Two-Level Probabilistic Grammars for Natural Language Parsing

Gabriel G. Infante-Lopez

Two-Level Pr[illegible] Grammars
for Natura[illegible]

SIKS Dissertation Series 2005-05.
The research reported in this thesis has been carried out under the auspices of SIKS,
the Dutch Research School for Information and Knowledge Systems.

For further information about UvA or SIKS publications, please contact

Informatics Institute
Universiteit van Amsterdam
Kruislaan 403
1098 SJ Amsterdam
The Netherlands
phone: +31-20-525 5359
fax: +31-20-525 2800
e-mail: ii@science.uva.nl
homepage: http://www.science.uva.nl/ii/
homepage: http://www.siks.nl/

Two-Level Probabilistic Grammars for Natural Language Parsing

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.mr. P.F. van der Heijden
ten overstaan van een door het college voor
promoties ingestelde commissie, in het openbaar
te verdedigen in de Aula der Universiteit
op woensdag 6 april 2005, te 12.00 uur

door

Gabriel Gaston Infante Lopez

geboren te San Salvador de Jujuy, Argentina.

Promotores: Prof.dr. M. de Rijke
Prof.dr.ir. R. Scha

The investigations were supported by the Netherlands Organization for Scientific Research (NWO) under project number 220-80-001.

Drawings on the cover by Santiago Infante Lopez.

Cover designed by Lucrecia Resnik.

Printed and bound by Soluciones Gráficas.
http://www.solucionesgraficas.com.ar

ISBN: 90–5776–135–1

a las penas,
a las vaquitas nada

Contents

Acknowledgments

Well, I finally reached the point where I can write the acknowledgments of my thesis. I imagined this moment countless of times. I do not know how to include all the people that have participated in one way or another in this thesis. My thesis has transcended my professional life and has become something I can not describe with words. Hence, these acknowledgments are a way to say thank you to all the people that has played an important role in my life during the last five years.

My PhD period started with Eliana and me moving to Enschede, The Netherlands. Moving from one culture to a completely different one was a tough experience. Fortunately, Eliana and I had each other to cope with the new situation. I would like to thank Eliana for her help, company and support during the first period of my PhD studies.

During my time in Enschede, I worked at the Formal Methods and Tools Group, at the University of Twente. From that group, I would like to thank Joost-Pieter Katoen, Holger Hermanns and Ed Brinksma for their support and understanding of my need to move to Amsterdam. In Twente I made very good friends. Among others, I would like to mention Dino Distefano, Pedro D'Argenio and Wilco Braam.

After one year in Enschede I moved to the University of Amsterdam, where I spent one year working in Modal Logic. I want to thank Carlos Areces and Maarten de Rijke for teaching me the little logic I know. Carlos was the bridge that took me from Enschede to Amsterdam.

I want to thank Maarten de Rijke. I want to thank him for all he taught me. Working with him was sometimes very tough but, now that I see it with some perspective, always rewarding. I also want to thank him for backing me up during difficult times. He made me feel that he believed that I could finish my studies and my thesis, sometimes he believed it more than I did. I want to thank him for all the opportunities he gave me, even though I could not take advantage of all of them. I want to thank him for allowing me, after much discussion, to switch once again the direction of my PhD studies, this time to Natural Language Processing.

Remko Scha played an important role in the final development of this thesis when he became my co-supervisor. I want to thank him for his time, his patience and most of all, his insightful comments.

Maarten de Rijke, Remko Scha, Pieter Adriaans, Khalil Sima'an, Walter Daelemans, Geert-Jan Kruijff and Peter Grünwald acted as members of my thesis committee, and I am very grateful for their comments. Thanks to Laura Alonso i Alemany, David Ahn, Valentin Jijkoun, Karin Müller, and Stefan Schlobach who proof read the manuscript. I am very grateful to the Netherlands Organization for Scientific Research (NWO) for supporting the research that led to this thesis.

Many thanks go to my very good friends Gabriele Musillo and Caterina Caracciolo. For their support, for all the chatting we had, for the difficult times and easy ones. Gabriele also convinced me that Natural Language Processing is an interesting area. He was right. To Christof Monz, for all the good times we spent together.

To the FaMAF and the people there: Nicolás, Javier, Pedro, Damián. They provided me with working facilities during my long stays in Argentina.

To Nicolás Wolovick, Sergio Urinovsky, Lucrecia Resnik, Maximiliano Oroná, Miguel Valero Espada and Manolo Gragera, Marcos Kurban, Pilar Kurban, and Lorena Sosa, for becoming my family.

To all the people that hosted me while I was traveling back and forth: Carlos, Juan, Caterina, Miguel, Manolo, Jorge, Rubén, Bea, Jesús. To Sisay Fissaha Adafre and Maarten Marx for their logistic help.

I want to thank Miguel Titán Valero Espada and Manolo Titán Gragera; their company and help was invaluable.

I like to thank Laura who helped me a great deal during the last period of my thesis. She provided me with things I had missed for a long time.

To my mother, Marta Silvia, who gave me so many good examples of life. To my father Juan Carlos for all his encouragement. He saw me start my PhD studies but he cannot see me finish. I deeply wish he could read this.

Finally, I want to thank my little son Santiago, he provided me with the strength I needed. We both had to get used to my long periods of absence. Every time I came back he received me as if I had never left. He did not understand why I was away, but I am sure that in the future, once he is able to understand this, he will agree with my decision to continue my PhD studies, and that he will be proud of me. I think a person should pursue his dreams: I have pursued and realized this one. My next one is to be physically close to him.

Gabriel Infante-Lopez
Córdoba, Argentina
February, 2005

Chapter 1
Introduction

Natural language is a very complex phenomenon. Undoubtedly, the sentences we utter are organized according to a set of rules or constraints. In order to communicate with others, we have to stick to these rules up to a certain degree. This set of rules, which is language dependent, is well-known to all speakers of a given language, and it is this common knowledge that makes communication possible. Every sentence has a clear organization: words in an utterance glue together to describe complex objects and actions. This hidden structure, called syntactic structure, is to be recovered by a parser. A *parser* is a program that takes a sentence as input and tries to find its syntactic organization. A parser searches for the right structure among a set of possible analyses, which are defined by a *grammar*. The language model decides what the syntactic components of the sentence are and how they are related to each other, depending on the required level of detail.

Natural language parsers are used as part of many applications that deal with natural language. Applications like question answering, semantic analysis, speech recognition, etc. may rely heavily on parsers. The degree of detail in the information output by the parser may change according to the application, but some amount of parsing plays a role in many language technology applications, and parser performance may be crucial for the overall performance of the end-to-end application.

Designing and building language models is not a trivial task; the design cycle usually comprises designing a model of syntax, understanding its underlying mathematical theory, defining its probability distribution, and finally, implementing the parsing algorithm. The building of every new language model has to complete at least these steps. Each is very complex and constitutes a line of research in itself. To help handling the intrinsic complexity of these steps a sufficient level of *abstraction* is required. Abstraction is important as it helps us deal with complex objects by representing them with a subset of their characteristic features. The selected features characterize the object

for a given task or context. For example, the way we understand or see cars depends very much on the task we want to carry out with them; if we want to drive them, we do not need to know how their engines work, whereas if we are trying to fix a mechanical problem we better do. Moreover, abstraction has proven to be an important scientific principle. The way abstraction helps humans in dealing with complex systems can best be illustrated by the history of computer science. The complexity of systems has increased hand in hand with the introduction of programming languages that allow for increasing levels of abstraction, climbing all the way from machine code to assembly language to imperative languages such as Pascal and C, to today's object-oriented languages such as Java.

Back to natural language parsing — what does abstraction have to do with parsing? Our view is that state-of-the-art natural language models lack abstraction; their design is often *ad hoc*, and they mix many features that, at least conceptually, should be kept separated. In this thesis, we explore new levels of abstraction for natural language models. We survey state-of-the-art probabilistic language models looking for characteristic features, and we abstract away from these features to produce a general language model. We formalize this abstract language model, establish important properties of the models surveyed, and with our abstract model we investigate new directions based on different parameterizations.

1.1 Probabilistic Language Models

Roughly speaking, the syntactic analysis of natural language utterances aims at the extraction of linguistic structures that make explicit how words interrelate within utterances. Syntactic structures for a sentence x are usually conceived as trees $t_1(x), \ldots, t_n(x)$ whose leaves form the sentence x under consideration. A *grammar* is a device that specifies a set of trees. The trees in this set are said to be *grammatical trees*. Indirectly, the concept of a grammatical sentence is defined as follows: a sentence x is said to be *grammatical* if there is a grammatical tree that yields x.

Most natural language grammars tend to assign many possible syntactic structures to the same input utterance. In such situations, we say that the sentence is *ambiguous*. Ambiguity is the most important unsolved problem that natural language parsers face. This contrasts with human language processing, which in most cases selects a single analysis as the preferred one for a given utterance. The task of selecting the single analysis that humans tend to perceive for an input utterance — disambiguation — is an active area of research in the field of natural language processing. Because of the roles of world knowledge, cultural preferences and other extra-linguistic factors, disambiguation can be seen as a decision problem under uncertainty. In recent

years, there have been different proposals for a solution, mainly based on probabilistic models. Probabilistic models assign probabilities to trees and then disambiguate by selecting the tree with the highest probability.

From a formal language perspective, the notion of a *language* coincides with the formal notion of a *set of strings*. *Probabilistic languages* extend this definition so that a language is a *probability distribution* over a set of trees. In particular, a probabilistic language model is a probability distribution over a set of utterance-analysis pairs. Usually, a recursive generative grammar is used to describe a set of possible utterance-analysis pairs, possibly allowing multiple pairs for the same utterance. Crucially, defining a probabilistic language model allows us to view disambiguation as an optimization task, where the *most probable analysis* T^* is selected from among those that a grammar G generates together with an input utterance U. If P is a probability function over utterance-analysis pairs, i.e., a *language model*, we may describe this optimization task as follows:

$$\begin{aligned} T^* &= \operatorname*{argmax}_{T \in G} P(T|U) \\ &= \operatorname*{argmax}_{T \in G} \frac{P(T,U)}{P(U)} \\ &= \operatorname*{argmax}_{T \in G} P(T,U), \end{aligned}$$

where $\operatorname{argmax}_{x \in X} f(x)$ stands for the $x \in X$ such that $f(x)$ is maximal, and where $P(U)$ is the same for all trees and, consequently, can be left out.

The step of enriching a given generative grammar with probabilities is a non-trivial task. Apart from the empirical question of how to do so in a way that allows good disambiguation, in the sense that the model selects the same preferred analysis as humans do, there are various formal and practical issues concerning the definition of a correct model. In order to fully understand a language model, it is necessary to abstract away from specific peculiarities and to identify its relevant features. The latter can be summarized as follows.

Set of possible trees: For a given utterance, the language model chooses a tree that articulates the syntactic structure of the utterance from a fixed set of possible trees. For example, in a Context Free Grammar (CFG), this set is defined by the grammar's tree language.

Probabilities Probabilistic language models assign a probability value to each tree, and this probability value is then used as a way to filter out unwanted trees. A significant part of the definition of a language model is used to establish the way probabilities are assigned to trees. For example, in a probabilistic context free

grammar, each rule has a probability value associated to it, and the probability of a tree is defined as the product of the probabilities assigned to the rules building up the tree.

Parameter estimation Probabilistic models contain many parameters that define the grammar's disambiguation behavior. These parameters have to be estimated. The probability model specifies the set of values the parameters might take. This, in turn, defines the set of possible grammars.

Expressive power The expressive power of language models gives us an idea of their algorithmic complexity, and it allows us to compare different models. For probabilistic models, determining their expressive power is a difficult job because the parameter estimation algorithm has to be taken into account.

Tweaking State-of-the-art parsing algorithms are not just language models — they have been optimized considerably in order to improve their performance on real natural language sentences. Some of the optimized parameters are hard to model and are usually outside the language model.

Parsing complexity Parsing complexity has become an issue again in recent years, because of the appearance of theoretically appealing models that seem very hard to implement efficiently. Parsing complexity should be as low as possible. The aim is to emulate the apparently linear time humans take to process a sentence.

Addressing all of these items in a single thesis would be overly ambitious. This list is meant to provide the context — throughout this thesis we study various specific aspects against this general background.

1.2 Designing Language Models

Designing a parsing algorithm involves a sequence of decisions about:

1. the grammatical formalism,
2. a probabilistic version of the formalism,
3. techniques for estimating probabilities, and
4. a parsing algorithm.

This cycle can be seen almost everywhere in the parsing literature (Collins, 1999; Eisner, 2000; Bod, 1998; Charniak, 1999; Ratnaparkhi, 1999). It seems that every interesting new parser uses a new formalism. The design is time consuming, and usually

parsers are only evaluated empirically. It is clear, however, that better empirical results do not necessarily convey a better understanding of the parsing problem. Usually, descriptions of state-of-the-art language models do not clearly state how the features presented in the previous section are defined or implemented, or how they fit in the design cycle. The grammatical framework is rarely the only decision responsible for the parsers's performance; on top of the decision, there is a lot of tweaking involved. For example, Collins (1997) defines a simple formalism, but in order to achieve his results, all the tweaking reported in (Bikel, 2004) is needed.

Designers of parsers often conflate their decisions about the different features we identify in Section 1.1. For example, let us zoom in on one of the characteristics identified in Section 1.1 and then step back to adopt a more abstract perspective, and see what this gives us. In the definition of today's state-of-the-art language models, Markov chains, and more specifically n-grams, are widely used because they are easy to specify and their probabilities easy to estimate. N-grams are both a component in the definition of a model and a technique to assign probabilities. They are central to language models, and, consequently, every property of language models must be evaluated with respect to n-grams. It might be helpful, though, to step back from n-grams, and think of them as special cases of probabilistic regular languages. Mathematical properties of probabilistic regular languages are as well understood as those of n-grams, but they fit more directly into the overarching theory of formal languages. This perspective allows us to clearly separate the definition of the model (using regular languages) and the procedure for estimating probabilities (using probabilistic regular language induction techniques).

In this thesis, we propose a language modeling formalism that abstracts away from any particular instance. We investigate three state-of-the-art language models and discover that they share a very noticeable feature: the set of rules they use for building trees is built on the fly, meaning that the set of rules is not defined *a priori*. The formalisms we review have two different levels of derivations even though this is not explicitly stated. One level is for generating the set of rules to be used in the second step, and the second step is for building the set of trees that characterize a given sentence. Our formalism, based on *Van Wijngaarden grammars* (W-grammars), makes these two levels explicit.

1.2.1 W-Grammars as the Backbone Formalism

W-grammars were introduced in the 1960s by Van Wijngaarden (1965). They are a very well-known and well understood formalism that is used for modeling programming languages (Van Wijngaarden, 1969) as well as natural languages (Perrault, 1984). W-grammars have been shown to be equivalent to Turing machines (Perrault, 1984),

which are more powerful than we need: most state-of-the-art language models use grammatical formalisms that are much closer to context freeness than to Turing machines. In this thesis, we constrain the set of possible W-grammars in order to come closer to the expressive power of these grammatical formalisms. We denote this constrained version as *CW-grammars*.

Originally, W-grammars did not use probabilities, but part of the work presented below extends the formalism with probabilities. In this way, we define probabilistic CW-grammars (PCW-grammars). We show that probabilities are an essential component of the resulting formalism, not only because of the statistical perspective they bring, but also because of the expressivity they add. With PCW-grammars, we prove that Markovian context free grammars (Collins, 1999; Charniak, 1997), bilexical grammars (Eisner, 2000) and stochastic tree substitution grammars (Bod, 1998) are particular instances of probabilistic CW-grammars. The probabilistic version of CW-grammars helps us to prove properties for these models that were previously unknown.

1.3 Formal Advantages of a Backbone Formalism

From a theoretical point of view, general models help us to clarify the set of parameters a particular instance has fixed, and to make explicit assumptions that underlie a particular instance. It might be the case that these assumptions are not clear, or that, without taking the abstract model into account, the designer of a particular instance is completely unaware of them.

The role of probabilities: Our approach to parsing comes from a formal language perspective: we identify features that are used by state-of-the-art language models and take a formalism off the shelf and modify it to incorporate the necessary features. When analyzing the necessary features from the formal language perspective, the need for probabilities and their role in parsing are the first issue to address. In Chapter 3, we answer many questions regarding the role of probabilities in probabilistic context free grammars. We focus on these grammars because they are central to the formalism we present.

Consistency properties: General models do not add anything *per se*. Their importance is rather in the set of instances they can capture and the new directions they are able to suggest. In Chapter 4, we show that bilexical grammars, Markovian context free grammars and stochastic tree substitution grammars are instances of our general model. Our model has well-established consistency properties which we use to derive consistency properties of these three formalisms.

1.4 Practical Advantages of a Backbone Formalism

From a computational point of view, general models for which a clear parsing algorithm and a relatively fast implementation can be defined, produce fast and clear implementations for all particular instances. New research directions are also suggested by a general formalism. These directions are a consequence of instantiating the models's parameters in a different way or by re-thinking the set of assumptions the particular instances have made. A brief description of the directions explored in this thesis follows.

Explicit use of probabilistic automata: Earlier, we mentioned that Markov models are heavily used in parsing models and that they can be replaced by probabilistic regular languages. Since our formalism is not bound to Markov models, we can use any algorithm for inducing probabilistic automata. In Chapter 5, we explore this idea. We define a type of grammar that uses probabilistic automata for building the set of rules. We compare two different classes of grammars depending on the algorithm used for learning the probabilistic automata. One of them is based on n-grams, and the other one is based on the minimum divergence algorithm (MDI). We show that the MDI algorithm produces both smaller and better performing grammars.

Splitting the training material: The fact that probabilistic automata replace Markov chains in the definition of our model allows us to think of a regular language as the union of smaller, more specific sublanguages. Our intuition is that the sublanguages are easier to induce and that the combination of them fully determines the whole language. In Chapter 6, we explore this idea by splitting the training material before inducing the probabilistic automata, then inducing one automaton for each component, and, finally, combining them into one grammar. We show that in this way, a measure that correlates well with parsing performance can be defined over grammars.

Sequences as features: Our formalism allows us to isolate particular aspects of parsing. For example, the linear order in which arguments appear in a parse tree is a fundamental feature used by language models. In Chapter 7, we investigate which sequences of information better predict sequences of dependents. We compare sequences of part-of-speech tags to sequences of non-terminal labels. We show that part-of-speech tags are better predictors of dependents.

1.5 What Can You Find in this Thesis?

In my opinion there are two different types of research. The first one pushes the frontier of knowledge forward, jumping from one point to a more advanced, better performing one. This pushing forward is sometimes carried out in a disorderly way, leaving many gaps along the way. The second line of research tries to fill in these gaps. Both types are very important, and neither of them can exist without the other. The second provides a solid foundation to the first one in order to make new jumps possible. After a jump, a huge a amount of work is waiting to be done by the second type of research.

This thesis belongs solidly to the second type of research. Here, the reader will find a formal analysis of existing models. The reader will also find a general model that encompasses many of the models studied, as well as some properties these models enjoy — properties that we want the models to have and properties that were not known before and that the general model lets us prove. Finally, the reader will find a few explorations along new research directions also suggested by our model.

We hope that after having read the thesis, the reader will understand the language modeling task better. We also hope to have provided the area of natural language modeling with a more solid background. This background comprises consistency and expressive properties generally believed but not formally proven. In the thesis we also provide initial steps in promising new research directions.

In contrast, the reader will not find a state-of-the-art language model here. The reader will not find any claims regarding the universal structure natural language possesses either. It is very clear to me that the structure of natural language is, at this point, as unknown as it was when I first started. I can only see, so far, that formal languages with complexity up to context freeness can help us quite a lot in handling most syntactic structures.

1.6 Thesis Outline

Chapter 2 (Background and Landscape): This chapter introduces the machinery of formal languages. It covers formal language theory from regular languages to machine learning-based parsing algorithms, touching on context free grammars, W-grammars, and other formalisms.

Chapter 3 (The Role of Probabilities): This chapter investigates the role of probabilities in probabilistic context free grammars. Among others it answers questions like: "can probabilities be mimicked with rules?", "can a grammar fully disambiguate a language?".

Chapter 4 (CW-Grammars as a General Model): This chapter presents our formalism. It shows that bilexical grammars, Markovian context free grammars and stochastic tree substitution grammars are instances of our formalism.

Chapter 5 (Alternative Approaches for Generating Bodies of Grammar Rules): This chapter explores the replacement of n-gram models by a more general algorithm for inducing probabilistic automata. It shows that the alternative algorithm produces smaller and better performing grammars.

Chapter 6 (Splitting training material optimally): This chapter investigates different ways to split the training material before it is used for inducing probabilistic automata. It defines a measure over grammars that correctly predicts their parsing performance.

Chapter 7 (Sequences as Features): This chapter focuses on a very specific aspect of syntax. We compare how two different features, each of them based on sequences of information, help to predict dependents of verbs. One of the features is based on sequences of part-of-speech tags while the other is based on sequences of non-terminals labels.

Chapter 8 (Conclusions): This chapter summarizes and combines conclusions of all the chapters.

Appendix A (Parser Implementation): This appendix discusses aspects related to the implementation of our parsing algorithm for probabilistic CW-grammars. It discusses the prerequisites a grammar has to fulfill for a parser to return the most probable tree. It also discusses some of the optimization techniques we implemented to reduce parsing time.

Appendix B (STOP symbol): This appendix reviews Collins's (1999) explanation of the necessity of the STOP symbol. The appendix uses Markov chains theory to re-explain and justify its necessity.

Chapter 2
Background and Language Modeling Landscape

This chapter has two main goals. The first is to present the background material required for most of the forthcoming chapters, the second one is to situate this thesis in the landscape of natural language parsing. When presenting different approaches for dealing with language models, we adopt a formal language perspective. In Section 2.2 we present regular automata, in Section 2.3 context free grammars, in Section 2.4 W-grammars, and Section 2.5 we deal with other formalisms that do not fall in any of the previous categories but that are used for natural language parsing. Finally, in Section 2.6 we deal with approaches used for natural language parsing that are mainly based on machine learning techniques.

2.1 The Penn Treebank

We start by presenting the material that indirectly defines our task. The Penn treebank (PTB) (Marcus et al., 1993, 1994) is the largest collection of syntactically annotated English sentences, and probably the most widely used corpus in computational linguistics. It is also the basis for the experiments reported in this thesis, and it defines the kind of information we are going to try to associate to naturally occurring sentences.

The PTB project started in 1989. Between then and 1992, 4.5 million words of American English were automatically part of speech (POS) tagged and then manually corrected. Then, each sentence was associated to a parse tree that reflected its syntactic structure. The first release of the PTB uses basically a context free phrase structure annotation for parse trees, where node labels are mostly standard syntactic categories like NP, PP, VP, S, SBAR, etc. In 1995, a new version was released; this second version applied a much richer annotation scheme, including co-indexed null elements (traces)

to indicate non-local dependencies, and function tags attached to the node labels to indicate the grammatical function of the constituents.

The parsed texts come from the 1989 Wall Street Journal (WSJ) corpus, and from the Air Travel Information System (ATIS) corpus (Hemphill et al., 1990). The second release is the basis for the experiments in Chapters 5, 6 and 7. A third release came out later on, using basically the same annotation schema as the second but also including a parsed version of the Brown Corpus (Kucera and Francis, 1967).

The POS tag set is the same in all three releases. It is based on the Brown Corpus tag set, but the PTB project collapses many Brown tags. The reason for this simplification is that the statistical methods, which were used for the first automatic annotation and envisaged as potential "end users" of the treebank, are sensitive to the *sparse data problem*. This problem comes into play if certain statistical events (e.g., the occurrence of a certain trigram of POS tags) do not occur in the training data, so that their probability cannot be properly estimated. The sparseness of the data is related to the size of the corpus and the size of the tag set. Thus, given a fixed corpus size, the sparse data problem can be reduced by decreasing the number of tags. Consequently, the final PTB tag set has only 36 POS tags for words and 9 tags for punctuation and currency symbols. Most of the reduction was achieved by collapsing tags that are recoverable from lexical or syntactic information. For example, the Brown tag set had separate tags for the (potential) auxiliaries *be*, *do* and *have*, as these behave syntactically quite differently from main verbs. In the tag set of the PTB, these words have the same tags as main verbs. However, the distinction is easily recoverable by looking at the lexical items. Other tags that are conflated are prepositions and subordinating conjunctions (conflated in `IN`) and nominative and accusative pronouns (conflated in `PRP`), as these distinctions are recoverable from the parse tree by checking wether `IN` is under `PP` or under `SBAR`, and whether `PRP` is under `S` or under `VP` or `PP`.

The syntactic annotation is guided by the same considerations as POS tagging. For instance, there is only one syntactic category, labeled `SBAR`, for *that-* or *wh*-clauses and only one `S` for finite and non-finite (infinitival or participial) clauses, although the two types behave syntactically quite differently. Again, the argument is that these distinctions are recoverable by inspecting the lexical material in the clause; and parsers basically use the simple treebank categories.

In general, only maximal projections (`NP`, `VP`, ...) are annotated, i.e., intermediate X-bar levels (`N'`, `V'`) are left unexpressed, with the exception of `SBAR`. In the first release of the PTB, the distinction between complements and adjuncts of verbs was expressed by attaching complements under a `VP` as sisters of the verb and by adjoining adjuncts at the `VP` level. In the second release, both complements and adjuncts are attached under `VP`.

In our experiments, we used the PTB as training and test material. We train models

on the data provided by the PTB and we try to obtain, for instances not used in the training material, the tree that the PTB would have associated to them. In our experiments, we did not work directly with the PTB, but with a dependency version of the trees in the PTB. That is, we transformed the PTB into dependency trees to obtain the training material for our experiments.

2.1.1 Transformation of the Penn Treebank to Dependency Trees

The experiments we present in the forthcoming chapters use unlabeled dependency structures. We choose to use such structures because they allow us to isolate better than phrase structures the aspects of syntax and language modeling that we want to investigate. We transformed all trees in the PTB to dependency trees; the transformation procedure is based on the ideas of Collins (1999). He defines the transformation from phrase structure trees to dependency trees as a function from a tree to its associated dependencies. Two steps are involved in defining this transformation. First, the trees are lexicalized by adding the word to each non-terminal label; second, dependencies are derived by extracting $n - 1$ dependencies from each rule with n children. Let us explain both steps in more detail.

Step 1: Lexicalization of parse trees. Black et al. (1993); Jelinek et al. (1994); Magerman (1995a) introduced the concept of *lexicalization* of non-terminal rules as a way of improving parsing accuracy. Each non-terminal node in a tree is modified by adding the *head-word* to it. Headwords are assigned through a function that identifies the "head" of each rule in the grammar. More precisely, the function $head(X \rightarrow Y_1, \ldots, Y_n)$ returns a value h such that $1 \leq h \leq n$, where h is the index of the head.

The function *head* is used for adding lexical information to all non-terminals in a tree. The function *headword* adds lexical information to all non-terminals in a tree and *headword* is defined recursively as follows.

> **Base case:** If a non-terminal X is on the left-hand side of a rule $X \rightarrow x$, where X is a non-terminal part of speech, and x is a lexical item; then $headword(X \rightarrow x) = x$.
>
> **Recursive case** Assume X is a non-terminal on the left-hand side of a rule $X \rightarrow Y_1 \ldots Y_n$, and $h = head(X \rightarrow Y_1 \ldots Y_n)$; we put $headword(X) = headword(Y_h)$

Step 2: Derivation of dependencies from lexicalized trees. With the headword for each non-terminal in the tree defined, the next step is to identify a set of dependencies between words in the sentence. A dependency is a relationship between

two word-tokens in a sentence, a *modifier* and its *head*, which we will write as $modifier \rightarrow head$. The dependencies for a given tree are derived in two ways:

- Every rule $X \rightarrow Y_1 \ldots Y_n$ such that $Y_1, \ldots, Y_n$ are non-terminals and $n \geq 2$ contributes the following set of dependencies: $\{headword(Y_i) \rightarrow headword(Y_h) : 1 \leq i \leq n, i \neq h\}$, where $h = head(X \rightarrow Y_1 \ldots Y_n)$.
- If X is the root non-terminal in the tree, and x is its headword, then $x \rightarrow$ END is a dependency

Clearly, the key component in the transformation process is the function *head*. This function has been implemented mainly as a lookup table. For further details on the definition of the function *head*, see (Collins, 1999, Appendix A).

The PTB provides us with the training material for inducing our own grammars. The grammars learnt in this thesis are Probabilistic Constrained W-Grammars (PCW Grammars), a new formalism which is presented in Chapter 4. Our formalism is related to probabilistic regular languages, probabilistic context free grammars and W-grammars. In this chapter we present these three different formalisms. The relation of PCW Grammars to each of the three formalism will become clear in the remainder of the thesis.

2.2 Probabilistic Regular Automata

Let us start by recalling some preliminary notation and concepts from formal languages. Let Σ be a finite alphabet and Σ^* the (infinite) set of all strings that can be built from Σ; ϵ denotes the empty string. A *language* is a subset of Σ^*. By convention, symbols from Σ will be denoted by letters from the beginning of the alphabet $(a, b, c, \ldots)$ and strings from Σ^* will be denoted by letters from from the end of the alphabet $(\ldots, x, y, z)$. The size of a string $x \in \Sigma^*$ is written $|x|$. If $\equiv$ denotes an equivalence relation on a set X, we write $[x]$ to denote the component of the equivalence class containing x, that is $[x] = \{y \in X : y \equiv x\}$.

