech and natural language workshop* (pp. 306–311). Pacific Grove, CA: Morgan Kaufmann.

Bouma, G., & van Noord, G. (1993). Head-driven parsing for lexicalist grammars. Experimental results. In *Proceedings of the 6th Conference of the European Chapter of the ACL* (pp. 71 – 80). Utrecht, The Netherlands.

Callmeier, U. (2000). PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, *6 (1) (Special Issue on Efficient Processing with HPSG)*, 99 – 108.

Cancedda, N., & Samuelsson, C. (2000, April). Experiments with corpus-based lfg specialization. In *Proceedings of the sixth conference on applied natural language processing* (pp. 204–209). Seattle, Washington, USA: Association for Computational Linguistics.

Carpenter, B. (1992). *The logic of typed feature structures.* Cambridge, UK: Cambridge University Press.

Carroll, J. (1993). *Practical Unification-based Parsing of Natural Language*. Doctoral dissertation, University of Cambridge.

Carroll, J., & Oepen, S. (2005). High efficiency realization for a wide-coverage unification grammar. In *Natural language processing - IJCNLP 2005, second international joint conference* (p. 165-176).

Chomsky, N. (1956). Threee models for the description of language. *IRE Transactions on Information Theory*, *2*, 113-124.

Copestake, A. (1992). The ACQUILEX LKB. Representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing* (pp. 88 – 96). Trento, Italy.

Copestake, A. (2002). *Implementing typed feature structure grammars*. Stanford, CA: Center for the Study of Language and Information.

Copestake, A., Flickinger, D., Sag, I. A., & Pollard, C. (1999). *Minimal Recursion Semantics. An introduction.* In preparation, CSLI Stanford, Stanford, CA.

Cramer, B., & Zhang, Y. (2010). Constraining robust constructions for broad-coverage parsing with precision grammars. In C.-R. Huang & D. Jurafsky (Eds.), *23rd international conference on computational linguistics* (p. 223-231). Tsinghua University Press.

Dridan, R. (2009). *Using lexical statistics to improve HPSG parsing*. Doctoral dissertation, Saarland University.

Dridan, R., & Oepen, S. (2011, October). Parser evaluation using elementary dependency matching. In *Proceedings of the 12th international conference on parsing technologies* (pp. 225–230). Dublin, Ireland: Association for Computational Linguistics.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering*, *6 (1)*, 15 – 28.

Flickinger, D., Oepen, S., & Ytrestøl, G. (2010). Wikiwoods: Syntacto-semantic annotation for english wikipedia. In N. Calzolari et al. (Eds.), *Proceedings of the 7th International Conference on Language Resources and Evaluation.* European Language Resources Association.

Gazdar, G., Klein, E., Pullum, G., & Sag, I. (1985). *Generalized phrase structure grammar* (Vol. 10) (No. 3). Harvard University Press.

Goldstein, S. (1988). *Using an active chart parser to convert any context-free grammar to backus-naur form.* Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation* (1st ed.). Addison-Wesley.

Kaplan, R. M., & Bresnan, J. (1995). Lexical-functional grammar : A formal system for grammatical representation. *The Mental Representation of Grammatical Relations*, *3*(3), 173–281.

Kasami, T. (1965). *An efficient recognition and syntax algorithm for context-free languages* (Tech. Rep. No. AFCLR-65-758). Air Force Cambridge Research Laboratory, Bedford, MA.

Kay, M. (1989). Head-driven parsing. In *Proceedings of the 1st International Workshop on Parsing Technologies* (pp. 52 – 62). Pittsburgh, PA.

Kiefer, B., & Krieger, H.-U. (2000). A context-free approximation of Head-driven Phrase Structure Grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies* (pp. 135 – 146). Trento, Italy.

Kiefer, B., & Krieger, H.-U. (2002). A context-free approximation of head-driven phrase structure grammar. In S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit (Eds.), *Collaborative language engineering. a case study in efficient grammar-based processing.* Stanford: CSLI Publications.

Kiefer, B., Krieger, H.-U., Carroll, J., & Malouf, R. (1999). A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics* (pp. 473 – 480). College Park, MD.

Kiefer, B., Krieger, H.-U., & Prescher, D. (2002). A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. In *Proceedings of the 19th international conference on computational linguistics - volume 1* (pp. 1–7). Morristown, NJ, USA: Association for Computational Linguistics.

Krieger, H.-U. (2004). A corpus-driven context-free approximation of Head-Driven Phrase Structure Grammar. In G. Paliouras & Y. Sakakibara (Eds.), *Icgi* (Vol. 3264, p. 199-

210). Springer.

Krieger, H.-U. (2007). From UBGs to CFGs, a practical corpus-driven approach. *Nat. Lang. Eng.*, *13*(4), 317–351.

Krieger, H.-U., & Schäfer, U. (1994). $\mathcal{TDL}$: a type description language for constraint-based grammars. In *Proceedings of the 15th conference on computational linguistics - volume 2* (pp. 893–899). Stroudsburg, PA, USA: Association for Computational Linguistics.