A *probabilistic language* L is a probability distribution over Σ^*. The probability of a string $x \in \Sigma^*$ is denoted as $p_L(x)$. If the distribution is modeled by some syntactic machine A, the probability of x according to the probability distribution defined by A is denoted as $p_A(x)$.

Two probabilistic languages L and L' are equal if $\forall w : p_L(w) = p_{L'}(w)$; note that this definition implicitly states that the two languages contain the same strings.

We now introduce deterministic finite automata (for a more detailed introduction, see (Hopcroft and Ullman, 1979)):

2.2.1. DEFINITION. A *deterministic finite automaton* (DFA) A is a tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states;
- q_0 is the initial state;
- Σ is the alphabet;
- $\delta : Q \times \Sigma \rightarrow Q$ is a transition function;
- $F \subseteq Q$ is the set of final states.

We extend δ in the usual way to a function $\delta : Q \times \Sigma^* \rightarrow Q$ by putting $\delta(q, \epsilon) = q$ and $\delta(q, aw) = \delta(\delta(q, a), w)$.

We now adapt this definition to the probabilistic setting:

2.2.2. DEFINITION. A *probabilistic deterministic finite automaton* (PDFA) A is a tuple $(Q, \Sigma, \delta, q_0, \gamma)$, where Q, Σ, δ, q_0 define a DFA and γ is a function with two profiles:

- $\gamma : Q \times \Sigma \rightarrow \mathbb{R}$ (transition probabilities) and,
- $\gamma : Q \rightarrow \mathbb{R}$ (final-state probabilities).

The function γ is recursively extended to $\gamma : Q \times \Sigma^* \rightarrow \mathbb{R}$ such that $\gamma(q, \epsilon) = 1$ and $\gamma(q, ax) = \gamma(q, a) \cdot \gamma(\delta(q, a), x)$. The probability of a string x starting from the state q is defined as $p(q, x) = \gamma(q, x) \cdot \gamma(\delta(q, x))$. The probability of a string x is $p(x) = p(q_0, x)$. Let X be a set of strings, $p(X) = \prod_{x \in X} p(x)$. We say that a probability distribution over L is a *probabilistic deterministic regular language* (PDRL) if it is produced by a PDFA. As probabilistic languages define probability distributions over Σ, it is required that $0 \leq p(x) \leq 1$ and $p(\Sigma^*) = 1$ (*consistency condition*). In contrast to non-probabilistic automata, the deterministic and non-deterministic version of probabilistic automata are not equivalent (Dupont et al., 2004).

Automata play a fundamental role in this thesis, both from a theoretical and from a practical point of view. In Chapters 5, 6 and 7 we use automata for building PCW-grammars. We induce automata from training material and then combine them for defining grammars. Inferring automata, then, is a fundamental activity in this thesis.

2.2.1 Inferring Probabilistic Deterministic Finite Automata

The problem of inferring a PDFA can be seen as a particular instance of the wider task of inferring formal grammars from a finite set of examples. This task has been extensively studied under the paradigm of identification in the limit (Carrasco and Oncina, 1994, 1999; Thollard et al., 2000). Under this paradigm, a learner is supplied by an (infinite) stream of input data, generated according to a language. The learning problem is to identify the language that explains the data stream. In every iteration step, the learner reads another piece of data and outputs a grammar of a given family of grammars. The main questions addressed by this framework are

- For which family of grammars are there algorithms which identify a correct hypothesis at some point in time (in the limit) for any instance of the representation class, and just output syntactic variants of the result from that point on?
- Does a specific algorithm identify any/one specific instance of a representation class in the limit?

Note that the learner is not asked to realize that he has found a correct hypothesis. Gold (1967) introduced this paradigm and he showed that negative examples are required even for learning the class of regular languages, or equivalently for inducing deterministic finite automata (DFAs).

When training data includes negative examples, the regular positive and negative inference (RNPI) algorithm can be used for learning PDFAs (Oncina and Garcia, 1992). This algorithm was proven to identify in the limit the class of regular languages. Negative information, however, is not always available in practical domains such as natural language or speech applications. A promising approach to learn DFAs only from positive examples has been proposed by Denis (2001). There is, however, one more complication: real data is generally noisy, because the data itself does not consistently follow a formal syntax. If we choose to learn PDFAs we can, in principle, handle both the lack of negative information and the presence of noise. One possible approach to learn PDFAs consists in reducing the class of machines of interest to a special case of *Markov chains* called *n-grams*. These models, however, form a proper subclass of PDFAs in which the maximal order of dependence between several symbols in a sequence is bound. We come back to this difference in more detail in Chapter 5 by exploring how the two approaches deal with the creation of context free like rules used for parsing.

Several inference algorithms for probabilistic automata have been proposed. For example, Carrasco and Onciana's ALERGIA algorithm (Carrasco and Oncina, 1994, 1999), a stochastic extension of the RPNI algorithm, is not restricted to learning acyclic automata. Most of the algorithms for inducing PDFAs follow the same approach; they

start by building an acyclic automaton, called the *initial automaton*, that accepts only the strings in the training material. Next, the algorithms generalize over the training material by merging states in the initial automaton. In other words, they usually build a sequence of automata $A_0, \ldots, A_k$, where A_0 is the initial automaton and A_j results from merging some states in A_{j-1} into a single state in A_j. For example, the ALERGIA algorithm merges states locally, which means that pairs of states will be merged if the probabilistic languages associated to their suffixes are close enough. This local merging implies that there is no explicit way to bind the divergence between the distribution defined by the initial automaton and the distribution defined by any automaton in the sequence of automata built by the algorithm.

To avoid this problem, the minimal divergence algorithm (MDI) (Thollard et al., 2000) trades off minimal size and minimal divergence from the training sample. We use the MDI algorithm in our experiments, and, since it is a key component in the forthcoming chapters, we will now provide a more detailed presentation of its working principle.

2.2.2 The MDI Algorithm

Before discussing the MDI algorithm, let us introduce some useful concepts and notations.

2.2.3. DEFINITION. The *relative entropy* or *Kullback-Leibler divergence* between two probability distributions $P(x)$ and $Q(x)$, defined over the same alphabet A_X, is

$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right),$$

where log denotes the logarithm base 2.

The Kullback-Leibler divergence is a quantity which measures the difference between two probability distributions. One might be tempted to call it a "distance," but this would be misleading, as the Kullback-Leibler divergence is not symmetric.

Let I_+ denote the *positive sample*, i.e., a set of strings belonging to the probabilistic language we are trying to model. Let $PTA(I_+)$ denote the *prefix tree acceptor* built from the positive sample I_+. The prefix tree acceptor is an automaton that only accepts the strings in the sample and in which common prefixes are merged, resulting in a tree-shaped automaton. Let $PPTA(I_+)$ denote the *probabilistic prefix tree acceptor*. This is the probabilistic extension of the $PTA(I_+)$, in which each transition has a probability proportional to the number of times it is used while generating, or equivalently parsing, the sample of positive examples. Let $C(q)$ denote the count of state q, that is, the number of times the state q was used while generating I_+ from $PPTA(I_+)$.

Let $C(q, \texttt{END})$ denote the number of times a string I_+ ended on q. Let $C(q, a)$ denote the count of the transition (q, a) in $PPTA(I_+)$. The $PPTA(I_+)$ is the maximal likelihood estimate built from I_+. In particular, for $PPTA(I_+)$, the probability estimates are:

$$\hat{\gamma}(q, a) = \frac{C(q, a)}{C(q)} \text{ and } \hat{\gamma}(q) = \frac{C(q, \texttt{END})}{C(q)}$$

Figure 2.1.(a) shows a prefix tree acceptor built from the sample

$$I_+ = \{a, bb, bba, baab, baaaba\}.$$

Let A be an automaton with set of states Q, and let Π be a partition of Q. The probabilistic automaton A/Π denotes the automaton derived from A with respect to the partition Π. A/Π is called the *quotient automaton* and it is obtained by merging states of A belonging to the same component π in Π. When a state q in A/Π results from the merging of states q' and q'' in Q, the following equalities must hold:

$$\gamma(q, a) = \frac{C(q', a) + C(q'', a)}{C(q') + C(q'')}, \forall a \in \Sigma, \text{ and } \gamma(q) = \frac{C(q', \texttt{END}) + C(q'', \texttt{END})}{C(q') + C(q'')}.$$

Quotient Automata and Inference Search Space

We define $Lat(PPTA(I_+))$ to be the *lattice of automata* which can be derived from $PPTA(I_+)$, that is, the set of all probabilistic automata that can be derived from $PPTA(I_+)$, by merging some states. This lattice defines the search space of all possible PDFAs that generalize the training sample (Dupont et al., 1994).

Figure 2.1.(b) shows the quotient automaton $PPTA(I_+)/\Pi$ corresponding to the partition

$$\Pi = \{\{0, 1, 3, 5, 7, 10\}, \{2, 8, 9\}, \{4\}, \{6\}\}.$$

for the prefix tree acceptor in Figure 2.1.(a). Each component of the partition represents a set of merged states. Recall that each state is named with a natural number. Each component is denoted with the number of the minimal state inside it.

By construction, each of the states in $PPTA(I_+)$ corresponds to a unique prefix. The prefixes may be sorted according to the standard order $<$ on strings. The standard order is the lexical order found in dictionaries. For instance, according to the standard order, the first strings in the alphabet $\Sigma = \{a, b\}$ are $\epsilon < a < b < aa < ab < ba < bb < aaa < \ldots$. This order also applies to the prefix tree states. A partition of the set of states of $PPTA(I_+)$ consists of an ordered set of subsets, each subset receiving the rank of its state of minimal rank in the standard order. The MDI algorithm proceeds in $N - 1$ steps, where $N = \mathcal{O}(I_+)$ is the number of states of $PPTA(I_+)$. The partition $\Pi(i)$ at step i, that is, the quotient automaton obtained at step i, is obtained by merging the two first subsets, according to the standard order defined above, of the partition

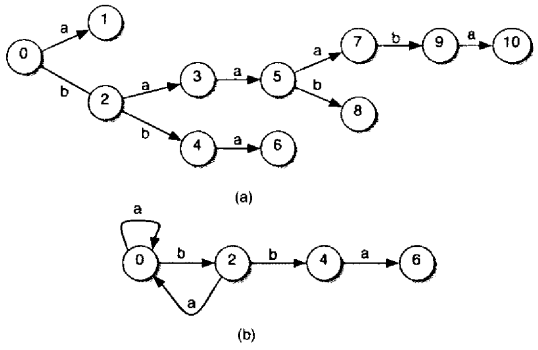


Figure 2.1: A prefix acceptor (a) and a quotient automaton (b).

$\Pi(i-1)$ at step $i-1$, so that $PPTA(I_+)/\Pi(i)$ is a compatible automaton. Two automata are said to be *compatible* if the following holds. Assume A_1 is a temporary solution and A_2 is a tentative new solution derived from A_1 by merging some states, and let $\Delta(A_1, A_2) = D_{KL}(PPTA(I_+)||A_2) - D_{KL}(PPTA(I_+)||A_1)$ be the divergence increment while going from A_1 to A_2. The new solution A_2 is considered to be *compatible* with the training data if the divergence increment relative to the size reduction, that is, the reduction of the number of states, is small enough. Formally, let `alpha` denote a compatibility threshold. The compatibility is satisfied if:

$$\frac{\Delta(A_1, A_2)}{|A_1| - |A_2|} < \texttt{alpha}.$$

Summing up, the MDI algorithm takes a set of strings as input and outputs a PDFA. Theoretically, the set of strings fed to the MDI algorithm are produced by the unknown PDFA, the MDI algorithm tries to recover.

Clearly, the MDI algorithm might output different automata for different values of `alpha`. Then, a valid question is how to choose the right value of `alpha`. In order to determine the best value of `alpha`, we will now discuss how to evaluate automata.

2.2.3 Evaluating Automata

We use two measures to evaluate the quality of a probabilistic automaton. The first, called *test sample perplexity* (PP), is based on the *test sample perplexity* of strings x belonging to a test sample, according to the distribution defined by the automaton. Let A be an automaton, and let p be the probability distribution defined by A. The perplexity PP associated to A is defined as

$$PP = 2^{LL},$$

where

$$LL = -\frac{1}{||S||}\sum_{x \in S} \log P(x).$$

$P(x)$ is the probability assigned to the string x by the automata A, S is a sample set of strings that follow the right distribution, and $||S||$ is the number of symbols in S. The minimal perplexity $PP = 1$ is reached when the next symbol is always predicted with probability 1 from the current state, while $PP = |\Sigma|$ corresponds to uniformly guessing from an alphabet of size $|\Sigma|$. Intuitively, perplexity (PP) measures the uncertainty faced by an automaton when it is fed a new string.

It is hard to track down the origin of LL, the most appealing explanation we found in (Jurafsky and Martin, 2000) is related to cross entropy. Let us see how. The cross entropy is useful when we do not know the actual probability distribution p that generated sequences of words $w_1, \ldots, w_n$. It allows us to estimate some $\hat{p}$ which is a model of p, i.e., an approximation to p. The *cross entropy* of $\hat{p}$ on p is defined by

$$H(p, \hat{p}) = \lim_{n \to \infty} \frac{1}{n} \sum_{W \in L} p(w_1, \ldots, w_n) \log \hat{p}(w_1, \ldots, w_n).$$

That is, we draw sequences of words w_i according to the probability distribution p, but sum the log of their probability according to $\hat{p}$.

If the automaton is a stationary ergodic process, then using the Shannon-McMillan-Breiman theorem we rewrite

$$H(p, \hat{p}) = \lim_{n \to \infty} -\frac{1}{n} \log \hat{p}(w_1, \ldots, w_n).$$

For sufficiently large n, we can rewrite:

$$H(p, \hat{p}) = -\frac{1}{n} \log \hat{p}(w_1, \ldots, w_n),$$

which corresponds to our definition of LL.

The second measure we use to evaluate the quality of an automaton is the number of *missed samples* (MS). A missed sample is a string in the test sample that the automaton failed to accept. One such instance is enough to have PP undefined (LL infinite). Since an undefined value of PP only witnesses the presence of at least one MS, we decided to count the number of MS separately, and compute PP without taking MS into account. This choice leads to a more accurate value of PP, and, moreover, the value of MS provides us with information about the generalization capacity of automata: the lower the value of MS, the larger the generalization capacities of the automaton. The usual way to circumvent undefined perplexity is to smooth the resulting automaton with unigrams, thus increasing the generalization capacity of the automaton, which is usually paid for with an increase in perplexity. We decided not to use any smoothing techniques, as we want to compare bigram-based automata with MDI-based automata in the cleanest possible way.

2.3 Probabilistic Context Free Grammars

Context free grammars are a key component in our formalism, PCW-grammars. We use them for proving the consistency properties of our own formalism and as the backbone of our parsing algorithm. We present them here following the standard conventions (e.g., (Aho and Ullman, 1972; Hopcroft and Ullman, 1979)).

A *context free grammar* (CFG) is defined as quadruple $\langle T, N, S, R \rangle$, consisting of a terminal vocabulary T, a non-terminal vocabulary N, a distinguished symbol $S \in N$, usually called the *start symbol* or *axiom* and a set of productions or rewriting rules P. The sets T, N, and R are finite; T and N are disjoint ($T \cap N = \emptyset$), and their union can be denoted V ($V = T \cup N$). In the case of a CFG, the rules of the grammar will be written as $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in V^*$. Rules of the form $A \rightarrow w$, where $w \in T$ are referred to as *lexical rules*.

Given a CFG G, a *parse tree* based on G is a rooted, ordered tree whose non-terminal nodes are labeled with elements of N and whose terminal nodes are labeled with elements of T. Those nodes immediately dominating terminal nodes will be referred to as *preterminal*; the other non-terminal nodes will be referred to as non-lexical. A syntactic tree based on G is said to be *well-formed* with respect to G if for every non-terminal node with label A and daughter nodes labeled $A_1, \ldots, A_k$, there is a rule in P of the form $A \rightarrow A_1 \ldots A_k$. We shall distinguish between a tree that is compatible with the rules of the grammar, and a tree that also spans a sentence. A syntax tree is said to be generated by a grammar G if

1. The root node is labeled with S (the distinguished symbol).

2. The tree is well-formed with respect to G.

The conventional rewrite interpretation of CFGs (see, for instance, (Hopcroft and Ullman, 1979)) will also be used in the definition of our stochastic models. Given two strings w_1 and $w_2 \in V^*$, we say that w_1 *directly derives* w_2, if $w_1 = \delta A \gamma$, $w_2 = \delta \alpha \gamma$, and $A \rightarrow \alpha$ is a rule in P. Similarly, w_1 *derives* w_2 (in one or more steps) if the reflexive transitive closure of A directly derives α (written $A \rightarrow^* \alpha$ to indicate the application of zero or more rules in order to derive string α from non-terminal A).

A *probabilistic context free grammar* (PCFG) is a context free grammar G with set of rules R in which a probability has been attached to every rule in R. That is, for every rule of grammar G, $A \rightarrow \alpha \in R$, it must be possible to define a probability $P(A \rightarrow \alpha)$. Moreover, the probabilities associated to all the rules that expand the same non-terminal must sum up to 1.

$$\sum_{A \rightarrow \alpha \in R} P(A \rightarrow \alpha) = 1.$$

Using an auxiliary notation $A_{i,j}$ to denote a non-terminal node A of the parse tree spanning positions of the sentence from i through j, we can define the three assumptions of the model:

1. Place invariance: $\forall i, P(A_{i,i+|\zeta|} \rightarrow \zeta)$ is the same.
2. Context freedom: $P(A_{ij} \rightarrow \zeta | \text{anything outside } i \text{ through } j) = P(A_{ij} \rightarrow \zeta)$.
3. Ancestor freedom: $P(A_{ij} \rightarrow \zeta | \text{ any ancestor nodes above } A_{ij}) = P(A_{ij} \rightarrow \zeta)$.

The probabilities attached to the rules can be used either to heuristically guide the parsing process or to select the most probable parse tree(s). The probability of a certain derivation, i.e., a parse tree, can be computed by multiplying the probabilities of all the productions applied in the derivation process. Let ψ be a finite parse tree, well-formed with respect to G, and let f be the counting function, such that $f(A \rightarrow \alpha; \psi)$ indicates the number of times rule $A \rightarrow \alpha$ has been used to build tree ψ. Then we can write:

$$P(\psi) = \prod_{A \rightarrow \alpha \in R} P(A \rightarrow \alpha)^{f(A \rightarrow \alpha; \psi)}.$$

In contrast to PDFAs, where the consistency property is defined over the set Σ^*, the consistency property for PCFGs is defined over the set ψ_G of trees accepted by G. P is said to be *consistent* if

$$\sum_{\psi \in \Psi_G} P(\psi) = 1.$$

The consistency property for PCFGs is not always satisfied, (see, for instance (Booth and Thompson, 1973)), because it depends on the probability distribution over the rules, $P(A \rightarrow \alpha)$. However, if, as usual, the estimation of the probabilities is carried out by means of the maximum likelihood estimator (MLE) algorithm, it can be proved that this property holds. Chi and Geman (1998) generalizes this approach by means of the relative weighted frequency method.

2.4 W-Grammars

In the mid-1960s, Aad van Wijngaarden developed a grammar formalism specially for the formal definition of programming languages, based on a combination of generality and simplicity. The formalism was first presented by Van Wijngaarden (1965) and was adopted for a new programming language design project that eventually produced ALGOL 68. Grammars within this formalism are called *van Wijngaarden grammars* often shortened to *vW-grammars* or *W-grammars*. Some authors used the name *two-level grammars*, but this could lead to confusion, since affix-grammarians also use it

to name the general class that includes W-grammars, affix grammars and the rest of variants of AGs as well. Therefore, the name W-grammars is preferred here.

We use the concept of two-level grammars to develop our own formalism, which is a constrained version of W-grammars. W-grammars are too expressive and the computational complexity of dealing with such big expressivity is *very high*. Our formalism is very close to the PCFG formalism in expressivity but it uses many ideas found in W-grammars. We give here a brief introduction to W-grammars for comparisons with our own formalism.

The basic idea of W-grammars is that, rather than enumerating a finite set of rules over a finite symbol alphabet, a W-grammar constructs a finite meta-grammar that generates the symbols and rules of the grammar. In this way, one can define a Chomsky-type grammar with infinitely many non-terminals and rules.

The definition given here follows (Chastellier and Colmerauer, 1969):

2.4.1. DEFINITION. A *W-grammar* is defined by the 6th-tuple $(V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ such that:

- V is a set of symbols called *variables*. Elements in V are noted with calligraphic characters, e.g., $\overline{A}, \overline{B}, \overline{C}$.
- NT is a set of symbols called *non-terminals*. Elements in NT are noted with upper-case letters, e.g., X, Y, Z.
- T is a set of symbols called *terminals*, noted with lower-case letters,e.g., a, b, c, such that V, T and NT are pairwise disjoint.
- $\overline{S}$ is an element of V called *start symbol*.
- $\xrightarrow{m}$ is a finite binary relation defined on $(V \cup NT \cup T)^*$ such that if $x \xrightarrow{m} y$ then $x \in V$. The elements of $\xrightarrow{m}$ are called *meta-rules*.
- $\xrightarrow{s}$ is a finite binary relation on $(V \cup NT \cup T)^*$ such that if $r \xrightarrow{s} s$ then $s \neq \epsilon$. The elements of $\xrightarrow{s}$ are called *pseudo-rules*.

W-grammars are rewriting devices. As rewriting devices, they consist of rewriting rules, but, in contrast to standard rewriting systems, the rewriting rules of W-grammars do not exist a-priori. Pseudo-rules and meta-rules provide mechanisms for building the rules that will actually be used in the rewriting process. The rewriting rules are denoted by $\overset{w}{\Longrightarrow}$ and are defined below. In general, a rule $\alpha \overset{w}{\Longrightarrow} \beta$ indicates that α should be rewritten as β. For W-grammars, these rules are built by first selecting a pseudo-rule, and second, using meta-rules for instantiating all the variables that the pseudo-rule might contain. Once all variables have been instantiated, the resulting relation can be viewed as a derivation rule, like in context free grammars. The different values a variable in a pseudo-rule can take are given by the meta-rules. In other words, the

relation generated by meta-rules defines the set of values a variable can have. Once all variables in a pseudo-rule have been instantiated, we obtain a "real" rule.

The idea of rule instantiation is explained in the following example.

2.4.2. EXAMPLE. Let $W = (V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ be a W-grammar such that $V = \{\overline{S}, \overline{A}\}$, $NT = \{S, A\}$.

meta-rules	pseudo-rules
$\overline{S} \xrightarrow{m} S$	$S \xrightarrow{s} \overline{A}$
$\overline{A} \xrightarrow{m} A\overline{A}$	$A \xrightarrow{s} a$
$\overline{A} \xrightarrow{m} A$	

For building a rewriting rule, we first take a pseudo-rule, say $S \xrightarrow{s} \overline{A}$, with all its variables instantiated. For this particular pseudo-rule, the variable $\overline{A}$ is the only one that needs to be instantiated. Possible instantiations are defined through meta-rules. For this example, the variable $\overline{A}$ can be rewritten as $\overline{A} \xrightarrow{m} A\overline{A} \xrightarrow{m} AA\overline{A} \xrightarrow{m} AAAA$. Replacing the instantiation $AAAA$ for the variable $\overline{A}$ in $S \xrightarrow{s} \overline{A}$ yields the rewriting rule $S \xRightarrow{w} AAAA$. Note that pseudo-rules are used only one time to construct $\xRightarrow{w}$ rules.

In order to formalize the derivation process and to define the language accepted by a W-grammar, we first extend $\xrightarrow{m}$ to a relation between a sequence of strings in the usual way: If $x \xrightarrow{m} y$ then $vxw \xrightarrow{m} vyw$ for any $x, y, v, w \in (NT \cup T \cup V)^*$. With $\xrightarrow{m}{}^*$ we denote the reflexive and transitive closure of $\xrightarrow{m}$. The relation $\xRightarrow{w}$ is formally defined as follows.

2.4.3. DEFINITION. Let r and s be in $(NT \cup T)^*$. We say that $r \xRightarrow{w} s$ if there exist r', s' in $(V \cup NT \cup T)^*$ such that $r' \xrightarrow{s} s'$ and such that r and s can be obtained from r' and s' respectively by substituting each occurrence of a variable U by a string $t \in (T \cup NT)^*$ such that $U \xrightarrow{m}{}^* t$. If U occurs more that once in r' or s', the same string t has to be substituted in all occurrences. The elements of $\xRightarrow{w}$ are called *w-rules*.

A w-rule $\alpha \xRightarrow{w} \beta$ defines only one step in the rewriting process. The entire rewriting procedure is defined by extending $\xRightarrow{w}$ to elements in $(T \cup NT)^*$ as follows. If $r \xRightarrow{w} s$ then $p, r, q \xRightarrow{w} p, s, q$ for any r, s, p and q in $(T \cup NT)^*$. Also, $\xRightarrow{w}{}^*$ is the reflexive and transitive closure of $\xRightarrow{w}$. When a string is rewritten using w-rules, we call that derivation a *w-derivation*.

2.4.4. EXAMPLE. Let $W = (V, NT, T, N, \xrightarrow{m}, \xrightarrow{s})$ be a W-grammar with $V = \{\overline{N}, \overline{L}\}$, $NT = \{U\}$, $T = \{a, b, c\}$ and the set of meta-rules and pseudo-rules as follows:

meta-rules	pseudo-rules
$\overline{N} \xrightarrow{m} U$	$\overline{N} \xrightarrow{s} \overline{N}a, \overline{N}b, \overline{N}c$
$\overline{N} \xrightarrow{m} \overline{N}U$	$\overline{N}U\overline{L} \xrightarrow{s} \overline{N}\overline{L}, \overline{L}$
$\overline{L} \xrightarrow{m} a$	$U\overline{L} \xrightarrow{s} \overline{L}$
$\overline{L} \xrightarrow{m} b$	
$\overline{L} \xrightarrow{m} c$	

This grammar generates the language $L = \{(a,)^n(b,)^n(c,)^n : n > 0\}$, which is known to be context-sensitive (Hopcroft and Ullman, 1979). Note that meta-rules will produce sequences of non-terminals U, while a variable $\overline{N}$ can be instantiated with any string in $\{U^* : n > 0\}$.

The pseudo-rule $\overline{N} \xrightarrow{s} \overline{N}a, \overline{N}b, \overline{N}c$ indicates how many a's, b's and c's the body of the w-rule resulting from using this w-rule will have. The pseudo-rule $\overline{N}U\overline{L} \xrightarrow{s} \overline{N}\overline{L}, \overline{L}$ is used to build w-rules that rewrite the sequence of U's to its corresponding non-terminals. Table 2.1 shows the w-rules used for w-deriving a, a, b, b, c, c.

w-rule	how to derive it
$UU \xRightarrow{w} UUa, UUb, UUc$	From $\overline{N} \xrightarrow{s} \overline{N}a, \overline{N}b, \overline{N}c$ with $\overline{N} := UU$.
$UUa \xRightarrow{w} Ua, a$	From $\overline{N}U\overline{L} \xrightarrow{s} \overline{N}\overline{L}, \overline{L}$ with $\overline{N} := U$ and $\overline{L} := a$.
$UUb \xRightarrow{w} Ub, b$	From $\overline{N}U\overline{L} \xrightarrow{s} \overline{N}\overline{L}, \overline{L}$ with $\overline{N} := U$ and $\overline{L} := b$.
$UUc \xRightarrow{w} Uc, c$	From $\overline{N}U\overline{L} \xrightarrow{s} \overline{N}\overline{L}, \overline{L}$ with $\overline{N} := U$ and $\overline{L} := c$.
$Ua \xRightarrow{w} a$	From $U\overline{L} \xrightarrow{s} \overline{L}$ with $\overline{L} := a$.
$Ub \xRightarrow{w} b$	From $U\overline{L} \xrightarrow{s} \overline{L}$ with $\overline{L} := b$.
$Uc \xRightarrow{w} c$	From $U\overline{L} \xrightarrow{s} \overline{L}$ with $\overline{L} := c$.

Table 2.1: W-rules used to w-derive string a, a, b, b.

Finally, using the w-rules built in Table 2.1, the w-derivation of string a, a, b, b, c, c is:

$$
\begin{aligned}
UU \xRightarrow{w} UUa, UUb, UUc &\xRightarrow{w} Ua, a, UUb, UUc \\
&\xRightarrow{w} a, a, UUb, UUc \\
&\xRightarrow{w} a, a, Ub, b, UUc \\
&\xRightarrow{w} a, a, b, b, UUc \\
&\xRightarrow{w} a, a, b, b, Uc, c \xRightarrow{w} a, a, b, b, c, c
\end{aligned}
$$

W-derivations are represented as a tree. The tree corresponding to the derivation shown in Example 2.4.4 is given in Figure 2.2. Commas separate the units for replacing or rewriting symbols and are a key point in the definition of W-grammars.

Finally, we define the string language generated by a W-grammar as follows:

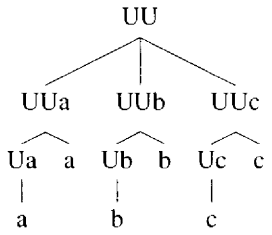


Figure 2.2: A derivation tree for the string a, a, b, b, c, c.

2.4.5. DEFINITION. Let W be a W-grammar. The *string language* $L(W)$ generated by W is the set defined by:

$$L(W) = \left\{ \beta \in T^+ : \text{ there exists } \alpha \in (T \cup NT)^* \text{ such that } \overline{S} \xrightarrow{m}{}^* \alpha \overset{w}{\Longrightarrow}{}^* \beta \right\}$$

Intuitively, a string β belongs to the language $L(W)$ if and only if there is an α that is an instance of the start variable $\overline{S}$, and if there are rules $\overset{w}{\Longrightarrow}$ can be build from α to derive β.

As with CFGs, W-grammars define a tree-language:

2.4.6. DEFINITION. Let $W = (V, NT, T, N, \xrightarrow{m}, \xrightarrow{s})$ be a W-grammar and $L(W)$ its language. Let x be an element in $L(W)$. A tree yielding x is defined as the w-derivation used for w-deriving x.

A w-tree pictures the w-rules that have been used for deriving a string. The way in which the w-rule has been obtained from the pseudo-rules and meta-rules remains hidden, i.e., there is no way to deduce the way in which variables have been instantiated. This property is very important, and it also holds for our formalism. We come back to this point in Chapter 4.

2.5 Further Probabilistic Formalisms

In the literature, many different approaches have been proposed for dealing with natural language parsing. In this section we present a brief review of existing formalisms to place our approaches into a bigger context of probabilistic formalism. Since many formalisms have been proposed, we can only provide a short overview of only some of them. There are many relations between the formalisms discussed in this section and the work presented in this thesis, but we only sketch the most fundamental relations here. More specific relations are described in Chapter 4, where we relate our formalism to three specific formalisms: bilexical grammars, Markovian CFGs, and data-oriented parsing.

2.5.1 Dependency Based Approaches

The probabilistic link grammar model of Lafferty et al. (1992), grammatical trigrams, might be considered the earliest work on probabilistic dependency grammars. It is a generative model that specifies a probability distribution over the space of parse/sentence pairs, and it is trained in an unsupervised way, by means of an approach similar to the Inside-Outside algorithm (Manning and Schütze, 1999). Another related proposal is Lynx (Venable, 2001). Like grammatical trigrams, Lynx are probabilistic models based on link grammars (Sleator and Temperley, 1993, 1991). Eisner (1996), in his model C, uses a dependency grammar, with unlabeled links (as opposed to the labeled links or connectors representing grammatical relationships between words of the link grammars). Carroll and Charniak (1992) focus on dependency grammars as well. They define an inductive algorithm to create the grammar, which performs incrementally: a new rule is introduced only if one of the sentences in the learning corpus is not correctly analyzed by means of the current rule set.

Head Automaton Grammars. Alshawi (1996) describes lexicalized head automata, a formalism representing parse trees by means of head-modifier relations. For each head, a sequence of left and right modifier words is defined together with their corresponding relations. A head automaton grammar (HAG), is defined as a function that defines a head automaton for each element of its (finite) domain. A head automaton is an acceptor for a language of string pairs $\langle x, y \rangle$ (the left and right modifiers), so that the language generated by the entire grammar is defined by expanding the special start symbol \$ into $x\$y$ for some $\langle x, y \rangle$, and then recursively expanding the words in strings x and y. A generative probability model is provided (Alshawi describes five parameter types), as well as a parsing algorithm which is analogous to the CKY algorithm (with a cost of $O(n^5)$). Eisner and Satta (1999) provide a translation from head automaton grammars to bilexical CFGs, obtaining a parsing algorithm for HAGs performing in time $O(n^4)$. Moreover, if the HAGs belong to the particular subclass of split head automaton grammars, a $O(n^3)$ parsing algorithm is provided.

Eisner (2000) introduces weighted bilexical grammars, a formalism derived from dependency grammars which can be considered a particular case of head automaton grammars. Weighted bilexical grammars extend the idea of bilexical grammars so that, instead of capturing black-and-white selection restrictions (say, either a certain verb subcategorizes a certain noun or not), gradient selection restrictions are captured: each specific word is equipped with a probability distribution over possible dependents. Then, the task of the parser will be to find the highest-weighted grammatical dependency tree given an input sentence. A new parsing algorithm for bilexical grammars (a variant of the one described in (Eisner, 1996)) is introduced, improving performance

with respect to the previous and usually used version. This work also shows how the formalism can be used to model other bilexical approaches. Bilexical grammars are very important in this thesis; many of the grammars we build can be seen as bilexical grammars. In Section 4.2.1, we show that bilexical grammars are a subclass of PCW-grammars.

(Lexicalized) Tree Adjoining Grammars. Lexicalized tree adjoining grammars (or LTAGs, for short) present an example of a lexicalized probabilistic formalism. They are an extension of tree adjoining grammars (TAG) (Joshi, 1987)), for which a probabilistic model was devised in (Resnik, 1992). In LTAGs, each elementary structure has a lexical item on its frontier, the anchor. Schabes (1992) describes a very similar probabilistic model, and derives an unsupervised version of the inside-outside algorithm to deal with stochastic TAGs. The main difficulty lies in defining the initial grammar rules. Joshi and Srinivas (1994) use n-gram statistics in order to find an elemental structure for each lexical item. Then, richer structures can be attached to lexical items, creating supertags, so that each elementary tree corresponds to a supertag, which combines both phrase structure information and dependency information in a single representation.

The disambiguation performed by supertags can be regarded as a preliminary syntactic parse (almost-parsing), which filters an important number of elementary trees before the conventional steps of combining of trees by means of adjunction and substitution operations. Srinivas (1997) gives additional models and results. It is not our intention to provide full details about the extensive literature on this formalism, but we will add some pointers about some aspects of TAGs that are specially interesting in relation to our own work. Nederhof et al. (1998) propose an algorithm for efficiently computing prefix probabilities for a stochastic TAG. Satta (1998) provides an excellent review of techniques for recognition and parsing for TAGs. Eisner and Satta (1999) describe a proposal of a more efficient algorithm for parsing LTAGs. Xia et al. (2001) describe a methodology to extract LTAG grammars from annotated corpora, and Sarkar (2001) explores state-of-the-art machine learning techniques to enable statistical parsers to take advantage of unlabeled data, by exploiting the representation of stochastic TAGs to view parsing as a classification task. Emphasis is given to the use of lexicalized elementary trees and the recovery of the best derivation for a given sentence rather than the best parse tree.

Lexicalized Context Free Grammars. Eisner and Satta (1999) define a bilexical context free grammar as a CFG in which every non-terminal is lexicalized at some terminal symbol (its lexical head), which is percolated from the constituent's head child in the parse tree. Such grammars have the obvious advantages of encoding lexically

specific preferences and controlling word selection, at the cost of a significant increment in size; the number of rules grows at a rate of the square of the size of the terminal vocabulary. As a consequence, the increment in the grammar size makes standard context free grammar parsers quite inefficient. For example, CKY-based variants perform at $O(n^5)$. Eisner and Satta (1999) present a $O(n^4)$ recognition algorithm for bilexical CFGs (in CNF), plus an improved version which, while having the same asymptotic complexity, is often faster in practice. By recursively reconstructing the highest probability derivation for every item at the end of the parse, this algorithm can be straightforwardly converted into an algorithm capable of recognizing stochastic bilexical CFGs, where each lexicalized non-terminal has attached a probability distribution over all productions with the same non-terminal as a left-hand side.

Satta (2000) defines lexicalized context free grammars (LCFG) as CFGs in which every non-terminal is lexicalized at one or more terminal symbols, which are percolated from the non-terminals in the production right-hand side. Then, the degree of lexicalization of a LCFG can be defined, so that bilexical CFGs have a degree of lexicalization of 2. Their major limitation is that they cannot capture relationships involving lexical items outside the actual constituent, in contrast with history-based models.

2.5.2 Other Formalisms

Stochastic Unification Formalisms. Brew (1995) presents a stochastic version of the head-driven phrase structure grammar (HPSG) formalism which allows one to assign probabilities to type-hierarchies. Re-entrancy poses a problem: in some cases, even if two features have been constrained to the same value by unification, the probabilities of their productions are assumed to be independent. The resulting probability distribution is then normalized so that probabilities sum to one, which leads to problems with grammar induction, as pointed out by Abney (1997). This latter work defines stochastic attribute-value grammars, shows why one cannot directly transfer context free grammar methods to the attribute-value grammar case (which is essentially what was done in (Brew, 1995)) and gives an adequate algorithm for computing the maximum-likelihood estimate of their parameters using Monte Carlo sampling techniques, although it is yet unclear whether this algorithm is actually practicable, due to its computational costs. Johnson et al. (1999) argue that this algorithm cannot be used for realistic-size grammars, and instead propose two methods based on a different type of log-linear model, Markov random fields. They apply these algorithms to the estimation of the parameters of a stochastic version of a lexical-functional grammar.

Data Oriented Parsing. Bod (1995)'s approach is different from other stochastic approaches in that it skips the step of inducting a stochastic grammar from a corpus.

Instead of a grammar, the parser uses a corpus annotated with syntactic information, so that all fragments (i.e., subtrees) in this manually annotated corpus, regardless of size and lexicalization, are considered as rules of a probabilistic grammar. The underlying formalism in DOP is called *stochastic tree substitution grammars* (STSG). In Section 4.2.3, we show that STSGs are a subclass of PCW-grammars.

For the time being, we will describe an STSG as a device that constructs the entire tree for an input sentence as a combination of tree fragments, in such a way that the product of the probabilities is maximal. During the training procedure, a parameter is explicitly estimated for each sub-tree. Calculating the score for a parse in principle requires summing over an exponential number of derivations underlying a tree, which in practice is approximated by sampling a sufficiently large number of random parsing derivations from a forest, using Monte Carlo techniques.

Markovian Rules. Markovian rules have been successfully used for natural language parsing. The methodology followed by a Markovian rule consists in attaching headwords to each syntactic category in the parse tree, to incorporate lexical probabilities into a stochastic model. Markovian rules are studied in detail in Section 4.2.2.

A remarkable and highly popular parser that uses Markovian rules as a component is Collins's parser. Initially described in (Collins, 1996), it was improved in (Collins, 1997), and fully described in (Collins, 1999; Bikel, 2004). Collins uses a supervised learning approach, with the PTB as a knowledge source, for estimating the parameters of his model. The key of his proposal is a well motivated trade-off between the expressiveness of the statistical model and the independence assumptions that must be made for assuring a sound estimation of the parameters given the corpus. In the model, a parse tree is represented as a sequence of decisions corresponding to a head-centered top-down derivation of the tree. Independence assumptions are linguistically motivated and encoded in the X-bar schema, subcategorization preferences, ordering of the complements, placement of adjuncts, and lexical dependencies, among others. All these preferences are expressed by means of probabilities conditioned on lexical heads. The generative model involves the estimation of the probability of each rule from the PTB, i.e., the probability of generating the right part conditioned on the left part. Collins decomposes this probability into three factors: (1) accounting for the probability of generating the head H, (2) given the parent, the probability of generating the components to the left, and (3) the probability of generating the components to the right. Independence assumption is introduced in order to make the model feasible. This basic model is further extended by introducing distances (taking into account some idiosyncratic features). The parser was trained with the PTB. (Charniak, 1997) presents a similar proposal which combines head word bigram statistics with a PCFG. The system adds a new useful statistic to guide the parser decisions: the type of the par-

ent will also condition the probability of a rule. When parsing a sentence, the system makes no attempt to find all possible parses, but it uses techniques presented in (Caraballo and Charniak, 1998) to select constituents that promise to contribute to the most probable parses (according to the simple probabilistic CFG distribution). However, as the probability distribution is different, these techniques allow us to ignore improbable parses. Moreover, the resulting chart contains the constituents along with information on how they combine to form parses. The constituents are assigned the probability given the lexicalized model, and the parser returns the parse with the overall highest probability according to this full distribution.

All these formalisms give a clear picture of all the grammatical formalism that were defined and used for language modeling. In Chapter 4 we present another formalism. The relations between our formalism and the most representative formalism presented above are presented in Chapter 4.

2.6 Approaches Based on Machine Learning

The approaches to language modeling presented in the previous section are mainly based on a formal grammatical device, i.e., most of them have some kind of grammars that generates and accepts sentences; the syntactical analysis of a sentence is the result of such a derivation. Clearly, there exists a wide variety of approaches based on what is commonly know as *machine learning techniques*. Under this perspective language modeling is treated as a pattern recognition problem, and hence, not necessarily related to the theory of formal languages. In this section, we present the most representative approaches. It is also difficult to draw hard conclusions on separation line between the "formal" approaches and the "machine learning" approaches because usually approaches combine the two. In this section we group models whose underlying formalism is very simple, e.g., context free grammar, and that heavily rely on machine learning (ML) techniques for achieving good performance in parsing.

Increase Context Sensitivity. The systems Pearl (Magerman and Marcus, 1991) and Picky (Magerman and Weir, 1992) use context-sensitive derivation probabilities. The basic idea is to try and maximize the probability of a correct derivation for each of the sentences in the corpus (as opposed to the inside-outside idea of maximizing the addition of the probabilities of the sentences in the corpus given a grammar). In Pearl, for instance, the application probability of a rule is modeled as a conditional probability, conditioned on the context in which the mother category appears. A chart parser (PUNDIT) is employed, and probabilities are estimated by simply counting the applications of the rules in the ATIS portion of the PTB.

History Based Grammars. Black et al. (1993) present a more general framework called *history based grammars*. In this system, the term history is equivalent to context: the application of a rule is conditioned on arbitrary aspects of the context of the parse tree (the context information being both the dominating production and the syntactic and semantic categories of the words in the prior derivations. In his model, decision trees (see (Jelinek et al., 1994)), are trained from a treebank (computer manuals domain) and they are used to score the different derivations of sentences produced by a hand-written broad-coverage feature-based unification grammar (672 rules, 21 features).

Hermjakob and Mooney (1997) present a knowledge and context-based system (CONTEX) which, applying machine learning techniques, uses examples to generate a deterministic shift-reduce parser. The learning algorithm uses an extended version of the standard ID3 model (Mitchell, 1997) for more general hybrid decision structures, in combination with decision lists; it starts by assigning to each parse tree from the training corpus a sequence of shift-reduce parsing operations needed to produce the tree. In order to learn the specific action to be performed at any point of the derivation, the system relies heavily on an enriched context (to the left and right of each word), encoded in features which include morphological, syntactic, and semantic information (the previously built structure, a subcategorization table, and a knowledge base with semantic information about the words in the lexicon. The methodology is evaluated on a subset of sentences from the WSJ (only the ones fully covered by the 3000 most frequent words in the corpus).

Probabilistic LR Parsing. The standard LR parsing methodology performs a left-to-right scan of the input and constructs a rightmost derivation in reverse. Ng and Tomita (1991) extend the well-known generalized LR parsing algorithm from (Tomita, 1996) by attaching probabilities to the nodes of the graph structured stack which constitutes the kernel of the algorithm. Part of their proposal deals with how to consistently maintain these probabilities (initially derived from the probabilities attached to the rules of the PCFG) considering the three operations of the graph-structured stack (merging, local ambiguity packing, and splitting). However, it is not possible to use an algorithm like Viterbi in order to compute the most probable parse.

Other LR parsing approaches use PCFGs as a source including (Wright, 1990; Wright and Wrigley, 1989; Wright et al., 1991). In all of them, an LR parse table is derived from the context free grammar but, in addition, the rule probabilities are distributed among sets of actions in the LR table. The distribution is carried out so that it can be assured that the product of the probabilities associated to those LR actions performed in the derivation of any analysis will be exactly the same as the probability which would have been assigned to this analysis by the PCFG.

Carrol (1993) discusses the latter as well as other methodologies, and presents, together with Ted Briscoe, a more ambitious proposal. They start from a unification grammar (the ANLT grammar), from which a context free backbone grammar is automatically derived, together with an associated residue containing the dependencies between features and values not contained in the context free grammar. The parser must associate the reduce operations of the LR table with a filter based on the unification of the features contained in the residue. The backbone grammar generated from the ANLT grammar had 575 categories and more than 2,000 productions, and an LR parse table was automatically generated for this grammar. Unlike (Ng and Tomita, 1991), the probabilistic model consists in attaching probabilities not to the context free rules, but to the actions in the LR table. The model is then more context sensitive. In the experiments described, the learning is supervised, the training corpus consisting of a set of LR parse histories (with human intervention to correct the transition in the LR parse table). Inui et al. (1998) build on Bricoe and Carrol's work, and improve it by formalizing their model in such a way that it provides probabilistically well-founded distributions. Although they focus on the formal and qualitative aspects of the model, they show how their refinement is expected to improve parsing performance. It is worth noting that recent work by Nederhof and Satta (2002), which investigates the problem of extending parsing strategies to probabilistic parsing strategies. They conclude that LR parsing cannot be extended to become a probabilistic parsing strategy, because it lacks the property denoted as strong predictiveness property (SPP). In other words, probabilistic LR parsing algorithms might not preserve all the properties of the PCFG probability distributions, which means that LR parsers may sometimes lead to less accurate models than the grammars from which they are constructed.

Transformation Based (Error-Driven) Learning (TBL). Brill (1993) has applied TBL to grammar induction and parsing. The approach consists in learning a ranked list of transformational rules so that, starting from an initial imperfect binary right-branching tree for a sentence, the sequential application of each rule may transform a piece of the original tree, and in the end obtain a parse tree with fewer errors. The firing of each rule is conditioned on a context of one or two tags, so that the learning process (performed through a greedy search according to the largest error decrease criterion) needs quite a few number of sentences (150 /250 sentences for the ATIS and WSJ corpora) for obtaining the same accuracy of contemporary systems.

Instance-Based, Memory Based or Case-Based Learning. Instance-based algorithms (IBL) are a supervised way of inductively learning from examples, that are taken into account in order to classify new examples by analogy (the most similar instances are retrieved from memory, and used for extrapolation). Memory-based learning is a

direct descendant of the classical k-NN (k nearest neighbor) approach to classification. Simmons and Yu (1992) apply the idea to a context sensitive shift reduce (SR) parser. SR parsing is suitable for this classification proposal, since it breaks the parsing process into simple parse actions (shift, reduce, and fin), allowing the construction of an example base of parse states with their correct parse actions. A parse action is assigned to each parse state on the basis of the on information of the parse stack and the input buffer. The parser works on the level of POS tags and windows over the text with a context of five words to the left and to the right.

The ILK Group at Tilburg University has developed the TiMBL (Tilburg memory-based learning) environment, a general instance-based algorithm which compresses of the base of examples into a tree-based structure, the IGTree (see (Daelemans et al., 1997)), which in turn is used to classify new examples. The memory-based algorithms implemented in the TiMBL package have been successfully applied to a large range of NLP tasks, including shallow parsing (see (Daelemans et al., 1999)) and more recently, full parsing: Veenstra and Daelemans (2000). They construct a memory-based shift reduce parser, inspired by (Simmons and Yu, 1992). Cardie (1993a) addresses the lexical, semantic, and structural disambiguation of full sentences, within an information extraction environment. In a supervised training phase, the parser creates a case base of domain-specific context-sensitive word definitions. Then, given an unknown word and the context in which it occurs, an eventual robust parser could retrieve the definitions from the case base in order to infer the necessary syntactic and semantic features for the unknown word and then continue processing the text. The case retrieval algorithm is basically a k-NN algorithm, but it assumes all features are equally important for learning each type of knowledge, which, intuitively does not seem to be true. Therefore, the system takes advantage of decision trees for identifying the relevant features to be included in the k-NN case retrieval; the approach is fully described in (Cardie, 1993b).

Decision Tree Models. Magerman (1995b) has been a pioneer in the use of decision trees for syntactic parsing: he explores a wide variety of possible conditioning information and uses a decision-tree learning scheme to pick those analyses that seem to give the most purchase. Three different decision-tree models are used for (1) the POS tagging, (2) the node expansion, and (3) the node labeling. The decisions are based on lexical and contextual information of the parent and the child of the node.

Probabilistic Feature Grammars. Goodman (1997, 1998) presents probabilistic feature grammars in which each non-terminal is represented as a vector of feature-value pairs. Then, assuming binary-branching rules, the probability of the application of a rule can be decomposed as the incremental prediction of the feature values of each

of the two members of its right-hand side. As all conditioning variables are encoded through features, different factors such as lexical dependencies or distance features can be dealt with in a unified way. Probabilistic feature grammars put the emphasis on parameter estimation: having chosen the features, the parameters of the model are specified by choosing an order for the features being predicted and then applying the independence assumptions and choosing a back-off order for smoothing. The model is tested on the WSJ portion of the PTB, where the following features are considered: the non-terminal label, the headword, the head POS, distance features, and additional context (modifier non-terminals generated at earlier stages of the derivation).

Maximum Entropy Models. The use of maximum entropy (ME) models, has become very popular lately in various areas of NLP. Rosenfeld (1994) applied it to speech recognition tasks and Berger et al. (1996) to automatic translation, Ratnaparkhi (1998) applies it to several tasks: segmentation, morpho-syntactic disambiguation, PP-attachment, and syntactic parsing. Ratnaparkhi (1999) describes the latter application, which is also an example of lexicalized parser. Maximum entropy (ME) models overcome the limitations of independence among the variables. Without the need for an explicit grammar, they can learn, from a labeled set of examples, the model which has maximum entropy out of all the models compatible with this set of examples. In other words, given a collection of facts, ME models choose a model which is consistent with all the facts, but otherwise as uniform as possible. The basic element of any ME model are the features, binary-valued functions with two parameters, a context and an output.

Ratnaparkhi trains his system on a set of templates that are attached to each of the parsing procedures. These templates incorporate the type of factors the author considers relevant for the analysis: constituent headwords, headword combinations, generalizations (morpho-syntactic categories, constituent syntactic categories), and limited forms of look-ahead. The learning process is very simple, it is just counting, so that the features that appear less than 5 times in the corpus are rejected. Using 40,000 sentences from the PTB corpus, 1,060,000 features are incorporated in the model (most of them lexicalized), and each one is attached to one of the procedures.

Charniak (2000) presents a parser based upon a probabilistic generative model, an extension of the ones described above (Charniak, 1997; Collins, 1997). The probabilistic model is maximum-entropy-inspired, since it reformulates the basic maximum entropy probability function such that it considers the conditioning information of Markov grammar statistics as features. Moreover it is ultimately smoothed by means of deleted interpolation (instead of the standard feature selection of pure ME models). Charniak (1997) uses his bottom-up best-first chart parser to generate the candidate parses, and his top-down generative model to evaluate them (in a process which, for each constituent, first guesses its preterminal, then its lexical head, and last its expan-

sion into further constituents).

Other ML-based Models. The idea in explanation-based learning (EBL; (Rayner and Cater, 1996)) is that some of the grammar rules (specially in specific domains) tend to combine much more frequently in a certain way than others. Given a sufficiently large corpus parsed by the original (general) grammar, it is possible to learn the common combinations of rules and chunk them into macrorules; Samuelsson (1994), for example, defines an entropy threshold for automatically deriving these macrorules. The result is a specialized grammar, with a larger number of rules but with a simpler structure. In practice, parsing is shown to be faster — 3 to 4 times speed up for an LR parser — at a price of only 5% coverage loss, using a training corpus of a few thousand utterances. Zelle and Mooney (1996) describe a methodology to automate the construction of parsers based on another ML-based learning methodology, inductive logic programming (ILP). They have developed a system, CHILL, which begins with a well-defined parsing framework, shift-reduce parsing, and uses ILP to learn control strategies within this framework, inductively learning a deterministic shift-reduce Prolog parser that maps sentences into parses. CHILL represents a highly flexible application of ILP, allowing the induction over unbounded lists, stacks, and trees. They describe the application of the system to the automatic induction of parses that map natural language database queries into an executable logical form. In addition, there has been research using both neural networks and symbolic induction to learn parsers that produce case-role analyses (Miikkulainen, 1996). In NLP, neural networks have mostly been used basically to address low-level problems, although there are examples of applications to more complex problems such as parsing (sometimes in combination with symbolic approaches such as the above mentioned example).

Parser Combination and Reranking. A methodology for combining three input parsers in order to improve parsing results is described by Henderson and Brill (1999). The three parsers combined are the systems described in (Collins, 1997; Charniak, 1997; Ratnaparkhi, 1997). The two techniques used for combining parsers are parser hybridization and parser switching. The first one is based on combining the substructures of the three input parsers in order to produce a better parse.

Two hybridization strategies are used, namely constituent voting (a non-parametric version where the parsers vote on the membership of a certain constituent to the final parse) and naïve Bayes classifiers. The second technique, parser switching, chooses among entire candidate parsers. Again, two strategies are tested, a non-parametric version and a parametric version (naïve Bayes again). Experiments on the WSJ portion of the Penn Treebank show that all the combining techniques accomplish better accuracy than any of the single three parsers, and that the method is robust, as incorporating of

a poorly performing parser (a nonlexicalized PCFG parser) hardly affects the results.

Collins (2000) proposes two machine-learning methodologies for reranking the output of a given probabilistic parser. The idea is that in a first step the base parser returns a set of candidate parses (initially ranked according to the probabilities the parser has attached to them), and then a second step tries to improve this ranking, considering additional features of the trees. Both approaches are discriminative, since they aim at optimizing a criterion which is directly related to error rate. The first reranking technique in based on a generalization of PCFGs, Markov random fields (Abney, 1997), while the second technique is based on boosting of ranking techniques (Schapire and Singer, 1999) (here the ranking is a simple binary distinction between the highest scoring parse and the other ones). The methodology was evaluated on the PTB, including features ranging from rules or bigrams (pairs of non-terminals to the left and right of the rules head), to features involving the distance between head words. The first approach was too inefficient to run on the full data set, so only the boosting approach could be evaluated.

Collins (2001) gets more deeply into the differences between parametric maximum likelihood estimation methods (explicitly modeling the distributions) and distribution-free methods (models assuming that the training and test examples are generated from the same distribution, although it is unknown, the results hold across all distributions). Two methods are proposed: the first one, as in (Collins, 2000), is an application of the Adaboost algorithm to re-rank the output of an existing parser, while the second one uses the perceptron or support vector machines (SVM) algorithms. This second method is based on the representation of parse trees through tree kernels (a mechanism allowing one to convert them into efficiently treatable high-dimensional feature spaces). It is described in more detail in (Collins and Duffy, 2001), and the voted perceptron is applied on the ATIS portion of the PTB, for reranking the results of a PCFG. Collins and Duffy (2002) extend the results to the WSJ portion of the PTB, starting from the parses produced by model 2 of (Collins, 1999). The tree kernel allows the representation of all subtrees in the training data (the same representation used by DOP), so that the perceptron algorithm uses both the result from the base model as well as the subtrees information to rank the trees. The method accomplishes improvements of 0.5% and 0.6% respectively in labeled precision and recall with respect to the base model.

Carreras et al. (2002) present an approach to partial parsing (though potentially applicable to full parsing) which is based on (1) using local classifiers to recognize partial parsing patterns, and (2) using global inference methods to combine the results of these classifiers in a way that provides a coherent inference that satisfies some global constraints. Although such ensembles of classifier techniques had already been explored (see for instance (Punyakanok and Roth, 2000)), this work applies it to a deeper and

more difficult level of partial parsing, embedded clause identification. This way, the best decomposition of a sentence into clauses is selected by means of a dynamic programming scheme which considers previously identified partial solutions, and applies learning at several levels (for detecting beginnings and ends of potential clauses and for scoring partial solutions, including three different scoring functions). The Adaboost algorithm with confidence rated predictions (see (Schapire and Singer, 1999) is used as learning method. The approach is evaluated using the CoNLL-2001 competition corpus (Tjong Kim Sang and Déjean, 2001), outperforming the best system presented in this competition.

Summing up, this chapter shows that there exists a wide variety of approaches and formalisms that focus on natural language syntax. The classification we presented tries to separate the "formal" approaches from the "machine learning" approaches. Clearly, it is hard to classify parsing approaches into these two categories because they usually combine the two approaches. In the rest of the thesis we advocate an approach to natural language parsing that is based on the machinery provided by formal languages. Machine learning techniques are not used for the language modeling task, but they are used for the inference of the formal devices. The distinction is small but important. It is like using a PCFG as a language model and using machine learning for learning the PCFG itself.

Chapter 3

The Role of Probabilities in Probabilistic Context Free Grammars

3.1 Introduction

Probabilities have been used in many aspects of natural language processing. In the context of CFGs, probabilities have been used to define a probability distribution over the set of trees defined by a CFG. The resulting formalism, probabilistic context free grammars (PCFGs), extends context free grammars by assigning probabilities to the production rules of the grammar. PCFGs have been successfully used as the formalism underlying many approaches to natural language parsing, see e.g., (Eisner, 1996; Charniak, 1995; Collins, 1999; Eisner, 2000; Klein and Manning, 2003), to name just a few. In such approaches, probabilities have a very specific role: they are used as a filtering mechanism. In order to clarify this, we can think of the parsing procedure for a PCFG as a two-fold procedure. Suppose a sentence is given. First, a set of candidate trees is proposed, and, second, a tree is non-deterministically chosen from the set of candidate trees. The selected tree is returned as output.

CFGs define the set of candidate trees as the set containing all trees that yield the given sentence. In contrast, PCFGs are used in the parsing context to define the set of candidate trees as the set of trees that yield the given sentence with maximum probability. In other words, probabilities are used in the parsing framework to reduce the set of candidate trees suggested by the bare CFG. The size of the set of candidate trees gives an impression of how ambiguous a sentence is for a given input. It is well-known that ambiguity is a serious problem for parsers; in some cases, the number of parse trees assigned to a sentence may grow exponentially in the length of the sentence (Wich, 2000, 2001), and, consequently, probabilities play a very important role in parsing: they are used to decrease the size of the set of candidate trees, and consequently, they

decrease the amount of ambiguity.