Malouf, R., Carroll, J., & Copestake, A. (2000). Efficient feature structure operations without compilation. *Natural Language Engineering*, *6 (1)*, 29 – 46.

Matsuzaki, T., Miyao, Y., & Tsujii, J. (2007). Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th international joint conference on artifical intelligence* (pp. 1671–1676). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

McCarthy, J. (1983). The Little Thoughts of Thinking Machines. *Psychology Today*, *17*(12).

Moore, R. C. (1999). Using natural language knowledge sources in speech recognition. *Nato Asi Series F Computer And Systems Sciences*, *169*, 304–327.

Neumann, G., & Flickinger, D. (1999). *Learning stochastic lexicalized tree grammars from HPSG* (Tech. Rep.). Saarbrücken: DFKI.

Ninomiya, T., Matsuzaki, T., Shimizu, N., & Nakagawa, H. (2011). Deterministic shift-reduce parsing for unification-based grammars. *Natural Language Engineering*, *17*(3), 331-365.

Oepen, S., & Callmeier, U. (2000). Measure for measure: Parser cross-fertilization. Towards increased component comparability and exchange. In *Proceedings of the 6th International Workshop on Parsing Technologies* (pp. 183 – 194). Trento, Italy.

Oepen, S., & Carroll, J. (2000). Ambiguity packing in constraint-based parsing practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL* (p. 162-169).

Oepen, S., Flickinger, D., Tsujii, J., & Uszkoreit, H. (2002). (S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit, Eds.). Stanford: CSLI Publications.

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language*, *12 (4) (Special Issue on Evaluation)*, 411 – 436.

Oepen, S., Toutanova, K., Shieber, S., Manning, C., Flickinger, D., & Brants, T. (2002). The LinGO Redwoods treebank. Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics.* Taipei, Taiwan.

Petrov, S., Barrett, L., Thibaux, R., & Klein, D. (2006, July). 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* (pp. 433–440). Sydney, Australia: Association for Computational Linguistics.

Pollard, C., & Sag, I. (1994). *Head-driven phrase structure grammar.* Chicago, Illinois: Chicago University Press.

Pollard, C., & Sag, I. A. (1987). *Information-Based Syntax and Semantics. Volume 1: Fundamentals.* Stanford: CSLI Publications.

Rayner, M., Dowding, J., & Hockey, B. A. (2001). A baseline method for compiling typed unification grammars into context free language models. In *Interspeech'01* (p. 729-732).

Sag, I. A., Wasow, T., & Bender, E. (2003). *Syntactic Theory: A Formal Introduction* (2nd ed.). Stanford: Center for the Study of Language and Information.

Schmid, H. (2004). Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of the 20th international conference on computational linguistics.* Stroudsburg, PA, USA: Association for Computational Linguistics.

Shieber, S. M., Uszkoreit, H., Pereira, F., Robinson, J. J., & Tyson, M. (1983). The formalism and implementation of PATR-II. In *Research on interactive acquisition and use of knowledge* (pp. 43–53). Artificial Intelligence Center, SRI International.

Tomabechi, H. (1992). Quasi-destructive graph unification with structure-sharing. In *Proceedings of the 14th International Conference on Computational Linguistics* (pp. 440 – 446). Nantes, France.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, *10*, 189-208.

Ytrestøl, G. (2011a). Cuteforce - deep deterministic HPSG parsing. In *Proceedings of the 12th international conference on parsing technologies* (p. 186-197).

Ytrestøl, G. (2011b). Optimistic backtracking - a backtracking overlay for deterministic incremental parsing. In *Acl (student session)* (p. 58-63). The Association for Computer Linguistics.

Ytrestøl, G., Flickinger, D., & Oepen, S. (2009). Extracting and Annotating Wikipedia subdomains. In *Proceedings of the 8th Workshop on Treebanks and Linguistic Theories.*

Zhang, Y., & Krieger, H.-U. (2011, October). Large-scale corpus-driven pcfg approximation of an HPSG. In *Proceedings of the 12th international conference on parsing technologies* (pp. 198–208). Dublin, Ireland: Association for Computational Linguistics.

Zhang, Y., Matsuzaki, T., & Tsujii, J. (2009). HPSG supertagging: A sequence labeling view. In *Iwpt* (p. 210-213). The Association for Computational Linguistics.

Zhang, Y., Oepen, S., & Carroll, J. (2007). Efficiency in unification-based n-best parsing. In *Proceedings of the 10th international conference on parsing technologies* (pp. 48–59). Stroudsburg, PA, USA: Association for Computational Linguistics.

Zhang, Y., Oepen, S., & Carroll, J. (2010). Eficiency in unification-based n-best parsing. In H. Bunt, P. Merlo, & J. Nivre (Eds.), *Trends in parsing technology: Dependency parsing, domain adaptation, and deep parsing* (p. 223-241). Springer.