The role probabilities play in the context of formal languages has been widely studied in the literature, but the focus of such studies was not on the ability to reduce ambiguity. Properties of formal languages regarding consistency (Booth and Thompson, 1973; Wetherell, 1980; Chaudhuri and Rao, 1986), learnability conditions (Horning, 1969), parameter estimation (Manning and Schütze, 1999), etc, are very well-known. However, very little is known about the power of probabilities as a mechanism for ambiguity reduction.

We are interested in the following research issues: how important and how powerful are probabilities as a filtering mechanism? Clearly, different filtering mechanisms exist; as an example, suppose that a PCFG G is given and its probabilities are used to select a subset T' of trees in the tree language generated by G. It may be the case that there exists a non-probabilistic CFG G' such that its tree language is equal to T. Clearly, if such a grammar G' exists for all PCFG G's, then probabilities are not really needed. In contrast, if that is not the case, the role probabilities play in parsing becomes relevant, and different questions than those answered in (Booth and Thompson, 1973; Wetherell, 1980; Chaudhuri and Rao, 1986) need to be addressed.

In this chapter we focus on answers to basic questions regarding the use of probabilities as a filtering mechanism. We pay special attention to the following questions:

1. Is it possible to select the same set of candidate trees as with a PCFG using a vanilla context free grammar? The question is relevant because, if the answer is affirmative, this means that for a given PCFG there is a CFG that specifies the same set of candidate trees as the PCFG for all sentences; as a consequence probabilities would not really be essential.

2. Can we decide whether a given PCFG filters out all but one tree for all sentences in the language? Answering this question is equivalent to saying that the given PCFG has solved all ambiguities in the language accepted by the grammar.

3. What is the meaning of the probabilities associated with the set of candidate trees? Can this meaning be used in a useful manner?

In this chapter, we answer all these questions. We show

1. that probabilities cannot be mimicked by rules, i.e., their use is fundamental from a formal language perspective: whenever used as a filtering mechanism, probabilities can define a set of trees *not* capturable by CFGs,

2. that it is not possible to decide whether a PCFG filters all trees but one for all sentences, i.e., it is not possible to decide whether the filtering mechanism resolves all ambiguities,

3. that the probability measure assigned to the selected set of candidate trees can be interpreted as the probability of having captured in the set of candidate trees the correct tree. Moreover, this semantics can be used to asses the quality of PCFGs. We also show that under certain circumstances, which we describe in detail, it is advisable to add to the set of candidate trees, trees that do not have maximum probability in order to augment the probability of capturing the correct tree.

The chapter is organized as follows. Section 3.2 presents some background notation, Section 3.3, 3.4, and 3.5 address questions 1, 2, and 3 listed above respectively. Finally, Section 3.6 concludes the chapter.

3.2 Maximum Probability Tree Grammars

In this section we define the basic concepts we need in our discussion. Let $G = (T, NT, S, R)$ be a context free grammar. For our purposes, CFGs play two very important roles. The first one is to provide sentences with a syntactic explanation. As is well-known, this syntactic explanation is given by the way in which a sentence is rewritten from the start symbol. Formally, let G be a CFG and x a sentence in T^*. A *left-derivation* $t(x)$ is a sequence of rules $\langle r_1, \ldots, r_m \rangle$ such that

$$S \stackrel{r_1}{\Rightarrow} \alpha_1 \stackrel{r_2}{\Rightarrow} \alpha_2 \stackrel{r_3}{\Rightarrow} \ldots \stackrel{r_m}{\Rightarrow} \alpha_m,$$

where $\alpha_i \in (T \cup NT)^*$, $\alpha_m = x$, and r_i is a rule in R that rewrites the left-most non-terminal of α_{i-1}. The set of all left-most derivations, called the *tree language*, is denoted by $T(G)$, and x is called the *yield* of $t(x)$.

The second main role of CFGs is to define the set of sentences that are considered to be grammatical by the grammar. More precisely, let G be a CFG. The *language accepted* by G (notation: $L(G)$) is $L(G) = \{x : T(x) \in T(G)\}$.

In order to use CFGs to help us understand natural language, a procedure for finding derivations has to be defined. Let G be a CFG and x a sentence. A *complete parser* is a procedure that computes the following set:

$$T(x) = \{t(x) \in T(G)\},$$

while a parser is a procedure that, besides computing $T(x)$, chooses one tree non-deterministically from it. Formally,

$$Parser(x) = random(T(x))$$

where $random(X)$ is a function that selects an element from the set X assigning to

each element the same probability of being chosen.[1] For a given grammar G and a sentence x in $L(G)$ there may be multiple trees yielding the same sentence x.

In some cases the size of the set $T(x)$ grows exponentially in the length of x (see (Wich, 2000) for an example), and many of the trees in $T(x)$ are trees that we do not want as an answer. These undesired trees need to be filtered out from the set $T(x)$. One way to achieve this is to use probabilistic context free grammars. A *probabilistic context free grammar* (PCFG) is a pair (G, p), where G is a CFG and p a positive function defined over the set of rules such that for all A in NT:

$$\sum_{\alpha \in (N \cup NT)^*} p(A \rightarrow \alpha) = 1.$$

Even though probabilities in a PCFG are defined over the set of rules, they are used to define a probability distribution p over the set of derivations $T(G)$. The probability of a tree in $T(G)$ is defined as follows. Let (G, p) be a PCFG, and let $t(x) = \langle r_1, \ldots, r_m \rangle$ be a tree in $T(G)$. The *probability* assigned to $t(x)$ is $p(t(x)) = p(r_1) \cdot \ldots \cdot p(r_m)$. In words, the probability assigned to a tree is the product of the probabilities assigned to the rules building up the tree.

The distribution generated by the probabilities is used to select a subset of trees from the set of all possible trees yielding a sentence. A procedure that computes this subset is called a probabilistic parser. More formally, let G be a PCFG and let x be a sentence in $L(G)$. A *probabilistic complete parser* for a grammar G is a procedure that takes as input a sentence in the language and computes the following function:

$$PParser(x) = \{t(x) \in T(G) : \arg\max_{t(x)} \{p(t(x))\}\}.$$

Finally, a *probabilistic parser* is defined as the non-deterministic choice of a tree returned by a probabilistic complete parser. We can see a probabilistic parser as a two component algorithm, one implementing a probabilistic complete parser and a second implementing a non-deterministic choice.

3.2.1 Filtering Trees Using Probabilities

In this subsection, we consider the class of languages induced by probabilities when we consider them as a filtering mechanism. Let G be a PCFG. The set of *most probable trees* produced by G (noted $M(G)$) is a subset of $T(G)$ defined as follows:

$$M(G) = \bigcup_{x \in L(G)} PParser(x).$$

[1] Note that non-deterministic choice and uniformly distributed choice are not the same concept, non-determinism implies that there is no information about the underlying distribution, which clearly is not the case when using a uniform distribution. In the context of this thesis we use the two terms indistinguishably.

Note that there may be more than one tree bearing maximum probability for a given sentence. We allow for this: $M(G)$ contains them all. Note also that $M(G)$ is a subset of $T(G)$. Furthermore, the sets $M(G)$ and $T(G)$ are the same set if, and only if, for all sentences all trees in $L(G)$ share the same probability mass.

Based on the set of most probable trees $M(G)$, we define a new class of tree languages. The idea behind this class of languages is that they are like PCFGs but instead of taking the whole set of trees they only take the most probable ones. More formally, a *maximum probability tree grammar* (MPTG) is a PCFG where its tree language is defined as the set of most probable trees, while the set of strings accepted by the MTPG remains the same.

Note that all state-of-the-art parsers based on PCFG filter trees out in the way we have just defined. They return the trees yielding a given sentence that bear maximum probability, and thus, they, implicitly define an MPTG.

3.3 Expressive Power

We are interested in understanding what kind of expressiveness MPTGs have. In particular, can they be captured by CFGs? More concretely, does the mechanism of retaining only trees with maximum probability defined in MPTGs define tree languages that cannot be captured by CFGs? In this section we show that the set of trees identified by a probabilistic CFG plus a maximization procedure cannot be generated or specified directly by any CFG. In other words, we prove that probabilistic CFGs plus a maximization step define tree languages that are beyond the expressive power of the CFGs. To put it more formally: MPTGs are not strongly equivalent to CFGs. That is, the tree language generated by probabilistic context free grammars plus a maximization algorithm is beyond context free grammars. This section is devoted to prove this statement, which is formally expressed in the following theorem.

3.3.1. THEOREM. *MPTGs are not strongly equivalent to CFGs.*

Our strategy for proving the above theorem is based on a context free inherently ambiguous language. Recall that a context free inherently ambiguous language L is a language such that all CFGs generating it have at least one string in L that has two trees yielding it. What we present below is an inherently ambiguous language generated by a PCFG that assigns a unique tree bearing maximum probability to each string in the language. As a consequence, the MPTG induced by the grammar is unambiguous, but the tree language cannot be captured by a CFG because the language is inherently ambiguous, i.e., for all CFGs generating it, there is at least one sentence being yielded by two different trees. Let us make matters concrete now, the grammar $G = (T, NT, S, R)$ we

$$\begin{array}{ll} S \rightarrow_{2/3} AB & S \rightarrow_{1/3} S_2 \\ A \rightarrow_{1/2} aAb & S_2 \rightarrow_{1/2} aS_2d \\ A \rightarrow_{1/2} ab & S_2 \rightarrow_{1/2} aCd \\ B \rightarrow_{1/2} cBd & C \rightarrow_{1/2} bCc \\ B \rightarrow_{1/2} cd & C \rightarrow_{1/2} bc \end{array}$$

Figure 3.1: An MPTG for an inherently ambiguous language.

suggest is defined as follows. Put $T = \{a, b, c, d\}$, $NT = \{S, S_2, A, B, C\}$, and let the set of rules R be as described in Figure 3.1.

In order to better understand the complexity of G we split it into two different grammars $G_1 = (T_1, NT_1, S, R_1)$ and $G_2 = (T_2, NT_2, S, R_2)$, where

1. $T_1 = \{a, b, c, d\}$,
2. $NT_1 = \{S, A, B\}$,
3. $T_2 = \{a, b, c, d\}$, and
4. $NT_2 = \{S, S_2, C\}$;
5. R_1 and R_2 as defined by Figure 3.2.

Observe that G generates the following string language:

$$L = \{a^n b^n c^m d^m : n, m \in \mathbb{N}\} \cup \{a^n b^m c^m d^n : n, m \in \mathbb{N}\}.$$

L can be described as the union of the two context free languages generated by G_1 and G_2, respectively, namely $L_1 = \{a^n b^n c^m d^m : n, m \in \mathbb{N}\}$ and $L_2 = \{a^n b^m c^m d^n : n, m \in \mathbb{N}\}$.

We will now describe the tree language generated by G. Remember that G is a MPTG, and not just a PCFG. The set of its derivations is the set of trees bearing maximum probability. Define $L_3 = \{a^n b^n c^n d^n : n \in \mathbb{N}\}$, and let $T(L_3)$ denote the set

R_1	R_2
$S \rightarrow_{2/3} AB$	$S \rightarrow_{1/3} S_2$
$A \rightarrow_{1/2} aAb$	$S_2 \rightarrow_{1/2} aS_2d$
$A \rightarrow_{1/2} ab$	$S_2 \rightarrow_{1/2} aCd$
$B \rightarrow_{1/2} cBd$	$C \rightarrow_{1/2} bCc$
$B \rightarrow_{1/2} cd$	$C \rightarrow_{1/2} bc$

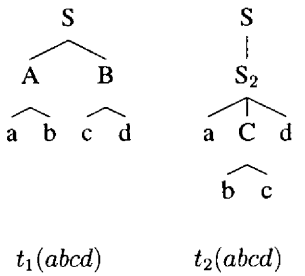
Figure 3.2: A decomposition of G.

of trees in $T(G)$ that yield a sentence in L_3. Our first observation is that all trees in $(T(G_1) \cup T(G_2)) \setminus T(L_3)$ have a unique derivation. Hence, they obviously belong to $M(G)$. For the trees in L_3 there are two possible derivations: one generated by G_1 and the other generated by G_2. In order to fully characterize $M(G)$, we need to determine which of the two derivations (if not both) belongs to $M(G)$.

Below, we show that only the derivations produced by G_1 belong to $M(G)$. In other words, all derivations produced by G_2 are filtered out. To obtain this characterization, we first characterize all trees in $T(L_3)$.

3.3.2. LEMMA. *Let x be a string in L_3, and let $t_1(x)$ be a tree in $T(G_1)$ and $t_2(x)$ a tree in $T(G_2)$. Then, the number of rules appearing in $t_1(x)$ is the same as the number of rules appearing in $t_2(x)$. Moreover, the rule $S \rightarrow_{2/3} AB$ appears once in $t_1(x)$, while the rule $S \rightarrow_{1/3} S_2$ appears once in $t_2(x)$.*

PROOF. Let x be a string in $L_3 = \{a^n b^n c^n d^n : n \in \mathbb{N}\}$. We prove the lemma by induction on n, the superscript in the definition of L_3. For the base case, let n be 1; then the lemma follows from the fact that the two possible trees, both pictured in Figure 3.3, have the same number of rules, and it is clear that the rule $S \rightarrow_{2/3} AB$ appears once in $t_1(x)$, while the rule $S \rightarrow_{1/3} S_2$ appears exactly once in $t_2(x)$.



$t_1(abcd)$ $t_2(abcd)$

Figure 3.3: Two derivations for *abcd*.

Let the statement be true for $k < n$, and let us show it for $x = a^n b^n c^n d^n$. Note, first of all, that for a word in L_3, the two possible trees follow the schema of Figure 3.4. Now, using this observation for the first part of the lemma, the string $a^{n-1} b^{n-1} c^{n-1} d^{n-1}$ can be derived by collapsing the A, B, S_2 and C non-terminals, respectively. According to the inductive hypothesis, the resulting trees have the same number of rules, and in the process of collapsing we have eliminated the same number of rules from the two trees, proving the first part of the lemma. For the second part of the lemma, consult Figure 3.4 again, where it can be seen that the rules $S \rightarrow_{2/3} AB$ and $S \rightarrow_{1/3} S_2$ appear once in $t_1(x)$ and $t_2(x)$, respectively. ⊣

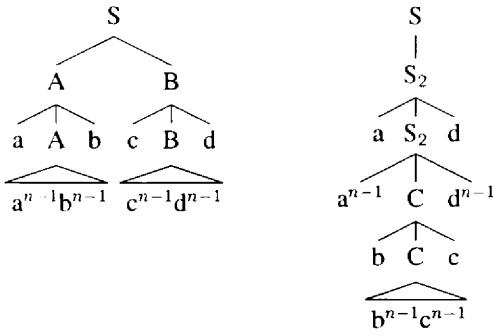


Figure 3.4: Derivations for $a^n b^n c^n d^n$.

Lemma 3.3.2 says that the probabilities of two derivations in L_3 yielding the same sentence are determined by the first rule in each grammar. That is, we can distinguish between the two possible probabilities assigned to the two derivations for a sentence in L_3 by simply observing the probability of the first rule in each of the derivations.

3.3.3. LEMMA. *Let x be a string in L_3, $t_1(x)$ in $T(G_1)$ and $t_2(x)$ in $T(G_2)$. Then $p(t_1(x)) > p(t_2(x))$.*

PROOF. The proof is immediate from Lemma 3.3.2. All derivations for a given string x in L_3 have the same number of rule applications, and except for the first rule applied (either $S \rightarrow_{2/3} AB$ or $S \rightarrow_{1/3} S_2$), all rules have equal probabilities associated with them. Hence the tree using the G_1-rule $S \rightarrow_{2/3} AB$ has the higher probability, as desired. ⊣

3.3.4. LEMMA. *$M(G)$ is equal to $T(G_1) \cup \{t(x) \in T(G_2) : x \in L_2 - L_3\}$.*

PROOF. The lemma is a direct consequence of Lemma 3.3.3. ⊣

Finally, with this characterization we can prove Theorem 3.3.1.

PROOF. Note first that for every string in $L(G)$ there is a unique tree in $M(G)$: if the string belongs to $L_1 \setminus L_3$ or to $L_2 \setminus L_3$ this is because the grammars G_1 and G_2 assign a unique derivation tree to each string. If, however, $x \in L_3$, then, by maximizing probabilities, we discard the tree belonging to $T(G_2)$, thus only leaving the tree in $T(G_1)$, as shown in Lemma 3.3.4. Summarizing, using probabilities we have obtained a unique tree for every sentence in an inherently ambiguous language. It is a well-known fact that CFGs cannot disambiguate an inherently ambiguous language (Hopcroft and Ullman, 1979). Hence, since L is inherently ambiguous (Parikh, 1966), there cannot be a CFG that generates all and only the trees in $MPTG(G)$. ⊣

In this section we have answered question (1) from the introduction, showing that there is no way to mimic probabilities using rules. In the next section we focus on question (2).

Ours is not the first study of the expressive power of weighted formal devices. Cortes and Mohri (2000) show that the expressive power of weighted automata is beyond regular languages. This result has in common with the result we present in Section 3.3 that they both show that weighted systems accept a wider set of languages than bare systems. The two results also use the same strategy; they present a language that does not belong to the bare grammatical formalism but that is capturable by the weighted version.

The two approaches differ in that in ours probabilities are used to select subtrees as a side product of filtering. Cortes and Mohri (2000) show that they found a well-known context free language to be accepted by a weighted automata under a general definition of acceptance. The approaches also differ in that ours is concerned with the tree-language and theirs with the string language. Finally, the two results differ in that our proof is technically trivial, while the other is rather involved.

The result presented in this section is not directly linked to statistics. As already discussed in (Abney, 1996), probabilities can help in many aspects of syntax (e.g., disambiguation, degrees of grammaticality, error tolerance, naturalness, structural preferences, learning, lexical acquisition). In that sense, weights are enough for capturing these phenomena. In this section we deal with probabilities (weights) as they are currently used for disambiguation in the literature. We respond to the question "What can we do with probabilities (weights)?" more than "How can we compute the probabilities (weights)?". Abney argues that, intuitively, probabilities can help disambiguating, but this section shows that they provide a mechanism that simply cannot be mimicked with rules. To put it very bluntly, we present a technical fact, not an intuition.

We have shown that probabilities add not only a statical perspective but also expressive power to CFGs. We think that this increment in the expressive power is due to a probabilistic parser's implicit but global requirement that it sees all rules building up a tree for choosing the one with maximum probability.

3.4 Undecidability

While probabilities buy us additional expressive power on top of CFGs, they do not buy us everything. Specifically, given that probabilities help to disambiguate their own language, would it not be nice if we could predict, that is, determine before parsing, whether a PCFG is capable of fully disambiguating its language?

We show that this is not possible. We establish the result by transforming an ar-

bitrary CFG into a PCFG such that the given CFG is unambiguous if, and only if, the corresponding PCFG has only one tree in the candidate list of each sentence. Our result then follows from the well-known fact that determining whether a CFG is unambiguous is undecidable.

Probabilities single out, for each sentence x, a set $PParser(x)$ of trees bearing maximum probability. An *ideal* grammar is one that filters out all trees but one for each sentence in the language. In other words, an ideal PCFG defines for each sentence x, its set $PParser(x)$ with cardinality equal to 1.

We want to prove that it is undecidable to determine whether a PCFG is ideal. To this end, we first prove that for every context free grammar G there is a way to extend it with probabilities such that the resulting set $M(G)$ contains the same set of trees as G. In other words, for any CFG we build a probabilistic version that does not filter out any tree. Our undecidability result follows from the fact that our question is equivalent to determining whether a CFG is unambiguous.

We have to build the probabilistic correlate of a CFG, in such a way that all trees associated to a given sentence bear the same probability. In this case, the set of trees with maximal probability is exactly the set of trees. We show the result for grammars in Chomsky Normal Form whose definition we now recall.

A context free grammar $G = (T, NT, R, S)$ is said to be in Chomsky Normal Form (CNF) if, and only if, every rule in R is of one of the following forms:

- $A \rightarrow a$ for some $A \in NT$ and some $a \in T$.
- $A \rightarrow BC$, for some $A \in NT$ and $B, C \in NT - \{S\}$.

Our strategy is to show that any grammar in CNF assigns the same probability to all trees yielding the same string. To this end we show that all trees yielding the same string in a CNF use the same number of rules; we then build a grammar assigning the same probability to all rules and we obtain what we are looking for.

We now present the lemmas needed.

3.4.1. LEMMA. *Let $G = (T, NT, S, R)$ be a grammar in CNF. All trees yielding a k-length sub-string of NT^* use the same number of rules.*

PROOF. Let us define a sequence $A_0, \ldots, A_n, \ldots$ of subsets of NT^* as follows: $A_0 = \{S\}$, A_1 consists of elements α in NT^* such that α is derived from S in one step, and, in general, α is in A_i if there is an element α' in A_{i-1} such that $\alpha' \Rightarrow \alpha$. The lemma is immediate from the fact that that all sets are pairwise disjoint, i.e., $A_i \cap A_j = \emptyset$ for every $i \neq j$. ⊣

3.4.2. COROLLARY. *Let G be a CFG. Every derivation producing a string x of length k in $L(G)$ has the same number of rules.*

3.4.3. LEMMA. *Let G be a context free grammar. G can be transformed into a probabilistic context free grammar G' with the special property that all rules have exactly the same probability value.*

PROOF. Let G be a grammar in CNF, and let R be its set of rules. Let X be the most frequent non-terminal in the left-hand sides of rules. Let n be the number of times X is the left hand-side of a rule. Let $Z_1, \ldots, Z_n$ be brand new non-terminal symbols. For every non-terminal Y we add as many rules $Y \rightarrow Z_i$ as needed to have the number of rules having Y in the left-hand side equal to n. We add probability $1/n$ to each of these new rules. The resulting grammar is a well-defined, though not necessarily consistent, probabilistic context free grammar, and all rules have exactly the same probability values as required. ⊣

The PCF grammar G', obtained from a grammar G as described in Lemma 3.4.3, is called the *uniform version* of G.

Note that the resulting grammar need not be consistent, given that some probability mass is going to non-terminating derivations — derivations that end up in the dummy non-terminal. Still, what is important to us is that the set of trees accepted by the PCFG remains the same, and, even more importantly, that every derivation producing the same sentence has the same probability value.

3.4.4. LEMMA. *Let G be a context free grammar, and let G' be its uniform version. Let x be a string in $L(G)$. Then all left-most derivations producing x have the same probability.*

PROOF. Since every string in the language has the same set of trees as G, the dummy rule is not used in any derivation of final strings. According to Lemma 3.4.1, every tree has the same number of rules. And since every rule has the same probability, every tree for the sentence l has the same probability. Finally, the set of trees bearing maximum probability is exactly the set of trees in the original grammar G. ⊣

As this lemma proves, trees defined through MPT include the class of trees defined via CFG. As a direct consequence, we have the following lemma:

3.4.5. LEMMA. *Deciding whether a PCFG disambiguates a tree language is undecidable.*

PROOF. We have built a grammar that assigns the same probability mass to all possible trees for a given string. As a consequence, the PCFG is unambiguous if, and only if, the non-probabilistic grammar is. Deciding whether the PCFG is unambiguous is equivalent to decided whether a CFG in CNF is unambiguous, which is known to be undecidable (Hopcroft and Ullman, 1979). ⊣

The above lemma answers question (2) from the introduction, saying that it is not possible to decide whether a PCFG has completely managed to solve all ambiguities. Note that the results in the present section combined with the results in the previous one imply that the class of tree languages described by PCFGs is a strict subclass of the tree languages described by MPTGs. The inclusion is implied by the present section while the proper inclusion is implied by the previous one.

3.5 The Meaning of Probabilities

Whenever only a single tree is required as output, all CF parsers face the question of how to select that single tree from a set of trees yielding the same sentence. They usually choose a tree non-deterministically, by randomly selecting a tree among all possible trees. The selection is made under the assumption that all trees in the candidate list (suggested by the grammar) have the same probability of being selected.

The use of probabilities is meant to reduce the size of the set of candidate trees. On the one hand, the probability value assigned to a tree captures that tree's chance of being generated by the grammar and, consequently, of being found in a tree-bank generated by the grammar. On the other hand, the idea of *correctness* is usually understood in terms of a comparison to a manually annotated tree-bank. The two things combined suggest that the probability assigned to a tree can be thought of as its chance of being the correct one. On this view, parsers try to find the tree that has the highest probability of being the correct one. Clearly, some non-determinism remains: there might be more than one tree bearing maximum probability and, consequently, parsers have to non-deterministically choose among all trees bearing maximum probability.

In the following two subsections we use the just defined semantics for two different purposes. First, we use it as a way to compare grammars. The general idea is to compare grammars according to the amount of non-determinism they have left for the non-deterministic choice. Second, we want to use it for boosting the probability of picking the right tree from the set of candidate trees by adding trees to it that do not bear maximum probability but that increase the probability mass of the candidate list in substantial way.

3.5.1 Using Probabilities for Comparing PCFG

Recall that a grammar is *ambiguous* if there is a sentence in the language that has a candidate list containing more than one analysis. Following (Wich, 2000, 2001), we can think of the degree of ambiguity of a PCFG as a quantity proportional to the size of the candidate lists, one per sentence in the language. That degree is related to both

the set of rules in the grammar and to the probabilities associated to the rules. Clearly, a grammar with a lower degree of ambiguity is preferred over one with a higher degree of ambiguities given that the first reduces the level of non-determinism by choosing non-deterministically from smaller sets, in the second phase.

We propose a measure that compares grammars with respect to the way they reduce nondeterminism in the second phase of the parsing process. The measure is based on the probabilistic distribution they generate over the set of trees. Our approach is sample-based, i.e., the measure is computed over a finite sample set of sentences, because is not possible to compute it for the whole language, as we will also show. The measure we propose computes the probability of a tree of being chosen under the two-stage parsing schema defined previously. This proposal has the advantage of taking into account two things: first, the confidence the probability measure has over the proposed list of candidates, and, second, the non-deterministic choice in the final step.

One important desideratum that we have for our measure for determining a grammar's ability to reduce ambiguity is that it should capture the remaining non-determinism after trees have been filtered out using probabilities. Clearly, the reduction on non-determinism is related to the size of the set of candidate trees. However, it is not a good idea to simply use the fraction of trees that were filtered out as a quality measure, or the size of the candidate list. The first idea is unsuitable because, in case the grammar generates only a single tree per sentence, the probabilities do not filter out any tree, and we would be assigning a very low score to the filtering mechanism. The second idea fails because there is no information on the size of the list of trees before using probabilities.

There has been quite a lot of research in the area of parsing evaluation (Lin, 1995; Marcus et al., 1994; Carroll et al., 1998; Musillo and Sima'an, 2002), but it does not seem appropriate to use any of these parser evaluation measures for quantifying ambiguity reduction. Parser evaluation measures are aimed at determining how well parsers perform on parsing standard sentences. Under these approaches, only grammars that output trees that follow the structure found in the tree-bank can be compared (or those for whom a transformation between formats exists (Watkinson and Manandhar, 2001)). Moreover, these approaches do not produce any information about the way in which the grammar has dealt with ambiguity.

There have been some attempts, both to show that PCFGs do indeed reduce ambiguity and to determine the extent to which they do this. For instance, Atsumi and Masuyama (1998) compare the size of the list of candidate analyses before and after having filtered out syntactic analysis with lower probability. Even though their motivations are very similar to ours, they do not offer an explicit measure for comparing different PCFGs with respect to their ambiguity reduction abilities.

Before giving the formal definition, let us give some more intuitions. The amount

of determinism for a given sentence x in the two-stage parsing procedure is given by two main ingredients: the (size of the) set of trees $T(x)$ yielding the sentence x, and the (size of the) set of trees bearing maximum probability $PParser(x)$. Both sets contribute to ambiguity reduction. The sizes of $T(x)$ and $PParser(x)$ capture the amount of ambiguity produced by the grammar before and after having used probabilities for filtering out trees, respectively.

PCFGs reduce the set of trees in the candidate list using a probability distribution over the set of possible analysis. The distribution specifies the probability each tree has of being the correct tree given the sentence.[2] Under our two-stage procedure the probability of selecting a particular tree is given by the product of the probability mass accumulated in the set $PParser(x)$ (that is, the probability of having the correct tree in $PParser(x)$) and the probability of uniformly selecting a particular tree from $PParser(x)$. More specifically, suppose the grammar defines a probability distribution p over the set of trees, specifying the probability each tree has of being the correct one. Suppose, moreover, that for a given sentence x from the sample set, we select the set of trees bearing maximum probability $PParser(x)$. The probability of selecting any particular instance of the trees in $PParser(x)$ using a uniform distribution is

$$Q_{\{x\}}(G) = p(PParser(x))\frac{1}{|PParser(x)|},$$

where $p(PParser(x))$ is the probability that the correct tree is in $PParser(x)$, while $\frac{1}{|PParser(x)|}$ is the probability of selecting it. The probability takes into account the probability mass concentrated in $PParser(x)$ and its size: the bigger the probability the better the output.

Since all trees in $PParser(x)$ have the same probability value p_x, $Q_{\{x\}}(G)$ can be simplified as follows

$$Q_{\{x\}}(G) = p_x|PParser(x)|\frac{1}{|PParser(x)|} = p_x.$$

Finally, assuming that parsing sentences are independent experiments, our measure is defined as follows:

$$Q_S(G) = \prod_{x \in S} p_x,$$

where S is a sample set of sentences from the grammar's accepted language, and p_x is the probability assigned to the tree returned by the parser. $Q_S(G)$ is equal to 1 if, and only if, there is a unique tree with maximum probability for each sentence in S. The measure is easily computable if we work with probabilistic parsers that return both trees and the probability value associated to the trees returned.

[2] "Correct tree" in the sense that it is the tree that appears in a sample tree-bank.

Finally, we say that a grammar G_1 is *better* than a grammar G_2 (based on a sample set S) if, and only if,

$$Q_S(G_1) < Q_S(G_2).$$

The measure does not capture the ambiguity reduction over the set of all possible sentence. Why? In the following section we show that it is simply not possible to compute it for the whole language.

In what follows we show that it is necessary to relativize our measure to a sample set: it is not possible to compute $Q_S(G)$ if S is equal to the language accepted by the grammar $L(G)$. Suppose that it were possible to compute $Q_{L(G)}(G)$. Then, we would also know whether $Q_{L(G)}(G)$ is equal to one. Since $Q_{L(G)}(G) = 1$ if, and only if, G has singled out exactly one element in the candidate list of each sentence, being able to compute $Q_{L(G)}(G)$ would imply that it is possible to determine whether G has completely disambiguated the language.

We have presented a measure for assessing grammars with respect to their ability to reduce ambiguity. The measure we presented can also be applied to state-of-art-parsers that return the selected analysis tree for a given input sentence *together with its probability* (Collins, 1997; Eisner, 1996; Klein and Manning, 2003).

Our measure has at least three kinds of advantages in comparison to standard parser evaluation methods:

1. It can be applied to unsupervised learned grammars for which the learned syntactic structure is not as clearly defined as the ones induced from tree-banks.

2. Our measure is not domain dependent. Since a grammar induced from a tree-bank is usually evaluated on the same type of sentences that were used for inducing it, its evaluated performance does not tell us much about the grammars' performance on sentences belonging to different domains from those covered in the tree-bank.

3. Our measure yields information about the parser that is complementary to the kind of information usually obtained by evaluating parsers (Lin, 1995; Marcus et al., 1994; Carroll et al., 1998), given that it does not provide any kind of information about the correctness of the resulting trees, and, moreover, the measure does need to have access to the 'right' tree. The precise relation between performance measured using existing parser evaluation measures and performance measured with our new measure (applied to parsers) remains to be explored.

This subsection showed a possible use of probabilities other than probabilities as filtering mechanism. The new semantics provides an answer to question (3) of the introduction; in the next section we show yet another application.

3.5.2 Using Probabilities for Boosting Performance

In this section we use the new semantics associated to probabilities for increasing the probability of guessing the right tree. We provide evidence that under certain circumstances the probability of getting the correct tree can be increased by adding trees with less than maximal probability to the set of candidates.

Note that the probability of $PParser(x)$ is the probability of having the correct tree in it. We can forget for a moment that $PParser(x)$ only contains trees bearing maximum probability and add trees to it in an attempt to increment its probability. Incrementing its probability has the advantage of incrementing the probability of capturing the correct tree, but has the disadvantage of decrementing the probability of randomly choosing the correct one. Clearly, there is a trade-off between the number of non-maximum probability trees we can add to $PParser(x)$ and the probability gained at the end of the random selection procedure. Let us take a closer look, and give conditions under which the probability of selecting the correct trees increases when picking from a set of trees properly extends the set of trees bearing maximum probability.

Let R be a set of trees disjoint with $PParser(x)$. We show that the probability of choosing the correct tree increases when R is added to the candidate list $PParser(x)$ if, and only if,

$$\frac{p(R)}{p_x} > |R| + |PParser(x)| - 1. \tag{3.1}$$

where p_x is the probability values shared by all trees in $PParser(x)$. The proof is simple. The condition in (3.1) above is fulfilled if, and only if,

$$p(R) + p_x > p_x|R| + p_x|PParser(x)|,$$

which is equivalent to

$$\frac{p(R) + p_x}{|R| + |PParser(x)|} > p_x,$$

which, in turn, holds if, and only if,

$$\frac{p(R \cup PParser(x))}{|R \cup PParser(x)|} > p_x.$$

The final result follows from the fact that $\frac{p(R \cup PParser(x))}{|R \cup PParser(x)|}$ is the probability of selecting the correct tree from the expanded list.

Extending the set of candidates is not new in the literature. Collins and Duffy (2001, 2002); Bod (2003) propose approaches other than uniformly selecting a tree. Our result gives an estimate of the number of trees one needs to consider in the selection phase to gain a significant amount of probability mass.

The present section focused on new uses of probabilities associated to trees. We answered question (3) by giving two new applications. We show that probabilities can be used for evaluation purposes and that they can be used to boost parsing performance.

As trivial as the results might look, it is very hard to find a parser implementation that reports on the probability values associated to its trees. In order to be able to compare parsers using the probability values they assign to their most probable trees, the probabilistic models used by the parsers should be consistent. If this is not the case, the probability values are meaningless. Clearly, this requirement is not trivial to fulfill. For most of the probabilistic parsers it is unknown whether their underlying models are consistent. Given the importance of the consistency conditions, we establish in Chapter 4, consistency properties for bilexical grammars, and Markovian context free grammars.

3.6 Conclusions and Future Work

This chapter has focused on questions related to the importance of probabilities in the context of parsing and on investigating uses of probabilities others than filtering. We have shown that probabilities, when used as a filtering mechanism, can add expressive power to grammars defining a class of tree languages beyond the expressivity of context free grammars. We also showed that it is not possible to decide whether probabilities solve all ambiguities in the language.

We also gave examples that probabilities can be used in other ways others than as filtering mechanism. We proposed to use them for evaluating the quality of PCFGs and for boosting the performance of parsers. Experiments to back up this theoretical finding would be of great value. We leave them out of the thesis and consider them as future work.

We argued that in order to use these ideas, it is essential that the underlying probability models be consistent. We show in Chapter 4 that PCW-grammars provide us with the machinery necessary to prove the consistency of bilexical grammars and Markovian rules whenever they are induced from tree-banks using n-grams. Despite this, proving consistency properties for the grammars we induce in Chapters 5, 6 and 7 are desirable results on the todo list.

Chapter 4
Constrained W-Grammars

This chapter consists of two sections. First, the grammatical formalism we propose is introduced. Our framework is based on so-called W-grammars, due originally to Van Wijngaarden (Van Wijngaarden, 1965; Mateescu and Salomaa, 1997). We constrain W-grammars to obtain *constrained W-grammars* (CWG) which are more suitable for statistical natural language parsing than W-grammars. As we have seen in the previous chapter, probabilities are a fundamental part of state-of-the-art natural language parsing technology. We extend CW-grammars with probabilities, defining *probabilistic constrained W-grammars* (PCWGs).

In the second part of this chapter we show that the formalism we introduce is general enough to capture grammatical frameworks used in three state-of-the-art parsers: bilexical grammars, Markovian context free grammars, and stochastic tree substitution grammars (STSGs). For each we provide an embedding into PCW-grammars, which allows us to discover properties about their expressive power and consistency, and relations between the formalisms studied.

4.1 Grammatical Framework

In this section we describe the grammatical framework we will be working with. We introduce constrained W-grammars, then present a probabilistic version, and also introduce technical notions needed in later sections.

4.1.1 Constrained W-Grammars

A *constrained W-grammar* (*CW-grammar*) is a 6-tuple $(V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ such that:

- V is a set of symbols called *variables*. Elements in V are denoted with overlined capital letters, e.g., $\overline{A}$, $\overline{B}$, $\overline{C}$.

- NT is a set of symbols called *non-terminals*; elements in NT are denoted with upper-case letters, e.g., X, Y, Z.

- T is a set of symbols called *terminals*, denoted with lower-case letters, e.g., a, b, c.

- V, T and NT are pairwise disjoint.

- S is an element of NT called the *start symbol*.

- $\xrightarrow{m}$ is a finite binary relation defined on $(V \cup NT \cup T)^*$ such that if $x \xrightarrow{m} y$, then $x \in V$. The elements of $\xrightarrow{m}$ are called *meta-rules*.

- $\xrightarrow{s}$ is a finite binary relation on $(V \cup NT \cup T)^*$ such that if $u \xrightarrow{s} v$ then $u \in NT$, v is not empty and no variable in v appears more than once. The elements of $\xrightarrow{s}$ are called *pseudo-rules*.

CW-grammars differ from Van Wijngaarden's original W-grammars in that pseudo-rules have been constrained. Comparing the above definition with the one presented in Section 2.4, we see that the original W-grammars allow pseudo-rules to have variables on the left-hand side as well as repeated variables on both the right- and left-hand side. The constrained version defined above yields a dramatic reduction in the expressive power of W-grammars. CW-grammars are weakly equivalent to context free grammars. Despite the reduction of expressivity, CW grammars are capable of fully capturing grammar formalism used in state-of-the-art parsers, something that context free grammars can not do by themselves.

CW-grammars are rewriting devices, and as such they consist of rewriting rules. They differ from the usual rewriting systems in that the rewriting rules do not exist *a priori*. Using pseudo-rules and meta-rules one builds 'real' rules that can be used in the rewriting process. The rewriting rules produced are denoted by $\overset{w}{\Longrightarrow}$ and are called *w-rules*. These rules are built by first selecting a pseudo-rule, and then using meta-rules for instantiating all the variables the pseudo-rule might contain.

For example, let $W = (V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ be a CW-grammar where $V = \{\overline{ADJ}\}$, $NT = \{S, Adj, Noun\}$, $T = \{ball, big, fat, red, green, \ldots\}$, and $\xrightarrow{m}$ and $\xrightarrow{s}$ are given by the following table:

meta-rules	pseudo-rules
$\overline{ADJ} \xrightarrow{m} \overline{ADJ}\,Adj$	$S \xrightarrow{s} \overline{ADJ}\,Noun$
$\overline{ADJ} \xrightarrow{m} Adj$	$Adj \xrightarrow{s} big$
	$Noun \xrightarrow{s} ball$
	$\vdots$

Suppose now that we want to build the w-rule $\overline{S} \xRightarrow{w} Adj\ Adj\ Noun$. We take the pseudo-rule $\overline{S} \xrightarrow{s} \overline{ADJ}\ Noun$ and instantiate the variable $\overline{ADJ}$ with $Adj\ Adj$ to get the desired w-rule. The w-rules defined by W have the following shape: $\overline{S} \xRightarrow{w} Adj^*\ Noun$. Trees for this grammar are flat, with a root node S and series of adjectives and nouns as daughters; see Figure 4.1.

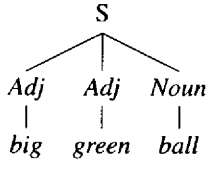


Figure 4.1: A flat w-tree.

The *string language* $L(W)$ generated by a CW-grammar W is the set $\{\beta \in T^+ : S \xRightarrow{w}{}^* \beta\}$. In words, a string β belongs to the language $L(W)$ if there is a way to instantiate w-rules $\xRightarrow{w}$ in such a way that they can derive β from S. A *w-tree* yielding a string l is defined as the $\xRightarrow{w}$ derivation producing l. A w-tree 'pictures' the w-rules (i.e., pseudo-rules + variable instantiations) that have been used for deriving a string; Figure 4.1 has an example. The way in which a w-rule has been obtained from pseudo-rules or the way in which its variables have been instantiated remains hidden. The *tree language* generated by a CW-grammar W is the set $T(W)$ consists of all w-trees generated by W yielding a string in $L(W)$.

4.1.1. THEOREM. *CW-Grammars are weakly equivalent to context free grammars.*

PROOF. Let $W = (V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ be a CW-grammar. Let $G_W = (NT', T', S', R')$ be a context free grammar defined as follows (to avoid confusion we denote the rules in R by $\rightarrow$): $NT' = (V \cup NT)$; $T' = T$; S' is identical to S; and $X \rightarrow \alpha \in R$ iff $X \xrightarrow{m} \alpha$ or $X \xrightarrow{s} \alpha$.

Obviously, one has to prove that G is well-defined. The most problematic part is the definition of the rules. We need to check whether all rules are well-formed. It is enough to check that every left-hand side has one and only one non-terminal. If a rule $X \rightarrow \alpha$ is in R because $X \xrightarrow{m} \alpha$, then it is clear that X is a non-terminal in V. If it is in because of $X \xrightarrow{s} \alpha$, then it is a non-terminal given that X has to be in NT.

Next we show that the two grammars generate the same language. To prove this we just prove that if $r \overset{w}{\Longrightarrow}^{*} s$ then $r \rightarrow^{*} s$. The proof is divided in three parts: (i) if $r \overset{m\ *}{\longrightarrow} s$ then $r \rightarrow^{*} s$; (ii) if $r \overset{s}{\longrightarrow} s$ then $r \rightarrow s$; (iii) if $r \overset{w}{\Longrightarrow} s$ then $r \rightarrow^{*} s$. (i) If $r \overset{m\ *}{\longrightarrow} s$ then $r \rightarrow^{*} s$ follows immediately from the definition of G_W; $\overset{m}{\longrightarrow}$ is a subset of $\rightarrow$. (ii) If $r \overset{s}{\longrightarrow} s$ then $r \rightarrow s$ also follows from the definition of G_W. (iii) Let r and s be such that $r \overset{w}{\Longrightarrow}^{*} s$. This implies that there exist r' and s' such that

$$\begin{array}{ccc} r' & \overset{s}{\longrightarrow} & s' \\ \downarrow^{m}_{*} & & \downarrow^{m}_{*} \\ r & & s \end{array}$$

Since $r' \in NT$, the only possibility for $r' \overset{m\ *}{\longrightarrow} r$ is for r to be equal to r', leading to the following situation:

$$\begin{array}{ccc} r & \overset{s}{\longrightarrow} & s' \\ & & \downarrow^{m}_{*} \\ & & s \end{array}$$

But, under the assumption that every variable appears once and only once in the body of s', both $\overset{s}{\longrightarrow}$ and $\overset{m\ *}{\longrightarrow}$ can be emulated by $\rightarrow$, implying that $r \rightarrow^{*} s$ as required.

In the other direction, any CFG G can define a CW-grammar W. The only necessary step for transforming G into a CW-grammar consists of partitioning the set of rules in G into two different subsets, one functioning as the set of meta-rules, and the other functioning as the the set of pseudo-rules. ⊣

Note that the previous result does not follow directly from any of the results given in Chapter 2. In the literature there are examples of different constraints applied to W-grammars (Mateescu, 1989a,b). These constraints were meant to make subclasses of enumerable languages. In contrast, our constraints are meant to reduce the expressive power of W-grammars to a level that allows us to capture the grammatical formalisms underlying state-of-the-art-parsers. The previous lemma shows that our constraints reduce the expressive power of W-grammars from Turing computable to context free.

Given a CW-grammar W, the *context free grammar underlying* W, which we denote by $CFG(W)$, is the grammar G_W defined in the proof of Theorem 4.1.1. In order to facilitate our forthcoming discussion we suppose that rules in G have been marked somehow to allow us to decide whether a rule in G corresponds to a meta-rule or to a pseudo-rule in the original CW-grammar. We refer to the set of rules in G marked as meta-rules as meta-rules and to the set of rules marked as pseudo-rules as pseudo-rules.

Let W be a W-Grammar and let $G = CFG(W)$ be its underlying context free grammar. Theorem 4.1.1 shows that both W and G accept the same string language. In what follows we turn to study the relation between W's tree language and G's tree

language. We show that there is a surjective mapping Γ from $T(G)$ to $T(W)$ which can be used for effectively parsing W-grammars, as we will soon discuss.

We define our *tree transformation function* Γ using a tree rewriting schema. The rewriting schema is applicable only to trees containing at least one meta-rule. Our intention is to rewrite those trees in $T(G)$ into trees in $T(W)$. After each application of the rewriting schema, the number of rules marked as meta-rules is reduced by one. The function Γ is defined as the recursive application of the schema until applications are no longer possible, i.e., until all meta-rules have been eliminated. The intermediate trees in the rewriting procedure do not necessarily belong to either of the two tree languages $T(G)$ or $T(W)$. The rewriting schema is pictured in Figure 4.2. Left-hand symbols of meta-rules and pseudo-rules have been marked with superscripts m and s respectively. The rewriting schema eliminates the rule $V \xrightarrow{m} X_1 X_2 \ldots X_k$ by transforming the tree in part (a) into the tree in part (b).

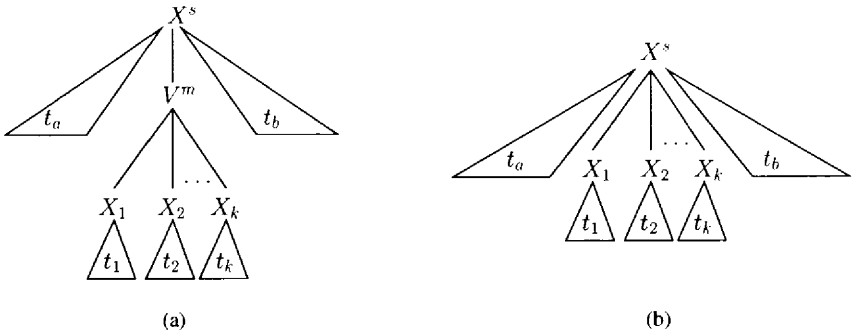


Figure 4.2: A tree rewriting schema.

The function Γ for transforming trees in $T(W)$ into trees in $T(G)$ is defined as follows:

$$\Gamma(t) = \begin{cases} t, & \text{if } t \text{ does not contain any meta-variable} \\ \Gamma(t'), & \text{for } t' \text{ such that } t \rightarrow t', \end{cases}$$

where $\rightarrow$ is the reflexive and transitive closure of the rewriting schema defined above. A simple inductive proof on the number of meta-rules shows that Γ is well defined for all elements in $T(G)$, i.e., Γ takes exactly one value for each t in $T(G)$.

In Figure 4.3 we picture the rewriting procedure for the tree in Figure 4.1. The variable $\overline{ADJ}$ that is eliminated in each step is marked with $*$ symbols. Since the tree in part (c) does not have any more variables, it corresponds to the result of applying function Γ to the tree in part (a).

The function Γ is very important for parsing. With it, we can implement a parser for W-grammars by using a parser for CFGs plus a procedure implementing the function

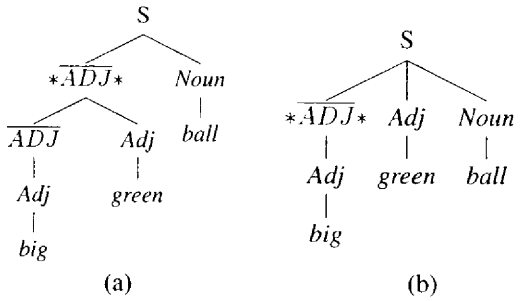


Figure 4.3: Two steps in the rewriting procedure. Γ applied to the tree in part (a) is results in the tree in Figure 4.1.

Γ. The CFG parser is first used to obtain the tree t in $T(G)$, and then the procedure implementing Γ is used for transforming t into a tree in $T(W)$.

Both the weak equivalence and the existence of the transformation function Γ suggest that CW-grammars are close to CF-grammars. Do they offer anything more than just context freeness? Since parsing technology is more interesting in the tree language than the string language, the strong equivalence between CW-grammars and CF-grammars becomes relevant. We give an example to show that CW-grammars are *not* strongly equivalent to context free grammars. In other words, the set of trees generated by CW-grammars differs from the set of trees generated by context free grammars.

4.1.2. EXAMPLE. Let $W = (V, NT, T, S, \xrightarrow{m}{}^{*}, \xrightarrow{s}{}^{*})$ be a CW-grammar with $V = \{\overline{A}, \overline{B}, \overline{S}\}$, $NT = \{A, B\}$, $T = \{a, b\}$, $\xrightarrow{m} = \{\overline{A} \xrightarrow{m} \overline{A}A, \overline{A} \xrightarrow{m} A, \overline{B} \xrightarrow{m} \overline{B}B, \overline{B} \xrightarrow{m} B\}$, and $\xrightarrow{s} = \{A \xrightarrow{s} a, B \xrightarrow{s} b, \overline{S} \xrightarrow{s} \overline{AB}\}$.

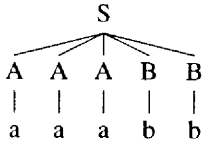


Figure 4.4: A w-tree for the string "*aaabb*".

The grammar W generates the language $\{a^*b^*\}$ through instantiations of the variables A and B to strings in A^* and B^*, respectively. The derivation $\xRightarrow{w}$ for a string *aaabb* is as follows: $S \xRightarrow{w} AAABB \xRightarrow{w} aAABB \xRightarrow{w} aaABB \xRightarrow{w} aaaBB \xRightarrow{w} aaabB \xRightarrow{w}{}^{*}$ *aaabb*. The tree representing this derivation (Figure 4.4) has only one internal level (labeled $AAABB$), and its leaves form the accepted string. No context free grammar can generate the kind of flat structures displayed in Figure 4.4 since any context free

grammar producing the same language as W has more than one intermediate level in its derivation trees.

4.1.2 Probabilistic CW-Grammars

In Chapter 3, probabilities were shown to be a fundamental part of PCFGs because they add expressive power to vanilla CFGs. In order to capture state-of-the-art-parsers, we need to be able to mimic the expressive power of PCFGs. Even though CW-grammars are more powerful than CFGs, we do not know for certain if their probabilistic version, to be defined below, adds expressive power to the vanilla version. Still, probabilities add to CW-grammars a statistical perspective which we show to be very useful in the following chapters, and which is necessary to easily capture grammar formalisms used in state-of-the-art parsers.

Probabilistic CW-grammars (PCW-grammars) are CW-grammars where the meta-rules and pseudo-rules are augmented with probability values, such that the probabilities belonging to meta-rules and pseudo-rules sharing the same left-hand side sum up to one. More formally, in a probabilistic CW-grammar $(V, NT, S, \xrightarrow{m}, \xrightarrow{s})$ we have that

- $\sum_{x \xrightarrow{m}_p y} p = 1$ for all meta-rules $x \xrightarrow{m}_p y$ having x as the left-hand side.
- $\sum_{x \xrightarrow{s}_p y} p = 1$ for all pseudo-rules $x \xrightarrow{s}_p y$ having x as the left-hand side.

Next, we need to define how we assign probabilities to derivations, w-rules, and w-trees. To start with derivations, if $\alpha' \xrightarrow{m}{}^{*} \alpha$ then there are $\alpha_1, \ldots, \alpha_k$ such that $\alpha_i \xrightarrow{m} \alpha_{i+1}$, $\alpha_1 = \alpha'$ and $\alpha_k = \alpha$. We define the probability $P(\alpha' \xrightarrow{m}{}^{*} \alpha)$ of a derivation $\alpha' \xrightarrow{m}{}^{*} \alpha$ to be $\prod_{i=1}^{k-1} P(\alpha_i \xrightarrow{m} \alpha_{i+1})$

Now, let $X \xRightarrow{w} \alpha$ be a rule; its probability $P(X \xRightarrow{w} \alpha)$ is defined as

$$P(X \xRightarrow{w} \alpha) = \sum_{\alpha' \in A} P(X \xrightarrow{s} \alpha') P(\alpha' \xrightarrow{m}{}^{*} \alpha),$$

where $A = \{\alpha' \in (V \cup NT \cup T)^+ : X \xrightarrow{s} \alpha', \alpha' \xrightarrow{m}{}^{*} \alpha\}$. In other words, the probability of a w-rule is the sum of the probabilities of all meta derivations producing it.

The *probability of a tree* is defined as the product of the probabilities of the w-rules making up the tree, while the *probability of a string* $\alpha \in T^+$ is defined as the sum of the probabilities assigned to all trees yielding α.

4.1.3. THEOREM. *Let W be a CW-grammar, let G be $CFG(W)$, and let W' be a PCW-grammar that extends W by assigning probability values to all meta-rules and*

pseudo-rules in W. There is a way to extend G into a PCFG G' such that W' and G' assign the same probability mass to all strings in the language accepted by G (which coincides with the language accepted by W).

PROOF. Let $G = (NT', T', S', R')$ be a PCFG with NT', T', S' as defined in the proof of Theorem 4.1.1 and R' such that $X \to \alpha \in R$ iff $X \xrightarrow{m} \alpha$ or $X \xrightarrow{s} \alpha$. Note that a $\overset{w}{\Longrightarrow}$ derivation τ might be the product of many different derivations using rules in R' (G-derivations for short); let us notate this set of G-derivations with $D(\tau)$. From the definitions it is clear that $p(\tau) = \sum_{\upsilon \in D(\tau)} p(\upsilon)$. To prove the theorem we need to show

1. that for two different $\overset{w}{\Longrightarrow}$ derivations of the string $\alpha\,\tau$ and τ', it holds that $D(\tau) \cap D(\tau') = \emptyset$, and

2. that for every G-derivation υ there is a $\overset{w}{\Longrightarrow}$ derivation τ such that $\upsilon \in D(\tau)$.

Item (1) follows from the facts that $\Gamma(\upsilon) = \tau$ for all υ in $D(\tau)$ and that $\Gamma(\upsilon') = \tau'$ for all υ' in $D(\tau')$; consequently, if there is an element in $D(\tau) \cap D(\tau')$, then τ is equal to τ'. Item (2) follows from the fact that Γ is defined for all G-derivations. ⊣

The above result does not follow from Theorem 4.1.1 because that result does not take probabilities into account.

For a given PCW-grammar W, the PCFG defined in the proof of Theorem 4.1.3 is called the *PCFG underlying* W.

As in the case of non-probabilistic CW-grammars, the tree language of $T(G)$ is related to the tree language $T(W)$ through the tree transformation function Γ defined above. The function Γ can also be used for computing the probability of a tree t in $T(W)$; its probability is equal to the sum of the probabilities of all trees t' in $T(G)$ such that $\Gamma(t') = t$. Moreover, since every tree in $T(G)$ is mapped to a tree in $T(W)$, we obtain that the W grammar is consistent if, and only if, its underlying PCFG is consistent.

4.1.3 Learning CW-Grammars from Treebanks

Suppose we have a collection of w-trees, from which we would like to induce a CW-grammar. We distinguish two different strategies for solving this problem. One strategy is aimed at using *supervised* techniques for learning PCFGs while the second is aimed at using *unsupervised* techniques for learning PCFGs.

The first approach consists of three steps.

1. Handcraft a set of meta-rules that might be used for meta-derivation.

2. Expand each tree with all meta-derivations. Such a procedure has the effect of transforming the CW-treebank into a CF-treebank.

3. Use any PCFG inducing mechanisms, like those discussed in Section 2.3 for inducing a PCFG from the transformed treebank.

Instead of returning a CW-grammar, the procedure returns the underlying PCFG G of a W-grammar. Since all meta-rules are known, a unique W-grammar can be defined from G.

Step (2) is straightforward only if meta-derivations produce unambiguous derivations, i.e., if for each string produced by meta-derivations produce there is only one way to derive it. In the case of ambiguous meta-derivations, a way to distribute the probability mass among all possible derivations has to be designed. In the literature there are different proposals for re-assigning probabilities (Sima'an and Buratto, 2003; Bod, 1998; Krotov et al., 1998). Most of these references refer to estimation techniques for STSGs that, as we will see, deal these problems from the very beginning.

The second approach consists of 2 steps.

1. Extract all bodies of rules from the w-treebank where each body of a rule is a string of non-terminal and terminal symbols.

2. Use the extracted strings to induce a PCFG grammar.

In contrast with the previous approach, the induced PCFG is not the underlying PCFG of any CW-grammar; instead, it is the grammar describing the meta-derivation a PCW-grammar should have. Pseudo-rules have to be handcrafted in order to make the meta-rules interact with pseudo-rules to rebuild the trees that have appeared in the CW-grammar.

Both approaches are different and require different techniques. Both have in common that one set of rules, either pseudo-rules or meta-rules have to be handcrafted. The presentation we have given here is rather abstract; the differences between the two become more evident in the rest of the thesis. In Section 4.2 we show that the learning methodology used for state-of-the-art parsers is an instance of the first approach, while the grammars used in the experiments of Chapter 5 are instances of the second approach.

4.1.4 Some Further Technical Notions

Below we will use PCW-grammars to "capture" models underlying a number of state-of-the-art parsers. The following will prove useful. Let F and G be two grammars with tree languages $T(G)$ and $T(F)$ and languages $L(F)$ and $L(G)$, respectively. Then, F is

f-equivalent to G if $L(F) = L(G)$ and there is a bijective function $f : T(F) \rightarrow T(G)$. Given two grammatical formalisms A and B, we say that A is *f-transformable* to B, if for every grammar F in A there is a grammar G in B such that F is f-equivalent to G. Note that the definition of f-transformable is a generalization of the concepts of weak and strong equivalence; both can be seen as f-equivalence for particular choices of the function f. Namely, two grammars are weakly equivalent if the function f is surjective, and they are equivalent if the function f is bijective and t and $f(t)$ are isomorphic trees, for all trees in the domain of f.

4.2 Capturing State-of-the-Art Grammars

In this section we show that PCW-grammars are a powerful formalism; we show that they are powerful enough to capture the models underlying a number of state-of-the-art parsers. Clearly, the grammatical framework underlying a parser is a key component of the overall definition of the parser which determines important characteristics of the parser, either directly or indirectly. Among others things, the grammatical framework defines the set of languages the parser can deal with, a lower bound on the parser's complexity, and the type of items that should be learned by the second component mentioned in Section 4.1.3. Hence, a thorough understanding of the grammatical framework on which a parser is based, provides a great deal of information about the parser itself. We are particularly interested in the following properties:

1. The expressive power of a grammar formalism.

2. Conditions under which the probability distribution defined over the set of possible syntactic analyses is consistent: if this is the case, the probabilities associated with an analysis can be used as meaningful probabilistic indicators both for further stages of processing (Manning and Schütze, 1999) and for evaluation (Infante-Lopez and de Rijke, 2004b).

3. The relation to other grammatical frameworks; this provides insights about the assumptions made by the various frameworks.

Since building a parser is a time consuming process, formal properties of the underlying grammatical framework are not always a priority. Also, comparisons between parser models are usually based on experimental evidence. In order to establish formal properties of parsers and to facilitate the comparison of parsers we believe that a unifying grammatical framework, from which the grammars of different parsers can be obtained as instances, is instrumental. We show that the PCW framework is capable of capturing three state-of-the-art grammatical formalisms, namely bilexical grammars

(Eisner, 2000), Markovian context free grammars (Collins, 1997), and stochastic tree substitution grammars (Bod, 1998). For each of these three formalisms, we provide an embedding in PCW-grammars, and we use this embedding to derive results regarding expressive power, consistency, and relations with other grammatical formalisms.

4.2.1 Bilexical Grammars

Bilexical grammars (Eisner, 1996, 2000) is a formalism in which lexical items, such as verbs and their arguments, can have idiosyncratic selectional influences on each other. Such grammars can be used for describing bilexical approaches to dependency and phrase-structure grammars, and a slight modification yields link grammars.

Background

A *split unweighted bilexical grammar* B is a 3-tuple $(W, \{r_w\}_{w \in W}, \{l_w\}_{w \in W})$ where:

- W is a set, called the (terminal) *vocabulary*, which contains a distinguished symbol ROOT.
- For each word $w \in W$, l_w and r_w are a pair of regular grammars with start symbols S_{l_w} and S_{r_w} respectively. Each grammar accepts some regular subset of W^*.

A *dependency tree* is a tree whose nodes (internal and external) are labeled with words from W; the root is labeled with the symbol ROOT. The children ('dependents') of a node are ordered with respect to each other and the node itself, so that the node has both *left children* that precede it and *right children* that follow it. A dependency tree T is *grammatical* if for every word token w that appears in the tree, l_w accepts the (possibly empty) sequence of w's left children (from right to left), and r_w accepts the sequence of w's right children (from left to right).

4.2.1. EXAMPLE. Let $B = (W, \{l_w\}_{w \in L}\}, \{r_w\}_{w \in L})$ be a split bilexical grammar defined as follows: $W = \{a, b, ROOT\}$, $l_a = b^*$, $r_a = \epsilon$, $l_b = \epsilon$, $l_{ROOT} = a$, $r_{ROOT} = \epsilon$ and $r_b = (a|b)^*$.[1] This grammar accepts the string "bbabaa" because l_{ROOT} accepts a, l_a accepts "bbb", l_a accepts ϵ, l_b accepts ϵ, and r_b accepts "a". See, for example, the tree in Figure 4.5.

[1] We use regular expressions instead of automata because the former are more compact. In order to make the example follow the definition, regular languages have to be transformed into automata.

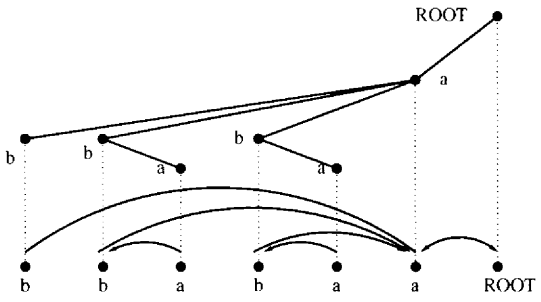


Figure 4.5: An example of a dependency tree.

Bilexical Grammars as CW-grammars

With every bilexical grammar B we can associate a CW-grammar W_B according to the following definition.

4.2.2. DEFINITION. Let $B = (W, \{l_w\}_{w \in L}\}, \{r_w\}_{w \in W})$ be a split bilexical grammar. Let $W_B = (V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ be the CW-grammar defined as follows:

- The set of variables V consists of the set of start symbols S_{l_w} and S_{r_w} from regulars grammars l_w and r_w respectively, for each w in W.
- The set of non-terminals NT is some set in 1-1-correspondence with W, e.g., it can be defined as $NT = \{W : w \in W\}$ using a string-priming operation.
- The set of terminals T is the set of words W.
- The set of meta-rules is given by the union of $\{w' \xrightarrow{m} w : w \in W\}$ and the rules in all of the right and left regular grammars in B.
- The set of pseudo-rules is given by $w' \xrightarrow{s} S_{\bar{l}_w} w S_{r_w}$ where $\bar{l}_w$ denotes the regular expression inverting (reading backwards) all strings in $L(l_w)$.

Below, we establish the (weak) equivalence between a bilexical grammar B and its CW-grammar counterpart W_B. The idea is that the set of meta-rules, which produce derivations that remain hidden in the tree, are used for simulating the regular automata. Pseudo-rules are used as a nexus between a hidden derivation and a visible one: for each word w in the alphabet, we define a pseudo-rule having w as a terminal, and two variables S_{l_w} and S_{r_w} marking the left and right dependents, respectively. These variables correspond to the start symbols for the left and right automata l_w and r_w, respectively. Instantiating the pseudo-rule associated to w would use a left and a right derivation using the left and the right automata, respectively, via meta-rules. The whole derivation remains hidden in the $\stackrel{w}{\Longrightarrow}$ derivation, as in bilexical grammars.

4.2.3. LEMMA. *Bilexical grammars are f-transformable to CW-grammars.*

PROOF. We have to give a function $f : T(B) \rightarrow T(W_B)$, where B is a bilexical grammar and W_B the grammar defined in Definition 4.2.2, such that f is invertible. A bilexical tree yielding the string $s = w_1, \ldots, w_n$ can be described as a sequence $u_1, \ldots, u_n$ of 3-tuples $\langle \alpha_i, w_i, \beta_i \rangle$ such that l_{w_i} accepts α_i and r_{w_i} accepts β. The desired function f transforms a dependency tree in a w-tree by transforming the sequence of tuples into a $\stackrel{w}{\Longrightarrow}$ derivation. We define f as $f(\langle \alpha, w_i, \beta \rangle) = W_i \stackrel{w}{\Longrightarrow} \alpha w_i \beta$. The rule corresponding to $\langle \alpha, w_i, \beta \rangle$ is the one produced by using the pseudo rule $W_i' \stackrel{s}{\longrightarrow} S_{l_w^-} x S_{r_w}$ and instantiating $S_{l_w^-}$ and S_{r_w} with α and β respectively. Since the sequence of tuples forms a dependency tree, the sequence of w-rules builds up a correct w-tree. ⊣

Weighted bilexical grammars are like unweighted bilexical grammars but all of their automata assign weights to the strings they generate. By Lemma 4.2.3, weighted bilexical grammars are a subset of PCW-grammars.

Expressive Power and Consistency

By Lemma 4.2.3 bilexical grammars are weakly equivalent to context free grammars. In order to prove that they are not strongly equivalent it is enough to note that the grammar in Example 4.2.1 can generate trees like the ones pictured in Figure 4.6 for arbitrary k. That is, the grammar in Example 4.2.1 can generate flat trees like the ones pictured in Figure 4.6 where nodes can have arbitrarily many siblings. Any CFG generating the same string language will produce non-flat structures.

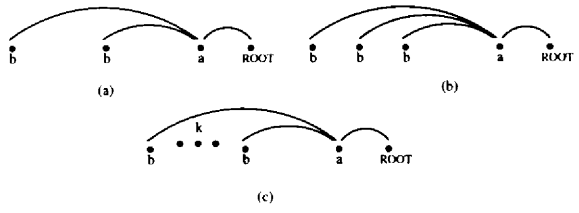


Figure 4.6: Different dependency trees that can not be generated by CFGs.

We now show that the learning mechanism proposed in (Eisner, 1996) produces consistent probability distributions. We start by presenting the way Eisner induces a bilexical grammar B from a bilexical treebank; next, we show how the given treebank can be transformed into a CF treebank. Finally, we show that the transformed treebank can be used to learn the underlying PCFG G of a PCW-grammar W such that W is equivalent to B.

In (Eisner, 1996), bilexical grammars are learned under the assumption that all words share a common automaton A and that this unique automaton is learned using bigrams, or equivalently using a degree one Markov chain (see Appendix B for an introduction to Markov chains in the context of natural language processing). The learning procedure proceeds as follows. First, training material for inducing the automaton is extracted from the bilexical treebank. The training material is constructed adding all right and left dependents strings in all bilexical trees. Extracted strings should be prefixed with a special mark "-s-" and postfixed with the special mark "-e-". These marks should be treated as indivisible units. For example, if a tree like the one pictured in Figure 4.5 is found in the treebank, strings "$-s-bba-e-$", "$-s-a-e-$", "$-s-a-e-$","$-s-a-e-$","$-s-b-e-$", "$-s-ba-e-$", "$-s-a-e-$", "$-s-a-e-$", "$-s-b-e-$", "$-s-ba-e-$", "$-s-b-e-$","$-s-b-e-$" are to be added to the training material. Since all words share the same automaton, the definition of grammar B is direct from automaton A.

Now, suppose that a bilexical grammar G is learned as described above using an automaton A, and suppose that G_A is the linear PCFG equivalent to the automaton A (for details on the equivalence between linear CFGs and automata see (Hopcroft and Ullman, 1979); for details on the probabilistic flavor see (Abney et al., 1999)).

Our proof is complete if we manage to transform the bilexical treebank into a CF treebank that can be used to learn a consistent CFG G. We also need to show that a W-grammar W can be defined such that its underlying PCFG is equal to G and that W is equivalent to B. In what follows we show how to accomplish this.

We start by transforming the bilexical-treebank into a CF-treebank. The main idea of the transformation is to rewrite trees using the inverse of the transformation defined in Lemma 4.2.3. Trees in the bilexical treebank are transformed into CF trees using the assumption that the meta-rules used actually belong to G. Figure 4.7 shows an example of such a transformation.

All transformed trees form a new CF-treebank. Using a maximum likelihood estimation technique (see Section 2.3 for details) we can induce a PCFG G. Note that the set of rules in G can be seen as the union of two subsets. The first is the set $\{X_a \rightarrow_1 -s-a-s-\}$ where a is terminal symbol and X_a is a non-terminal uniquely associated to a. All such rules have probability 1 since each variable X_a is always expanded with the same body across the whole treebank. The second subset is given by the set of rules in G_A.

G is a consistent grammar given that it has been induced using maximum likelihood (Chi and Geman, 1998; Joan-Andreu and Benedí, 1997). Our task now is to show that G can be used to build a CW-grammar W such that G is its underlying PCFG and W is equivalent to B.

In order to define W we have to define its set of rules and its set of meta-rules. The

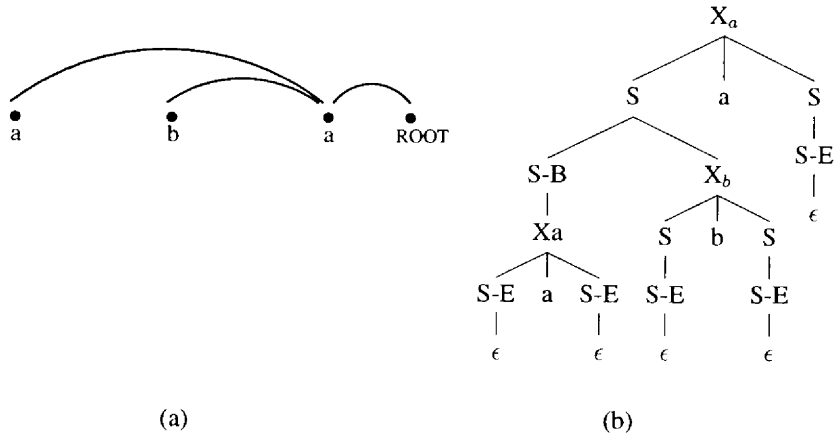


Figure 4.7: The dependency tree in (a) is transformed into the context free tree in (b).

set of pseudo-rules is given by the set $\{X_a \rightarrow_1 -s-a-s-\}$ while its set of meta-rules is given by the set of rules in G_A. It is immediate from the tree transformation function pictured in Figure 4.7 that W is strongly equivalent to B, moreover, it is immediate that both grammars assign the same probability value to all trees in their tree languages. Consequently, since G is consistent, W is consistent. Finally, since W is consistent, B is consistent as desired.

4.2.2 Markovian Context Free Grammars

In this subsection we capture one of the models presented by Collins: his so-called first model. The main idea behind (Collins, 1997, 1999) is to extend what he calls a "simple" CFG to a lexicalized back-off grammar.

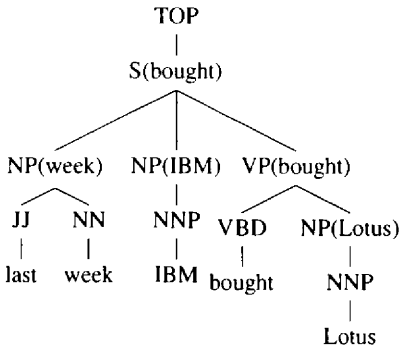
Background

Collins's first model may be viewed as a way to describe the probabilities assigned to CF-like rules. A rule has the following shape:[2]

$$P(h) \rightarrow L_n(l_n) \dots L_1(l_1) H(h) R_1(r_1) \dots R_m(r_m), \tag{4.1}$$

[2]In this subsection, we follow Collins's notion and denote the parent with P.

where H is the head-child of the phrase, and thus inherits the head word h from its parent P, and where $L_n(l_n)$, ..., $L_1(l_1)$ and $R_1(r_1)$, ..., $R_m(r_m)$ are left and right modifiers of H, respectively. Either or both of n and m may be zero, so that $n = m = 0$ for unary rules. Figure 4.8 shows a tree with its respective rules.




```
TOP → S(bought)
S(bought) → NP(week) NP(IBM) VP(bought)
NP(week) → JJ(last) NN(week)
NP(IBM) → NNP(IBM)
VP(bought) → VBD(bought) NP(Lotus)
NP(Lotus) → NNP(Lotus)
```

Figure 4.8: A lexicalized parse tree and the rules it contains; POS tags omitted.

Collins defines the probability of a rule such as (4.1) as the probability of its right-hand side, conditioned on the probability of its left-hand side, which is then decomposed as follows:

$$\mathcal{P}(L_n(l_n)\dots L_1(l_1)H(h)R_1(r_1)\dots R_m(r_m)|P(h)) = \mathcal{P}_h(H|P(h))\times$$
$$\times \prod_{i=1,\dots,n+1} \mathcal{P}_l(L_i(l_i)|L_1(l_1),\dots,L_{i-1}(l_{i-1}),P(h),H)\times$$
$$\times \prod_{j=1,\dots,m+1} \mathcal{P}_r(R_j(r_j)|L_1(l_1),\dots,L_{n+1}(l_{n+1}),R_1(r_1),\dots,R_{j-1}(r_{j-1}),P(h),H),$$

where l_{n+1} and r_{n+1} are defined as STOP. Collins approximates the probabilities using Markov independence assumptions for each order. In particular, the generation of the right-hand side of a rule such as (4.1), given the left-hand side, is decomposed into three steps:

1. The head constituent label of the phrase is generated, with probability equal to $\mathcal{P}_H(H|P,h)$.

2. Modifiers to the left of the head are generated, with probability equal to

$$\prod_{i=1\ldots n+1} \mathcal{P}_L(L_i(l_i)|P,h,H),$$

where $L_{n+1}(l_{n+1}) = \texttt{STOP}$. The STOP symbol is a non-terminal, and the model stops generating left modifiers as soon as it has been generated.

3. Modifiers to the right of the head are generated, with probability equal to

$$\prod_{i=i\ldots m+1} \mathcal{P}_R(R_i(r_i)|P,h,H),$$

where $R_{m+1}(r_{m+1})$ is defined as STOP.

We can think of the probabilities $\mathcal{P}_R(R_i(r_i)|P,h,H)$ and $\mathcal{P}_L(L_i(l_i)|P,h,H)$ as the probabilities assigned to arcs labeled $R_i(r_i)$ and $L_i(l_i)$ respectively in a zero-order Markov chain M. M has one state and as many arcs as combinations of symbols $R_i(r_i)$ and $L_i(l_i)$. M also determines the probability to be assigned to rules.

Markovian Context Free Grammars as CW-grammars

Recall that for capturing bilexical grammars, we first described the formalism using regular languages and later added probabilities. To capture Collins's first model we proceed in the opposite direction. We use the zero-order Markov models Collins builds to define regular languages and use these to build a CW grammar corresponding to Collins's model.

Independent of their order, Markov chains describe a regular language. Let $M = (S, P, F, I)$ be a Markov chain, where S is a sequence of states, P is the transition matrix, $F \subseteq S$ is the set of absorbing states, and I is the initial distribution of probabilities. We can directly transform M into an automaton A_M by taking S as the states of the automaton, F as the set of final states, and the initial state as the state that receives an initial probability mass.

Let NT be the set of possible phrase names, e.g., NP, PP, etc.; let W be the set of words in the lexicon; we assume that both sets are finite. For each pair $(H,w) \in NT \times W$ there are two Markov chains $r_{(H,w)}$ and $l_{(H,w)}$, such that Collins' rules can be rewritten as (4.1) as follows:

$$(P,h) \rightarrow (L_n,l_n)\ldots(L_1,l_1)(H,h)(R_1,r_1)\ldots(R_m,r_m).$$

For strings $(H,h)(L_1,l_1)\ldots(L_n,l_n)$ and $(H,h)(R_1,r_1)\ldots(R_m,r_m)$ the probability is given by the probabilities assigned to paths

$$(L_1,l_1)\ldots(L_n,l_n)\texttt{STOP}$$

and

$$(R_1, r_1) \dots (R_m, r_m)\texttt{STOP}$$

in the Markov chains $l_{(H,h)}$ and $r_{(H,h)}$ respectively.

4.2.4. DEFINITION. Let $B = (NT, W, \{l_{(H,w)}\}_{w \in W, h \in NT}\}, \{r_{(H,w)}\}_{w \in W, h \in NT})$ be a grammar based on Markov rules. Let $W_B = (V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ be the CW-grammar defined as follows:

- The set of variables V is given by the set of start symbols $S_{l_{(H,w)}}$ and $S_{r_{(H,w)}}$ from the regular grammars $l_{(H,w)}$ and $r_{(H,w)}$ respectively for each w in W.
- The set of non-terminals NT is some set in 1-1-correspondence with W, e.g., it can be defined as $NT = \{w' : w \in W\}$, where $\_'$ is a string priming operation.
- The set of terminals T is the set of words W.
- The set of probabilistic meta-rules is given by the union of the rules in each of the right and left regular grammars (i.e., the set given by $\mathcal{A} \xrightarrow{m} \alpha$ iff $A \to \alpha \in l_{(H,w)}$ or $\mathcal{A} \to \alpha \in r_{(H,w)}$ for some H and some w) plus the set $\{w' \xrightarrow{m} w : w \in W\}$.
- The set of pseudo-rules is given by $(P, h) \xrightarrow{s}_{\mathcal{P}_H(H|(P,h))} S_{l_{(H,h)}}(H, h)S_{r_{(H,h)}}$.

4.2.5. LEMMA. *Markov rules as used in Collins's first model are f-transformable to CW-grammars.*

The proof of this lemma is similar to the proof of Lemma 4.2.3 and we omit it here.

Expressive Power and Consistency

By Lemma 4.2.5 Collins's first model is weakly equivalent to context free grammars. Moreover, the idea behind Example 4.1.2 can be used to show that Collins's first model is not strongly equivalent to CFGs: Collins's first model can produce flat structures that cannot be captured by PCFGs (see Example 4.6 and Example 4.1.2 for examples of flat structures non-capturable by CFGs); as a consequence, the probabilistic version of Collins's first model cannot be captured using PCFGs.

As a consequence of Lemma 4.2.5, learning Markov rule-based grammars is equivalent to learning PCW-grammars, which, in turn, is equivalent to learning the PCFGs underlying PCW-grammars. Collins (1997) assumes that all hidden derivations are produced by Markov chains. Under the PCW-paradigm, his methodology is equivalent to transforming all trees in the training material by making all their hidden derivations visible and inducing the underlying PCFG from the transformed trees. Variables in the equivalent PCW-grammar are defined according to the degree of the Markov chain.

If the Markov chain used is of degree zero, there is only one variable (the Markov chain contains a unique state), and the induced Markov rule-based grammar is consistent. This consistency result follows from the fact that inducing a zero-degree Markov chain is the same as inducing the underlying PCFG in the equivalent PCW-grammar using maximum likelihood estimation, plus the fact that using maximum likelihood for inducing PCFGs produces consistent grammars (Chi and Geman, 1998; Joan-Andreu and Benedí, 1997).

Finally, the embedding of Collins's and Eisner's models into CW-grammars allows us to compare them. Both models have quite similar meta-rules and pseudo-rules. The main difference between their rules is that Collins's codify more information as variables. The embedding into CW-grammars allows us to see their learning step as a treebank rewriting procedure, followed by a CFG induction step. The treebank generated in the first step is the one used as training material in the second. According to this perspective, the two approaches differ in the rewriting function they use in the first step, while both approaches use maximum likelihood in the second step.

4.2.3 Stochastic Tree Substitution Grammars

Data-oriented parsing (DOP) is a memory-based approach to syntactic parsing. The basic idea is to use the subtrees from a syntactically annotated corpus directly as a stochastic grammar. The DOP-1 model (Bod, 1995) was the first version of DOP, and most later versions of DOP are variations on it. The underlying grammatical formalism is stochastic tree substitution grammars (STSG), which is the grammatical formalism we capture here.

Background

The grammatical formalism is extremely simple and can be described as follows: for every sentence in a parsed training corpus, extract every subtree. Now, we use these trees to form a stochastic tree substitution grammar. Formally, a *stochastic tree-substitution grammar* (STSG) G is a 5-tuple $\langle V_N, V_T, S, R, P\rangle$ where:

- V_N is a finite set of nonterminal symbols.
- V_T is a finite set of terminal symbols.
- $S \in V_N$ is the distinguished symbol.
- R is a finite set of trees, called *elementary trees*, whose top nodes and interior nodes are labeled by nonterminal symbols and whose yield nodes are labeled by terminal or nonterminal symbols.

- P is a function which assigns to every elementary tree $t \in R$ a probability $P(t)$. For a tree t with a root node symbol $root(t) = \alpha$, $P(t)$ is interpreted as the probability of substituting t for a node α. We require, therefore, for a given α that $\sum_{\{t:root(t)=\alpha\}} = 1$ (where t's root node symbol is α).

If t_1 and t_2 are elementary trees such that the left-most non-terminal leaves node symbol of t_1 is equal to the root node symbol of t_2, then $t_1 \circ t_2$ is the tree that results from substituting t_2 in this left-most non-terminal leaves node symbol in t_1. The partial function $\circ$ is called *leftmost substitution* or simply *substitution*. Trees are derived using leftmost substitution.

STSGs as CW-grammars

A STSG is not a context free grammar. The main difference, and the hardest to capture in a CFG-like setting, is the way in which probabilities are computed for a given tree. The probability of a tree is given by the sum of the probabilities of all derivations producing it. CW-grammars offer a similar mechanism: the probability of the body of a w-rule is the sum of the probabilities of all meta-derivations producing it. The idea of the equivalence is to associate to every tree produced by a STSG a w-rule of the PCW-grammar in such a way that the body of the w-rule codifies the whole tree.

To implement this idea, we need to code up trees as strings. The simplest way to achieve this is to visit the nodes in a depth-first left-to-right order. For each inner node, we write the CFG production, while for the leaves, we write the symbol itself if the symbol is a terminal and a primed version of it if the symbol is a non-terminal. For example, the code describing the tree in Figure 4.9(a) is

$$(A, BAB)B'(A, BAB)B'A'B'(B, a)a.$$

The first step in capturing STSGs is to build meta-rules capturing elementary trees using the notation just introduced. Specifically, let t be an elementary tree belonging to a STSG. Let S be its root and α its string representation. The CF-like rule $S' \rightarrow \alpha$ is called the *elementary rule* of t. Elementary rules store all information about the elementary tree. They have primed non-terminals where a substitution can be carried out. For example, if t is the elementary tree pictured in Figure 4.9.(b), its elementary rule is $S' \rightarrow (S, AB)(A, B)(A, ab)ab(B, AC)(A, ab)abC'$. Note the primed version of C in the frontier of the derivation.

4.2.6. DEFINITION. Let $H = (V_N, V_T, S, R, P)$ be a STSG. Let $W_H = (V, NT, T, S', \xrightarrow{m}, \xrightarrow{s})$ be the following CW-grammar.

- V is the primed version of V_T.

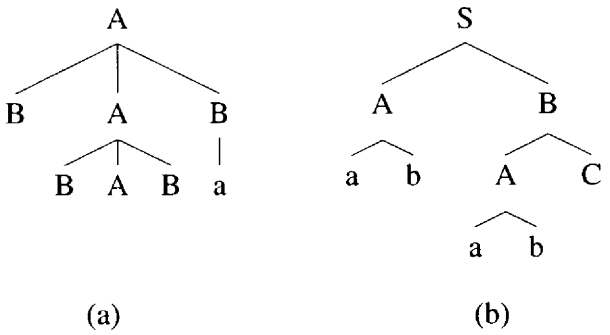


Figure 4.9: (a) A derivation tree. (b) An elementary tree.

- (A, α) is in NT iff $(A, \alpha) \rightarrow \epsilon$ appears in some elementary tree.
- T is exactly as V_T.
- S' is a new symbol.
- The set of meta-rules is built by transforming each elementary tree into its corresponding elementary rule.
- The set of pseudo-rules is given by $(A, \alpha) \xrightarrow{s} \epsilon$ if $A \rightarrow \alpha$ appears in a elementary tree, plus rules $S' \xrightarrow{s} S$.

Two remarks: first, all generative capacity is encoded in the set of meta-rules. In the CW-world, the body of a rule (i.e., an instantiated pseudo-rule) encodes a derivation of the STSG. Second, the probability of a w-rule is the sum of the probabilities of meta-derivations yielding the rule's body.

4.2.7. LEMMA. *Let $H = (V_N, V_T, S, R, P)$ be a STSG and W_H the CW-grammar for H as given in Definition 4.2.6. There is a one-to-one correspondence between derivations in H and meta-rule derivations in W_H.*

PROOF. Let t be a tree produced by H. We prove the lemma using induction on the length of the derivation producing t. If t has length 1, there is an elementary tree t_1 such that S is the root node and yields α, which implies that there is a meta-rule obtained from the elementary rule corresponding to the elementary tree t_1. The relation is one-to-one as, by definition, meta-rules are in one-to-one correspondence with elementary trees.

Suppose the lemma is true for derivation lengths less than or equal to n. Suppose t is generated by a derivation of length $n+1$. Assume there are trees t_1, t_2 with $t_1 \circ t_2 = t$. By definition there is a unique meta-rule r_1 corresponding with t_1 and by the inductive hypothesis there is a unique derivation for t_2. ⊣

4.2.8. LEMMA. *Let $H = (V_N, V_T, S, R, P)$ be a STSG, and W_H the CW-grammar for H as given in Definition 4.2.6. Then W_H accepts the same set of strings as H, i.e., STSGs and PCW-grammars are weakly equivalent.*

PROOF. Let α be a string in $L(H)$. There is at least one tree derivation $t_1 \circ \ldots \circ t_k$ yielding α. From Lemma 4.2.7 we know that there is a w-rule $S' \stackrel{w}{\Longrightarrow} \alpha$ such that after applying rules $(A, \beta) \stackrel{s}{\longrightarrow} \epsilon, \alpha$ is obtained. ⊣

4.2.9. COROLLARY. *Let $H = (V_N, V_T, S, R, P)$ be a STSG and W_H the CW-grammar for H as given in Definition 4.2.6. There is a one-to-one correspondence between derivations in H and W_H.*

4.2.10. COROLLARY. *STSGs are f-transformable to CW-grammars.*

PROOF. The result is a direct consequence of Lemma 4.2.8 and Lemma 4.2.9. ⊣

4.2.11. LEMMA. *Let $H = (V_N, V_T, S, R, P)$ be a STSG, and let W_H be the CW-grammar given in Definition 4.2.6. Both grammars assign the same probability mass to trees related through the one-to-one mapping described in Corollary 4.2.9.*

PROOF. A tree has a characteristic w-rule, defined by its shape. In other words, the probability of a w-rule according to the definition of PCW-grammars is given by the sum of the probabilities of all derivations producing the rule's body, i.e., all STSG derivations producing the same tree. As a consequence, a particular STSG tree, identified by the body of the corresponding w-rule, has the same probability mass as the amount assigned to its corresponding w-rule by the equivalent CW-Grammar. ⊣

Expressive Power and Consistency

By Corollary 4.2.10, STSGs are weakly equivalent to context free grammars. The consistency of a STSG depends on the methodology used for computing the probabilities assigned to its elementary trees. DOP-1 is one particular approach to computing these probabilities. Under the DOP-1 perspective, a tree t contributes all its possible subtrees to a new treebank from which the probabilities of elementary trees are computed. Probabilities of an elementary tree are computed using maximum likelihood. Since

the events in the new treebank are not independently distributed, the resulting probabilities are inconsistent and biased (Johnson, 2002). Solutions taking into account the dependence between trees in the resulting treebanks have been suggested (Sima'an and Buratto, 2003).

Consistency conditions cannot be derived for the DOP-1 estimation procedure because it does not attempt to learn the underlying PCFG. In fact, our formalism suggests that probabilities should be computed differently than they are done in DOP-1. By our embedding, a tree t in the treebank corresponds to the body of a pseudo-rule instantiated through meta-derivations; t is the final "string" and does not have any information about the derivation that took place. But viewing t as a final string changes the problem definition! Now, we have as input a set of elementary rules and a set of accepted trees. The problem is to compute probabilities for these rules: an unsupervised problem that can be solved using any unsupervised technique, e.g., (Carroll and Charniak, 1992; Chen, 1995). The consistency of the resulting STSG depends on the consistency properties of the unsupervised method.

4.3 Discussion and Conclusion

In this chapter we introduced constrained W-grammars and we augmented them with probabilities. Probabilities provide a disambiguation mechanism and a statistical perspective. It is still an open question whether probabilities add expressive power to CW-grammars; the strategy used in Chapter 3 can not be used for PCW-grammars. To see this, one should note that CW-grammars do not have any inherently ambiguous languages: any CW-grammar W with start symbol S, meta-rules M and pseudo-rules P can be transformed into an unambiguous grammar G' by defining G''s start symbol to be S', its unique meta-rule to be $S' \xrightarrow{m} S$ and its meta-rules to be $M \cup P$. G' is unambiguous because it hides inside meta-derivations practically all derivations. All trees in $T(G')$ are very flat, and there is exactly one tree for each string in the language.

Do PCW-grammars add anything? We think that the importance and contribution of our grammatical formalism has to be considered from the perspective of grammatical formalisms underlying state-of-the-art parsers. It is usually the case that grammatical formalisms used in parsers are not clearly stated, they are hard to identify from the definition of the parser (see (Bikel, 2004) for an account of the details of Collins's parser) and their formal properties require ad-hoc proofs (see (Bod, 1998) for expressive power properties of STSGs) or are not proven at all. We have shown that PCW-grammars provide a common formalism based on a well-known grammatical framework with computational properties that are very well understood. Clearly, PCW-grammars greatly reduce the expressive power of W-grammars, but still, we man-

age to capture the grammatical formalisms underlying state-of-the-art parsers with the remaining expressive power and to establish new and important facts about them.

In particular, we examined the expressive power of three formalisms (bilexical grammars, Markovian context free rules, and stochastic tree substitution grammars) together with some conditions under which the inferred grammars are consistent. Finally, we should point out that, despite their similarities, there is a fundamental difference between PCW-grammars and PCFGs, and this is the two-level mechanism of the former. This mechanism allows us to capture three state-of-the-art natural language parsers, which cannot be done using standard PCFGs only.

The results in this chapter shed light on the relationship between a number of grammatical formalisms, not just between context free grammars and PCW-grammars. In particular, we have shown that, from a formal perspective, bilexical grammars and Markovian context free grammars do not differ in a principled way: both are based on approximating bodies of rules using Markov models. We also found that STSGs and Markov rules have certain similarities. Markov rules and STSGs suppose that rule bodies are obtained by collapsing hidden derivations. That is, for Markov rules a rule body is a regular expression (or equivalently a Markov chain). Similarly, STSGs take this idea to the extreme by taking the whole sentence to be the yield of a hidden derivation. PCW-grammars naturally suggest intermediate levels of abstraction; in Chapter 5 we show that these levels can be used to reduce the size of grammars induced from treebanks, and, hence, to optimize parsing procedures.

From a theoretical point of view, the concept of f-transformable grammars, which we use heavily in our proofs, is a novel and very powerful concept that relaxes the known equivalence notions between grammars. Since arbitrary functions f can be defined between arbitrary tree languages and CFG-like trees, they can be used to map other formalisms like tree adjoining grammars (Joshi, 1985) or categorial grammars (Wood, 1993) to context free trees. As part of our future research, we aim to capture further grammatical formalisms and to characterize the nature of the functions f used to achieve this.

Chapter 5

Alternative Approaches for Generating Bodies of Grammar Rules

5.1 Introduction

So far, we have developed a grammatical formalism capable of capturing different state-of-the-art language models, which gave us a novel perspective on state-of-the-art language models. In Chapter 4 we identified that some language models use n-grams for building bodies of rules. From the literature, we know that n-grams have had a big impact on the state-of-the-art in natural language models. They are central to many language models (Charniak, 1997; Collins, 1997; Eisner, 1996; Collins, 2000), and despite their simplicity, n-gram models have been very successful. Modeling with n-grams is an induction task (Gold, 1967): given a sample set of strings, the task is to guess the grammar that produced that sample. Usually, the grammar is not chosen from an arbitrary set of possible grammars, but from some given restricted class. Grammar induction consists of two parts: choosing the class of languages amongst which to search and designing the procedure for performing the search. By using n-grams for grammar induction one addresses the two parts in one go, and the use of n-grams implies that the solution will be searched for in the class of probabilistic regular languages, since n-grams induce probabilistic automata and, consequently, probabilistic regular languages. But probabilistic regular languages induced using n-grams form a proper subclass of the class of all probabilistic regular languages; for instance, n-grams are incapable of capturing long-distance relations between words. At the technical level the restricted nature of n-grams is witnessed by the special structure of the automata induced from them, as we will see in Section 5.4.2.

N-grams are not the only way to induce regular languages, and they are not the most powerful way to do so. There is a variety of general methods capable of inducing

all regular languages (Denis, 2001; Carrasco and Oncina, 1994; Thollard et al., 2000). What is their relevance for natural language parsing? Recall from Chapter 4 that regular languages are used for describing the bodies of rules in a grammar. Consequently, the quality and expressive power of the resulting grammar is tied to the quality and expressive power of the regular languages used to describe them. And these properties, in turn, are influenced directly by the method used to induce them. At this point a natural question arises: can we gain anything in parsing from using general methods for inducing regular languages instead of methods based on n-grams? Specifically, can we describe the bodies of grammatical rules more accurately and more concisely by using general methods for inducing regular languages?

In the context of natural language parsing we present an empirical comparison between algorithms for inducing regular languages using n-grams on the one hand, and more general algorithms for learning the general class of regular language on the other. We proceed as follows. We generate our training data from the Wall Street Journal section of the Penn Tree Bank (PTB), transforming it to *projective* dependency structures, following (Collins, 1996). Since projective dependency structures can be seen as a special type of context free grammars (Gaifman, 1965), word dependents can be seen as bodies of context free rules. We extract these rule bodies and use them as training material for the rule induction algorithms we consider. The automata produced this way are then used to build grammars which, in turn, are used for parsing.

We are interested in two aspects of the use of probabilistic regular languages for natural language parsing: the quality of the induced automata and the performance of the resulting parsers. For evaluation purposes, we use two metrics: perplexity for the first aspect and percentage of correct attachments for the second, both explained in detail in Section 2.2.3. The main results of the chapter are that, measured in terms of perplexity, automata induced by algorithms other than n-grams describe rule bodies better than automata induced using n-gram-based algorithms. Moreover, the gain in automata quality is reflected by an improvement in parsing performance. The parsing performance of both methods (n-grams vs. general automata) can be substantially improved by splitting the training material into POS categories. As a side product, we find empirical evidence to explain the effectiveness of rule lexicalization (Collins, 1997; Sima'an, 2000) and parent annotation techniques (Klein and Manning, 2003) in terms of a reduction in perplexity in the automata induced from training corpora.

5.2 Overview

We want to build grammars using different algorithms for inducing their rules. Our main question is aimed at understanding how different algorithms for inducing regular

languages impact the parsing performance with those grammars. A second issue that we want to explore is how the grammars perform when the quality of the training material is improved, that is, when the training material is separated into part of speech (POS) categories before the regular language learning algorithms are run.

The first step is to transform the PTB into projective dependencies structures following (Collins, 1996). From the resulting tree bank we delete all lexical information except POS tags. Every POS in a tree belonging to the tree-bank has associated to it two different, possibly empty, sequences of right and left dependents, respectively. We extract all these sequences for all trees, producing two different sets containing right and left sequences of dependents, respectively.

These two sets form the training material used for building four different grammars. The four grammars differ along two dimensions: the number of automata used for building them and the algorithm used for inducing the automata. As to the latter dimension, in Section 5.4 we use two algorithms: the Minimum Discriminative Information (MDI) algorithm, and a bigram-based algorithm. As to the former dimension, two of the grammars are built using only two different automata, each of which is built using the two sample set generated from the PTB. The other two grammars are built using two automata per POS, exploiting a split of the training samples into multiple samples, two samples per POS, to be precise, each containing only those samples where the POS appeared as the head.

5.3 From Automata to Grammars

In this section we describe how to learn PCW-grammars from the automata that we are going to induce in Section 5.4. Since we will induce two families of automata ("Many-Automata" where we use two automata per POS, and "One-Automaton" where we use only two automata to fit every POS), we need to describe two automata-to-grammar transformations.

Let us start with the case where we build two automata per POS. Let w be a POS in the PTB; let A_L^w and A_R^w be the two automata associated to it. Let G_L^w and G_R^w be the PCFGs equivalent to A_L^w and A_R^w, respectively, following (Abney et al., 1999), and let S_L^w and S_R^w be the start symbols of G_L^w and G_R^w, respectively. We build our final grammar G with start symbol S, by defining its meta-rules as the disjoint union of all rules in G_L^w and G_R^w (for all POS w), its set of pseudo-rules as the union of the sets

$$\{W \xrightarrow{s}_1 S_L^w w S_R^w\}$$

and

$$S \xrightarrow{s}_1 S_L^w w S_R^w\},$$

where W^w is a unique new variable symbol associated to w.

When we use two automata for all parts of speech, the grammar is defined as follows. Let A_L and A_R be the two automata learned. Let G_L and G_R be the PCFGs equivalent to A_L and A_R, and let S_L and S_R be the start symbols of G_L and G_R, respectively. Fix a POS w in the PTB. Since the automata are deterministic, there exist states S_L^w and S_R^w that are reachable from S_L and S_R, respectively, by following the arc labeled with w. Define a grammar as in the previous case. Its start symbol is S, its set of meta-rules is the disjoint union of all rules in G_L^w and G_R^w (for all POS w), its set of pseudo-rules is $\{W^w \xrightarrow{s}_1 S_L^w w S_R^w, S \xrightarrow{s}_1 S_L^w w S_R^w : w$ is a POS in the PTB and W^w is a unique new variable symbol associated to $w\}$.

5.4 Building Automata

The four grammars we intend to induce are completely defined once the underlying automata have been built. We now explain how we build those automata from the training material. The process of building the automata consists of three steps:

1. Extracting the training material from the transformed PTB.
2. Applying the algorithm to learn automata to the training material.
3. Searching for the optimal automata.

Sections 5.4.1, 5.4.2 and 5.4.3 deal with steps 1, 2, and 3, respectively.

5.4.1 Building the Sample Sets

The training material is obtained as follows. We transform the PTB, sections 2–22, to dependency structures, as suggested by (Collins, 1999). All sentences containing CC tags are filtered out, following (Eisner, 1996). We also eliminate all word information, leaving only POS tags. For each resulting dependency tree we extract a sample set of right and left sequences of dependents. Figure 5.1 shows an example of a phase structure, Figure 5.2 shows its corresponding dependency tree, and Table 5.1 shows the sample sets of right and left dependents we extracted from it. The sample set used for automata induction is the union of all individual tree sample sets.

5.4.2 Learning Probabilistic Automata

Probabilistic deterministic finite state automata (PDFA) inference is the problem of inducing a stochastic regular grammar from a sample set of strings belonging to an

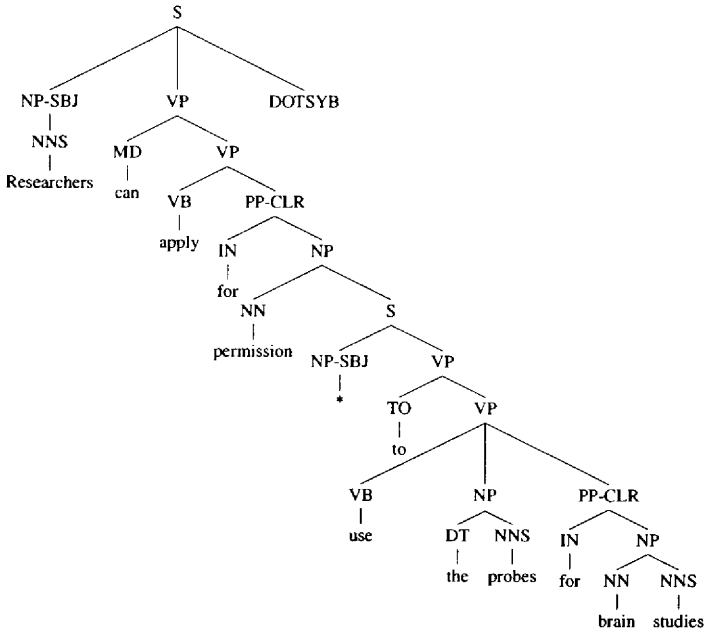


Figure 5.1: Tree extracted from the PTB, Section 02, file `wsj_0297.mrg`.

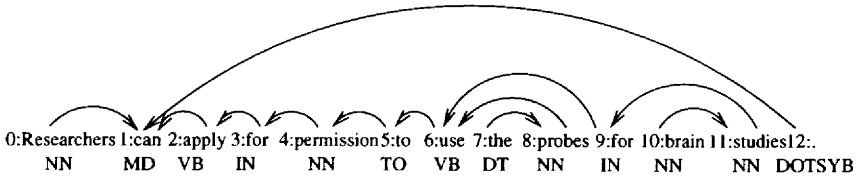


Figure 5.2: Dependency structure corresponding to the tree in Figure 5.1.

Word Position	Word's POS	Left	Right
0	NN	NN	NN
1	MD	MD NN	MD VB DOTSYB
2	VB	VB	VB IN
3	IN	IN	IN NN
4	NN	NN	NN TO
5	TO	TO	TO VB
6	VB	VB	VB NN IN
7	DT	DT	DT
8	NN	NN DT	NN
9	IN	IN	IN NN
10	NN	NN	NN
11	NN	NN NN	NN
12	DOTSYB	DOTSYB	DOTSYB

Table 5.1: Bags of left and right dependents. Left dependents are to be read from right to left.

unknown regular language. The most direct approach for solving the task is by using n-grams. The n-gram induction algorithm works as follows. It starts with an empty automaton, i.e., an automaton whose set of states and set of arcs are empty. It adds states and arcs to the initial automaton as follows. It adds a state to the current automaton for each sequence of symbols β of length n that appear in the training material. It adds an arc between states $a\beta$ and βb labeled b to the current automaton, if the sequence $a\beta b$ appears in the training set.

The probability assigned to the arc $(a\beta, \beta b)$ is defined as the number of times the sequence $a\beta b$ appeared in the training set divided by the number of times $a\beta$ was followed by any other character appeared in the training set. Note that a n STOP symbols have to be postfixed to each string in the training material before the induction algorithm is used. The role and importance of the STOP symbol is discussed in detail in Appendix B.

Clearly, the size and quality of the automata produced by the n-gram based algorithm depends on n. For our experiments, we chose n equal to 2 because n equal to 1 and n greater than 2 produce automata of very low quality. For n equal to 1 the automata are uncapable of learning dependencies between words. For n greater than 2 the algorithm immediately suffers from data sparseness, because the automata model bodies of rules which tend to be very short. Consequently, for the remainder of the chapter, we take n-grams to be bigrams. Note also that we do no use any smoothing technique that would make it possible to use other values of n and at the same time

avoid the problems mentioned. If we want to carry out a fair comparison of both methods, smoothing techniques used for optimizing automata based on n-grams should also be used for optimizing MDI-based automata. Two or more n-gram based automata smoothed into a single automaton would have to be compared against two or more MDI-based automata smoothed into a single automaton. It would be hard to determine whether the differences between the final automata are due to the smoothing procedure or to the algorithms used for creating the initial automata. By leaving smoothing out of the picture, we obtain a clearer understanding of the differences between the two algorithms to induce automata.

There are other approaches to inducing regular grammars besides ones based on n-grams. The first algorithm to learn PDFAs was ALERGIA (Carrasco and Oncina, 1994); it learns cyclic automata with the so-called state-merging method. The Minimum Discrimination Information (MDI) algorithm (Thollard et al., 2000) improves over ALERGIA and uses Kullback-Leibler divergence for deciding when to merge states. We opted for the MDI algorithm as an alternative to n-gram based induction algorithms, mainly because its working principles are radically different from the n-gram-based algorithm. The MDI algorithm first builds an automaton that only accepts the strings in the sample set by merging common prefixes, thus producing a tree-shaped automaton in which each transition has a probability proportional to the number of times it is used while generating the positive sample; see Section 2.2.2 for details.

For comparison purposes, let us repeat here the working principle of the MDI algorithm. The MDI algorithm traverses the lattice of all possible partitions for this general automaton, attempting to merge states that satisfy a trade-off that can be specified by the user. Specifically, assume that A_1 is a temporary solution of the algorithm and that A_2 is a tentative new solution derived from A_1. $\Delta(A_1, A_2) = D(A_0||A_2) - D(A_0||A_1)$ denotes the divergence increment while going from A_1 to A_2, where $D(A_0||A_i)$ is the *Kullback-Leibler divergence* or *relative entropy* between the two distributions generated by the corresponding automata (Cover and Thomas, 1991). The new solution A_2 is *compatible* with the training data if the divergence increment relative to the size reduction, that is, the reduction of the number of states, is small enough. Formally, let `alpha` denote a compatibility threshold; then the compatibility is satisfied if $\frac{\Delta(A_1,A_2)}{|A_1|-|A_2|} <$ `alpha`. For this learning algorithm, `alpha` is the unique parameter; we tuned it to get better quality automata.

Note that the working principles of the n-gram-based algorithm and the MDI algorithm are completely different. The n-gram based algorithm works locally by adding arcs and states depending on the local configuration of strings in the sample set. In contrast, the MDI algorithm starts by building an automaton that accepts only the strings in the training sample and it over-generalizes over the sample set by merging states that it considers that can be merged. In this way, the MDI algorithm is capable of detecting

long distance dependencies between symbols.

5.4.3 Optimizing Automata

We use three measures to evaluate the quality of a probabilistic automaton (and set the value of `alpha` optimally). The first two come from come Section 2.2.2 Let Q be a test bag extracted as T. We use perplexity (PP) and missed samples (MS) to evaluate the quality of a probabilistic automaton. The PP and MS measures are relative to a test sample Q; as described in section 5.4.1, we transformed section 00 of the PTB to obtain one. The third measure we use to evaluate the quality of automata concerns the size of the automata. We compute NumEdges and NumStates, that is, the number of edges and the number of states of the automaton.

We say that one automaton is of a *better quality* than another if the values of the 4 indicators—PP, MS, NumEdges, and NumStates—are lower for the first than for the second. Our aim is to find a value of `alpha` that produces an automaton of better quality than the bigram-based counterpart.

By exhaustive search, we determined the optimal value of `alpha`. We selected the value of `alpha` for which the MDI-based automaton outperforms the bigram-based one. An equivalent value of `alpha` can be obtained independently of the performance of the bigram-based automata by defining a measure that combines PP and MS. This measure should reach its maximum when PP and MS reach their minimums, see Chapters 6 and 7 for definitions of such functions.

We exemplify our optimization procedure by considering automata for the "One-Automaton" setting (where we used the same automata for all parts of speech). In Figure 5.3 we plot all values of PP and MS computed for different values of `alpha`, for each training set (i.e., left and right). That is, we fix a value of `alpha`, feed the MDI algorithm with the training material (sections 2–22 of the PTB), and compute PP and MS for the resulting automaton using the test sample (section 0 of the PTB).

From the plots we can identify values of `alpha` that produce automata having better values of PP and MS than the bigram-based ones. All such `alphas` are the ones inside the marked areas (between $5e-05$ and 0.00012 for the left side and $5e-05$ and 0.0001 for the right side); all automata induced using those `alphas` possess a lower value of PP as well as a smaller number of MS, as required. Based on these

	MDI		Bigrams	
	Right	Left	Right	Left
NumEdges	268	328	20519	16473
NumStates	12	15	844	755

Table 5.2: Automata sizes for the "One-Automaton" case, with `alpha` $= 0.0001$.

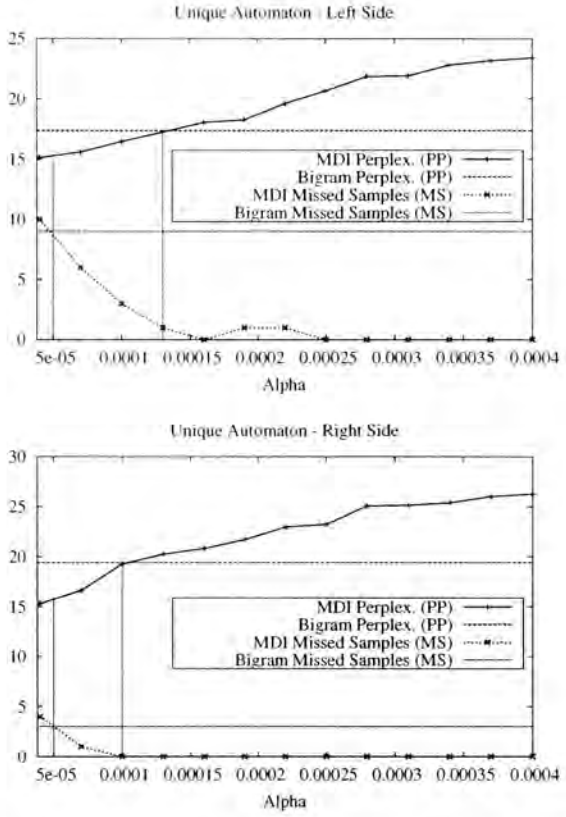


Figure 5.3: Values of PP and MS for automata used in building One-Automaton grammars. (X-axis): `alpha`. (Y-axis): missed samples (MS) and perplexity (PP). The two constant lines represent the values of PP and MS for the bigram-based automata.

explorations we selected `alpha` $= 0.0001$ for building the automata used for grammar induction in the "One-Automaton" case. Besides having lower values of PP and MS, the resulting automata are smaller than the bigram based automata (Table 5.2). MDI compresses information better; the values in the tables suggest that MDI finds more regularities in the sample set than the bigram-based algorithm.

To determine optimal values for the "Many-Automata" case (where we learned two automata for each POS) we used the same procedure as for the "One-Automaton" case, but now for every individual POS. We do not reproduce analogues of Figure 5.3 and Table 5.2 for all parts of speech but in Figure 5.4 we show some representative plots;

Besides allowing us to find the optimal `alphas`, the plots provide us with a great

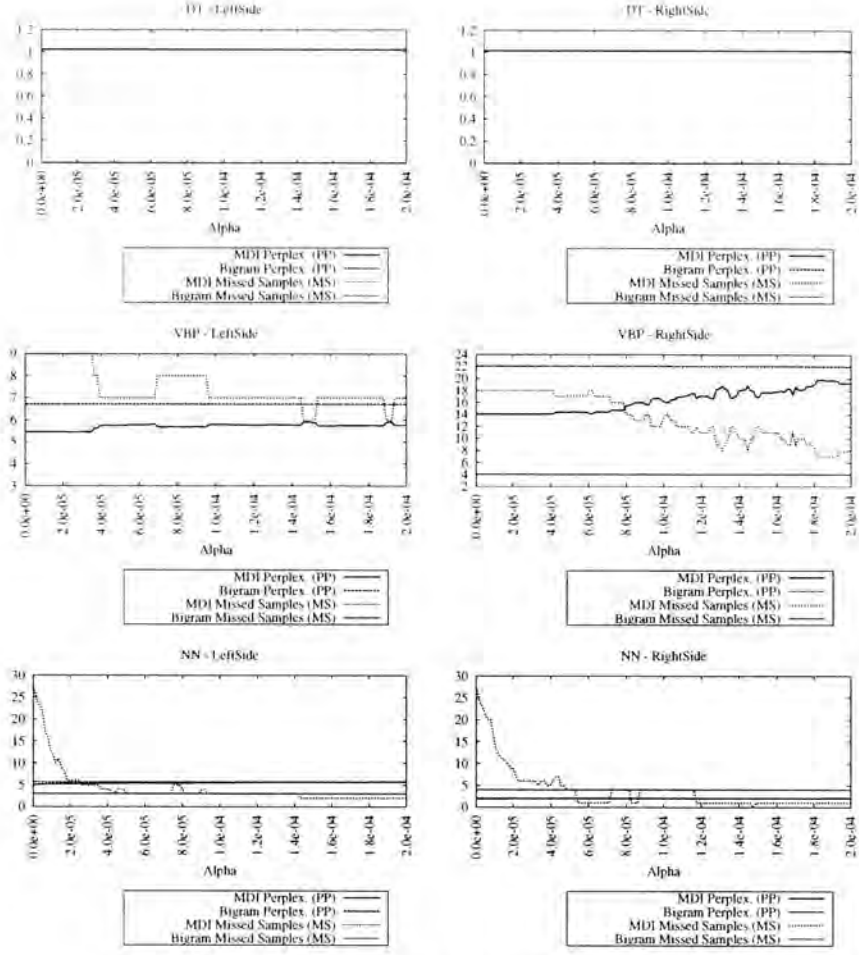


Figure 5.4: Values of PP and MS for different automata.

deal of information. The first point to note is that PP values for the one-automaton case are around 20 while they are around 7 for VBP and even lower for other POS (cf., Figures 5.3 and 5.4, second row). We think that such a notable difference exists because the training sets used for the many-automata case are much more homogeneous than the one used for the one-automaton. The one-automaton training set contains instances of regular expressions for all kind of phenomena. In contrast, the training material for many-automata has been split using POS information into more homogeneous classes. This is an important point and we come back to it in Chapter 6.

There are two remarkable things in the plots for VBP (Figure 5.4, second row). First, it is one of the few examples where the bigram-based algorithm outperforms the MDI algorithm. Second, the values of PP in this plot are relatively high and unstable compared to other plots. Lower perplexity usually implies better quality automata, and as we will see in the next section, better automata produce better grammars.

How can we obtain lower PP values for the automata associated to verbs? The class of words tagged with verbs tags, e.g., VBP, harbors many different behaviors, which is not surprising, given that verbs can differ widely in terms of, e.g., their subcategorization frames. One way to decrease the PP values is to split the class of words tagged with VBP into multiple, more homogeneous classes. One attempt to implement this idea is *lexicalization*: increasing the information in the POS tag by adding the lemma to it (Collins, 1997; Sima'an, 2000). Lexicalization splits the class of verbs into a family of singletons producing more homogeneous classes, as desired. A different approach (Klein and Manning, 2003) consists in adding head information to dependents; words tagged with VBP are then split into classes according to the words that dominate them in the training corpus. Following this strategy, in Chapter 6 we propose an algorithm for finding the optimal splitting of the training set, which we then use for splitting the training set corresponding to VB.

Some POS present very high perplexities, while other tags such as DT present a PP close to 1 (and 0 MS) for all values of `alpha`. Hence, there is no need to introduce further distinctions in DT, doing so will not increase the quality of the automata but will increase their number; for these particular cases, splitting techniques are bound to add noise to the resulting grammars. The plots also indicate that the bigram-based algorithm captures them as well as the MDI algorithm.

In Figure 5.4, third row, we see that the MDI-based automata and the bigram-based automata achieve the same value of PP (close to 5) for NN, but the MDI misses fewer examples for `alphas` bigger than $1.4e-04$. As pointed out, we built the

		MDI		Bigrams	
POS		Right	Left	Right	Left
DT	NumEdges	21	14	35	39
	NumStates	4	3	25	17
VBP	NumEdges	300	204	2596	1311
	NumStates	50	45	250	149
NN	NumEdges	104	111	3827	4709
	NumStates	6	4	284	326

Table 5.3: Automata sizes for three parts of speech in the "Many-Automata" case, with `alpha` = 0.0002 for all parts of speech.

One-Automaton-MDI using `alpha` = 0.0001 and even though the method allows us to fine-tune each `alpha` in the Many-Automata-MDI grammar, we used a fixed `alpha` = 0.0002 for all parts of speech, which, for most parts of speech, produces better automata than bigrams. Table 5.3 lists the sizes of the automata. The differences between MDI-based and bigram-based automata are not as dramatic as in the "One-Automaton" case (Table 5.2), but the former again have consistently lower NumEdges and NumStates values, for all parts of speech, even where bigram-based automata have a lower perplexity.

5.5 Parsing the PTB

We have just observed remarkable differences in quality between MDI-based and bigram-based automata. Next, we present the parsing scores, and discuss the meaning of the measures observed for automata in the context of the grammars they produce. The measure that translates directly from automata to grammars is automaton size. Since each automaton is transformed into a PCFG, the number of rules in the resulting grammar is proportional to the number of arcs in the automaton, and the number of non-terminals is proportional to the number of states. From Table 5.4 we see that MDI compresses information better: the sizes of the grammars produced by the MDI-based automata are an order of magnitude smaller that those produced using bigram-based automata. Moreover, the "One-Automaton" versions substantially reduce the size of the resulting grammars; this is obviously due to the fact that all POS share the same underlying automaton so that information does not need to be duplicated across parts of speech. We report the size of grammars *before* they are transformed to Chomskian normal form, i.e., the grammars contain rules with empty bodies and relabeling rules. Since our parser requires CNF grammars, we need to transform these grammars to CNF *before* parsing. For details about our parsing algorithms implementation see Appendix A.

One Automaton		Many Automata	
MDI	Bigram	MDI	Bigram
702	38670	5316	68394

Table 5.4: Numbers of rules in the grammars built.

To understand the meaning of PP and MS in the context of grammars it helps to think of PCW-parsing as a two-phase procedure. The first phase consists of creating the rules that will be used in the second phase. And the second phase consists of using the rules created in the first phase as a PCFG and parsing the sentence using a PCF parser.

PCW-grammars make this distinction clear, and the abstraction is also useful at this point. Since regular expressions are used to build rules, the values of PP and MS quantify the quality of the set of rules built for the second phase: MS gives us a measure of the number of rule bodies that should be created but that will not be created, and, hence, it gives us a measure of the number of "correct" trees that will not be produced. PP tells us how uncertain the first phase is about producing rules.

Finally, we report on the parsing accuracy. We use two measures, the first one (%Words) was proposed by Lin (1995) and was the one reported in (Eisner, 1996). Lin's measure computes the fraction of words that have been attached to the right word. The second one (%POS) marks as correct a word attachment if, and only if, the POS tag of the head is the same as that of the right head, i.e., the word was attached to the correct word-class, even though the word is not the correct one in the sentence. Clearly, the second measure is always higher than the first one. The two measures try to capture the performance of the PCW-parser in the two phases described above: (%POS) tries to capture the performance in the first phase, and (%Words) in the second phase. The measures reported in Table 5.5 are the mean values of (%POS) and (%Words) computed over all sentences in section 23 having length at most 20. We parsed only those sentences because the resulting grammars for bigrams are too big: parsing all sentences without any serious pruning techniques was simply not feasible.

	MDI		Bigrams	
	%Words	%POS	%Words	%POS
One-Automaton	0.69	0.73	0.59	0.63
Many-Automata	0.85	0.88	0.73	0.76

Table 5.5: Parsing results for the PTB.

From Table 5.5 we see that the grammars induced with the MDI algorithm outperform the grammars created with bigrams-based algorithm. Moreover, the grammars using different automata per POS outperforms the ones built using only a single automaton per side (left or right). The results suggest that an increase in quality of the automata has a direct impact on the parsing performance.

5.6 Related Work and Discussion

Modeling rule bodies is a key component of parsers. N-grams have been used extensively for this purpose (Collins, 1996, 1997; Eisner, 1996). In n-gram-based formalisms the generative process is not considered in terms of probabilistic regular languages. Considering them as such (like we do) has two advantages. First, a vast area of research for inducing regular languages (Carrasco and Oncina, 1994; Thollard et al.,

2000; Dupont and Chase, 1998) comes in sight. Second, the parsing device itself can be viewed under a unifying grammatical paradigm like PCW-grammars.

In our comparison we optimized the value of `alpha`, but we did not optimize the n-grams, as doing so would mean two different things. First, smoothing techniques would have to be used to combine different order n-grams. As pointed out, we would also have to smooth different MDI-based automata, which would leave us in the same point. Second, the degree of the n-gram. We opted for $n = 2$ as it seems the right balance of informativeness and generalization. In this chapter n-grams are used to model sequences of arguments, and these hardly ever have length > 3, making higher degrees useless. To make a fair comparison for the Many-Automata grammars we did not tune the MDI-based automata individually, but we picked a uniform `alpha`.

MDI presents a way to compress rule information on the PTB; of course, other approaches exists. In particular, Krotov et al. (1998) try to induce a CW-grammar from the PTB with the underlying assumption that some derivations that were supposed to be hidden were left visible. The attempt to use algorithms other than n-grams-based for inducing of regular languages in the context of grammar induction is not new; for example, Kruijff (2003) uses profile hidden models in an attempt to quantify free order variations across languages; we are not aware of evaluations of his grammars as parsing devices.

5.7 Conclusions

Our experiments in this chapter support two kinds of conclusions. First, modeling rules with algorithms other than n-grams not only produces smaller grammars but also better performing ones. Second, the procedure used for optimizing `alpha` reveals that some POS behave almost deterministically for selecting their arguments, while others do not. These conclusions suggest that splitting classes that behave non-deterministically into homogeneous ones could improve the quality of the inferred automata. We saw that lexicalization and head-annotation seem to attack this problem. Obvious follow-up questions arise: Are these two techniques the best way to split non-homogeneous classes into homogeneous ones? Is there an optimal splitting? Answers to these question will be given in Chapter 6.

Chapter 6
Splitting Training Material Optimally

6.1 Introduction

Our approach to parsing can be viewed as a simple CF parser with the special feature that our context free rules do not exist a priori. Instead, there is a device for generating them on demand. The device produces strings and these strings are used as CF bodies of rules. In the previous chapter, we used probabilistic automata for generating bodies of rules. These automata were not built manually, but we induced them from sample instances obtained from tree-banks. The general idea used for building the probabilistic automata model bodies of rules, consists of copying all bodies of rules inside the Penn Tree-bank (PTB) to a bodies of rules sample bag. This sample bag is treated as the sample set of a regular language and probabilistic automata are induced from it. Once the probabilistic automata have been built, they are used for defining a grammar that uses them for building rules on the fly.

The sample bag of rule bodies to be used as training material contains many different types of strings. For example, some strings may describe arguments of transitive verbs, while others may describe arguments of intransitive verbs. Chapter 5, together with previous work from the literature (Galen et al., 2004), suggests that models which are induced from the sample bag can be substantially improved as follows. First, split the training material into sets containing only homogeneous material; second, induce a model for each class, and third, combine the different models into one general model.

The assumption underlying this three step procedure is that the regular language we want to model is in fact the union of several languages. We split the material guided by the aim of keeping apart strings that belong to different languages. In this way, the models induced from each of the bags created after splitting the material are cleaner, because the algorithm that induces a regular language from one of the resulting training bags simply does not see any string belonging to other bags. By splitting the

data splitting we try to minimize the noise to which the learning algorithm is exposed.

To help shape our intuitions, let us consider two examples. First, suppose that the language we try to learn is the set $\{a, aa, aaa, aaaa, aaaaa\}$. Suppose that we have a bag of instances of this language and that we want to infer the original language from it. The learning algorithm might consider the bag of samples as instances of the language a^*, while if we split the sample bag into five different bags, each containing strings of the same length, it is more clear for the learning algorithm that each of the bags is produced by a language containing only one element.

Second, suppose that we want to model verb arguments. Simplifying, verb arguments can be thought of as the union of transitive verb arguments and intransitive verb arguments. Our working hypothesis is that a better model can be induced for verb arguments if we first split the training material into two different samples, one containing all the instances of transitive verbs and the other containing all the instances for intransitive verbs. Once the training material has been split, two models are induced, one modeling transitive verbs and the other modeling intransitive ones. The original material is divided into two bags to avoid the data instances from these two different phenomena from interfering with each other. Conceptually, if the data is not split, the algorithm for inducing a regular language for intransitive verbs sees the sample instances of transitive verbs as noise and vice-versa.

How do we split the training material? One possible way consists in defining a split by hand. Chapter 5 provides an example of this approach. There, the training material was split according to the head word of the dependency rules. As a consequence, two different automata for each part-of-speech (POS) were induced, one modeling right dependents and one modeling left dependents. In contrast, the approach we pursue in this chapter aims at finding an optimal splitting in an unsupervised manner. For this purpose we define a quality measure that quantifies the quality of a partition, and we search among a subset of all possible partitions for the one maximizing the proposed quality measure. Thus, one of our main challenges will be to find such a measure. Once the partition that optimizes our quality measure has been found, we use it for building as many automata as there are components in the partition. Finally, we use the induced automata for building PCW grammars, which we then use for parsing the PTB.

In this chapter we present a measure that quantifies the quality of a partition, we also show that the measure we found correlates with parsing performance. As a consequence, the procedure we use for splitting the material is a procedure that can be used for finding optimal grammars, optimal in the sense of parsing performance, without having to parse the PTB.

This chapter is organized as follows. Section 6.2 presents an overview of the chapter; Section 6.3 explains how to build grammars once the optimal partition has been

found; Section 6.4 explains how we search for the optimal partition, and Section 6.5 reports on the results on parsing the PTB. Section 6.6 discusses related work, and Section 6.7 states conclusions and describes future work.

6.2 Overview

We want to build grammars using training material that has been split into homogeneous classes of strings. Our main research goal is to understand how the elements in the training material interfere with each other, thus diminishing the quality of the resulting grammars. We also want to quantify the gain in terms of parsing performance that can be obtained by splitting the training material. Furthermore, we are interested in finding a quality measure for grammars that only takes the grammar's structure into consideration and helps us to predict the grammar's performance in parsing without actually parsing.

We proceed as follows. As in Chapter 5, we first transform the PTB into projective dependency structures following (Collins, 1996); see Section 2.1.1 for details. From the resulting tree-bank we delete all lexical information except POS tags. Every occurrence of a POS in a tree belonging to the tree-bank has associated to it two different, possibly empty, sequences of right and left dependents, respectively. We extract these sequences for all trees, producing two different bags containing right and left sequences of dependents, respectively.

We then proceed with a first splitting of the training material. For this purpose we use the POS tag of the head word as described in Chapter 5. This first splitting produces two different sample bags for each POS, one containing instances of left dependents, and the other containing instances of right dependents.

To keep our experiments focused we decided to split the training material of a single POS tag only: VB. VB is one of the POS with the highest value of perplexity (PP); experiments in Chapter 5 suggest that higher values of PP are due to the use of a single automaton for modeling different regular languages. Recall that, for instance, the values of PP drop considerably when the training material is split using the POS tag of the head word. Since the PP value associated to VB is one of the highest, VB seems to include words with substantially different behaviors, an intuition that is clearly confirmed by the literature (Levin, 1993; Merlo and Stevenson, 2001). We isolate the sentences containing the VB tag and see how dealing only with VB affects other tags.

The *initial partition* of the training set corresponding to the VB tag is split using syntactic information such as father tag, number of dependents, depth in the tree, etc. So, all instances in the training material that share the same feature are placed in the same bag. The initial partition aims at using external knowledge to split the material;

we try to characterize each of the resulting bags according to the output of the syntactic information used for building the split.

Recall that each component in the partition is a set of strings. We use such sets to build as many automata as there are components in the partition. For each of the automata built, we compute its quality, and the quality of the partition is defined as a combination of the qualities of those individual automata. Once the initial partition has been defined, genetic algorithms are used for finding a merging of components in the partition that optimizes the quality measure. Next, we use the optimal merging found by the genetic algorithm for building a PCW-grammar (see Section 6.3). Finally, we use the resulting grammar for parsing the PTB, and we report on the results in Section 6.5.

6.3 Building Grammars

In order to build a grammar we need to complete five steps: (1) obtain the training material from the PTB, (2) build an initial partition, (3) find an optimal partition containing the initial partition, (4) induce an automaton for each component in the optimal partition, and (5) put all automata together in a grammar. In this section we focus on steps (1), (4), and (5), while Section 6.4 focusses on steps (2) and (3).

6.3.1 Extracting Training Material

We extracted the training and testing material from the PTB. As we did in Chapter 5, all sentences containing CC tags are filtered out. We also eliminate all lexical information, leaving POS tags only. Dependents are extracted from dependency trees. For each dependency tree, we extract sample bags of right and left sequences of dependents. As an example, the tree in Figure 6.1 is transformed into the dependency tree shown in Figure 6.2. Its bags of left and right dependents are shown in Table 6.1.

From trees in sections 2–22 of the PTB we build two bags T_L and T_R containing left and right dependents respectively. From trees in sections 0–1 we build two different bags Q_L and Q_R, also containing left and right dependents respectively. The bags T_L and T_R are used as training material for automata induction algorithms, while bags Q_L and Q_R are used for evaluating the resulting automata.

6.3.2 From Automata to Grammars

Let T be a bag of training material extracted from the transformed tree-bank. Recall from Section 2.2.2 that we use two different measures for evaluating the quality of automata. Let Q be a test bag extracted as T. We use perplexity (PP) and missed

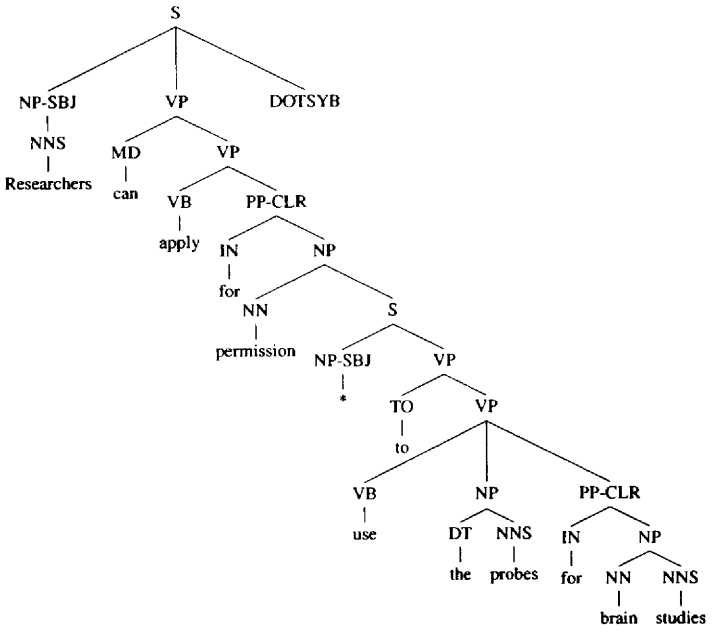


Figure 6.1: Tree extracted from the PTB, Section 02, file `wsj_0297.mrg`.

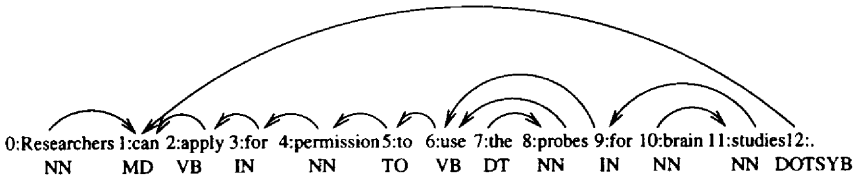


Figure 6.2: Dependency structure corresponding to the tree in Figure 6.1.

Word Position	Word's POS	Left	Right
0	NN	NN	NN
1	MD	MD NN	MD VB DOTSYB
2	VB	VB	VB IN
3	IN	IN	IN NN
4	NN	NN	NN TO
5	TO	TO	TO VB
6	VB	VB	VB NN IN
7	DT	DT	DT
8	NN	NN DT	NN
9	IN	IN	IN NN
10	NN	NN	NN
11	NN	NN NN	NN
12	DOTSYB	DOTSYB	DOTSYB

Table 6.1: Bags of left and right dependents. Left dependents are to be read from right to left.

samples (MS) to evaluate the quality of a probabilistic automaton. A PP close to 1 indicates that the automaton is almost certain about the next step while reading the string. MS counts the number of strings in the test sample Q that the automaton failed to accept.

Now, we describe how we build grammars once partitions over the bags of training material have been defined. Suppose that a partition $\Pi_{T_{\text{VB}}} = \langle \pi_1, \ldots, \pi_n \rangle$ has been found over the training material T_{VB}. Suppose also that, for each component π_i in the partition $\Pi_{T_{\text{VB}}}$, two automata $A_L^{\pi_i}$ and $A_R^{\pi_i}$, modeling left and right dependents respectively, have been induced. Finally, suppose that there are two automata A_L^w and A_R^w for all POS w in the PTB other than VB. Let G_L^w , G_R^w, $G_L^{\pi_i}$ and $G_R^{\pi_i}$ be their equivalent PCFGs obtained following (Abney et al., 1999). Let S_L^w, S_R^w, $S_L^{\pi_i}$ and $S_R^{\pi_i}$ be the start symbols of G_L^w, G_R^w, $G_L^{\pi_i}$ and $G_R^{\pi_i}$ respectively.

Our final grammar G is defined as follows. Its start symbol is S, its set of pseudo-rules is defined as the union of

$$\{W \xrightarrow{s}_1 S_L^w w S_R^w, \quad S \xrightarrow{s}_1 S_L^w w S_R^w : w \in POS\}$$

and

$$\{VB^{\pi_i} \xrightarrow{s}_1 S_L^{\pi_i} \text{VB} S_R^{\pi_i}, \quad S \xrightarrow{s}_1 S_L^{\pi_i} \text{VB} S_R^{\pi_i} : \pi_i \in \Pi_{T_{\text{VB}}}\},$$

and its set of meta-rules is the union of rules in G_L^w, G_R^w, $G_L^{\pi_i}$ and $G_L^{\pi_i}$ for all w in POS and π_i in $\Pi_{T_{\text{VB}}}$.

6.4 Splitting the Training Material

Let T^R_{VB} and T^L_{VB} be the training material corresponding to the right and left dependents of words tagged with VB. Let Q^L_{VB} and Q^R_{VB} be the left and right dependents from the tuning set whose head symbol is VB.

Let $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ be a partition of the set $T = T^R_{\text{VB}} \dot{\cup} T^L_{\text{VB}} \dot{\cup} Q^L_{\text{VB}} \dot{\cup} Q^R_{\text{VB}}$ (we denote the disjoint union of bags X and Y as $X \dot{\cup} Y$). Since Π is a partition of T, it induces a partition of each of the bags T^R_{VB}, T^L_{VB}, Q^L_{VB} and Q^R_{VB} when each of the sets is intersected with π_i. For example, the partition induced by Π over T_R is defined as $\Pi_{T^R_{\text{VB}}} = \langle \pi_1 \cap T^R_{\text{VB}}, \ldots, \pi_n \cap T^R_{\text{VB}} \rangle$.

Once a partition of T_R and T_L is defined, constructing a grammar is straightforward. Now, we focus on how to construct partitions Π. Partitions Π are defined in a twofold procedure. The first step defines an initial partition $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ using syntactic features. Syntactic features help to group VB arguments according to the position in which they appear in the sentences extracted from the tree-bank. In the second step, a quality measure for partitions is defined and an optimization of the global quality of the partition is performed. The optimization phase searches for the optimal partition among all partitions containing the initial partition. Consequently, the initial partition determines the space search for the optimization phase in the second step.

6.4.1 Initial Partitions

In order to define initial partitions we use features. A *feature* is a function f that takes two arguments; a dependency tree t and a number i. The number i is used as a reference to the i-th position in the sentence x to which f is applied. Since words in x are in direct correspondence with the nodes of the tree t yielding x, the index i also corresponds to a node in the tree t. A feature returns any piece of information regarding the position of the index i in the tree. Table 6.2 contains the features we use together with a brief description for each of them. For each feature f in Table 6.2, the table's third column shows the result of applying f to the tree in Figure 6.2 at position 2.

From a linguistic point of view, our features are used to characterize the dependents a verb might take. The underlying assumption is that features are capable of capturing the different behaviors that words tagged with VB might display. The idea is to group training instances according to their behavior. We divided the training material depending on the value a particular feature takes for a particular word in the particular tree where the word appears. We put all words' dependents tagged with VB with similar feature values into the same sample set. Consequently, the initial sample set is divided into smaller sample sets, each containing all dependents of words tagged VB

Name	Description	Example
WordStem	stem of the word at i	$WordStem(2) =$ `apply`
gFather	the grand-father of i	$gFather(2) =$ `NN`
Father	the father of i	$Father(2) =$ `IN`
Depth	the depth of the tree below i	$Depth(2) = 1$
rSibling	first left sibling of i	$rSibling(2) =$ `NONE`
FstLeftDep	the first left dependent of i	$FstRightDep(2) =$ `IN`
NumLeftDep	the numbers of left dependents of i	$NumRightDep(2) = 1$

Table 6.2: All features we use; they all take two arguments: a dependency tree t, and a node index i.

that share the same feature value. For example, suppose we use the feature *father*() to partition the training material. All components in the partition share the same value of *father* and there are as many components as there are possible outcomes for the feature *father*(). The underlying assumption becomes then, that all instances in a component are sampled from a regular language different from the regular language from which others components are sampled.

Formally, the initial partition is defined as follows. Let T be the bag containing all training material; let x be an element in T; let t_x be the tree in the tree-bank from which x was extracted. Let i_x be the position in t_x from which x was extracted. Finally, let $f_1, \ldots, f_k$ be the sequence of features we want to use for defining an initial partition. The initial partition $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ is given as the equivalence classes defined by the following equivalence relation R:

$$xRy \iff f_j(t_x, i_x) = f_j(t_y, i_y), j = 1, \ldots, k.$$

Once a feature has been defined, we have to assign new tags to all the material we used for building and testing the automata. For example, suppose that we use the *father* feature to produce the initial splitting. In this case, the tree in Figure 6.2 is transformed into the tree in Figure 6.3. The training material related to the retagged VB tags is shown in Table 6.3.

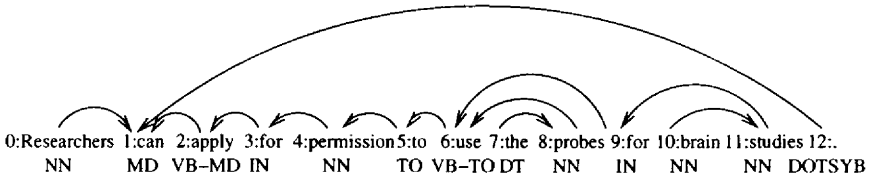


Figure 6.3: Dependency tree retagged according to the newly defined splitting.

Word Position	Word's POS	Left	Right
1	MD	MD NN	MD VB-MD DOTSYB
2	VB-MD	VB-MD	VB-MD NN IN
5	TO	TO	TO VB-TO
6	VB-TO	VB-TO	VB-TO NN IN

Table 6.3: Bags of left and right dependents.

We use features to induce a partition of the training material and of the testing material. Since the testing material is much smaller than the training material this might yield empty components. Since the values for our quality measures obtained from empty components are meaningless, we merge those empty components with those where the resulting automaton has the lowest perplexity. The resulting partition has no empty component. Such a partition is the starting partition for the algorithm searching for the optimal merging. We present 8 different grammars built using different features, the features used are described in Table 6.4. This table also shows the number of components each feature produces together with the number of components in after having searched for the best partition.

Grammar name	# components in initial partition	# components in optimal partition
Baseline	1	1
rSibling	6	1
NumRightDep	4	1
Father	13	6
gFather	30	9
Depth	17	10
FstRightDep	27	17
WordStem	373	57

Table 6.4: Features used and the number of components in the partitions they induce.

6.4.2 Merging Partitions

In this section we discuss the algorithm that searches for partitions over the training material containing the initial partition. We say that a new partition Π' *contains* a partition Π if for any two elements p and q that belong to the same component π_k, there is a component π' in Π' such that both elements are again in π'. Our intention is to search among the partitions that contain the initial partition for an optimal one. In

order to decide which partition is the optimal one, we first need to define a measure for evaluating its quality.

Recall that a component in a partition is used to define an automaton. For each component in the partition, we can define a value of PP and MS; in what follows we use each of these individual values to define a quality measure for the whole partition.

For every candidate merging, and for computing the partition's quality, we have to assign new tags to all the material. We assign new tags according to the redefinition of the features function we used for building the initial partition. For example, suppose that the *father*() feature was used for building the initial partition, suppose also that a candidate merge states that the component where *father* is equal to MD should be merged to the component where *father* is equal to TO. All tags VB in the training material with fathers tagged MD and TO have to be retagged with the same tag. For example, the tree in Figure 6.2 becomes the tree in Figure 6.4, where tags VB are renamed as VB-1.

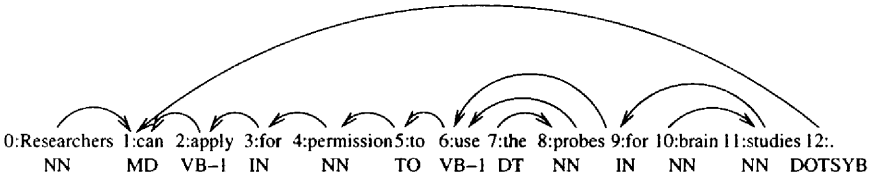


Figure 6.4: Assigning new tags for computing the merging of [TO] and [MD].

Our measure has two main parts, each of which considers the automata related to the left and to the right side. In order to simplify the exposition, we describe in detail our measure for the component referring to the right side. The component referring to the left side is obtained by replacing R in the superscripts with L.

Let $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ be a partition of the training material T. Let $A_i^R, i = 1, \ldots, k$ be the automata induced, as described in Section 6.3.2, using training sets $\pi_i \cap T_R$, respectively. Let PP_i^R and MS_i^R be the values of PP and MS respectively for the automaton A_i^R, computed using test sets $\pi_i \cap Q_R$, for $i = 1, \ldots, k$.

Our measure combines the values of PP_i and MS_i for all i. That is, we combine all values of PP and MS to obtain a quality value of the whole partition. PP and MS values can not be summed up directly given that the importance of an automaton is proportional to the number of times it is used in parsing. The importance of PP and MS values should be proportional to the number of times the corresponding automaton is to be used in generating bodies of rules. We have estimates of such frequencies using the training material. For that purpose, let

$$p_i^R = \frac{|\pi_i \cap T_R|}{|T_R|}, i = 1, \ldots, n.$$

One can view p_i^R as the probability of using the automata A_i^R. We use these probabilities to measure the expected value for MS and PP as follows. Let $E[MS_\Pi^R]$ be the expected value of MS for a right automata defined as

$$E[MS_\Pi^R] = \sum_{i=1}^{n} p_i^R MS_i^R.$$

Let $E[PP_\Pi^R]$ be the expected value of PP, defined as:

$$E[PP_\Pi^R] = \sum_{i=1}^{n} p_i^R PP_i^R.$$

Let $E[MS_\Pi^L]$ and $E[PP_\Pi^L]$ be the corresponding expected values for the left sides. Note that the expected values depend on a particular partition, hence the subscript Π. We are now in a position to compare the quality of two partitions according to the values they assigned to $E[PP_\Pi^R]$, $E[MS_\Pi^R]$, $E[PP_\Pi^L]$ and $E[MS_\Pi^L]$. We say that partition Π_1 is *better* than partition Π_2 if all of the following holds:

$$E[PP_{\Pi_1}^R] < E[PP_{\Pi_2}^R], \tag{6.1}$$
$$E[MS_{\Pi_1}^R] < E[MS_{\Pi_2}^R], \tag{6.2}$$
$$E[PP_{\Pi_1}^L] < E[PP_{\Pi_2}^L], \tag{6.3}$$
$$E[MS_{\Pi_1}^L] < E[MS_{\Pi_2}^L]. \tag{6.4}$$

Ideally, we would like to find a quality function q defined over the class of possible partitions such that $q(\Pi_1) < q(\Pi_2)$ if and only if Equations 6.1 through 6.4 are satisfied. If such a function exists, we can use many optimization methods for finding the partition for which q is minimal. But, it is easy to see that such a function does not exist. In what follows, we show that even a function q satisfying

$$q(\Pi_1) < q(\Pi_2) \iff \left(E[PP_{\Pi_1}^R] < E[PP_{\Pi_2}^R]\right) \wedge \left(E[MS_{\Pi_1}^R] < E[MS_{\Pi_2}^R]\right) \tag{6.5}$$

does not exist. Suppose that such a function does exist; suppose that the partitions Π_1 and Π_2 are two possible partitions with values $E[PP_{Pi_1}]$, $E[MS_{Pi_1}]$, $E[PP_{Pi_2}]$, $E[MS_{Pi_2}]$ for PP and MS. In order to compare the pair $(E[PP_{Pi_1}], E[MS_{Pi_1}])$ with the pair $(E[PP_{Pi_2}], E[MS_{Pi_2}])$ we can plot each pair as a vector, as shown in Figure 6.5. Since q is defined for all partitions, it takes values $q_1 = q(\Pi_1)$ and $q_2 = q(\Pi_1)$. Both q_1 and q_2 are real numbers, so $q_1 \leq q_2$ or $q_2 \leq q_1$, both possibilities imply that $q(\Pi_1) \leq q(\Pi_2)$ or $q(\Pi_2) \leq q(\Pi_1)$, which contradicts Equation 6.5 if $q_1 \neq q_2$. The constraints imposed by Equation 6.5 are impossible to fulfill because they required function q to map a partial order, defined over pairs of reals in the right-hand side to a total order defined over reals in the left-hand side. We can not apply a function minimization

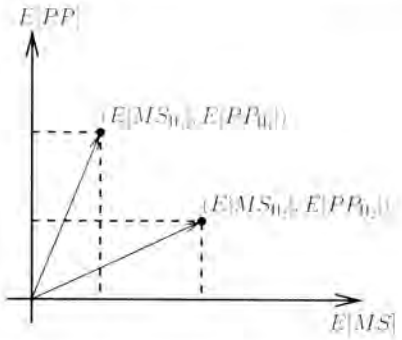


Figure 6.5: Values of $E[PP]$ and $E[MS]$ for two different partitions.

algorithm to minimize both $E[PP]$ and $E[MS]$ at the same time because they can not be combined in one function that minimizes both whenever the one function is minimized. Still, we want to minimize both values at the same time. We propose to circumvent this problem by fixing a reference point and biasing a function optimization algorithm to improve over the reference point. We try to optimize a given starting configuration of $E[PP^R_\Pi]$, $E[MS^R_\Pi]$, $E[PP^L_\Pi]$ and $E[MS^L_\Pi]$. We explain our algorithm for $E[PP^R_\Pi]$ and $E[MS^R_\Pi]$, extending it to the four components is straightforward. Suppose that our given reference point Π_0 is the one described by Figure 6.6. All vectors that satisfy Equation 6.5 are vectors inside the shaded area. In order to improve over the values of PP and MS defined by vector Π_0 we have to search for those vectors that lay in the shaded area. Observe that every vector in the shaded area codifies the

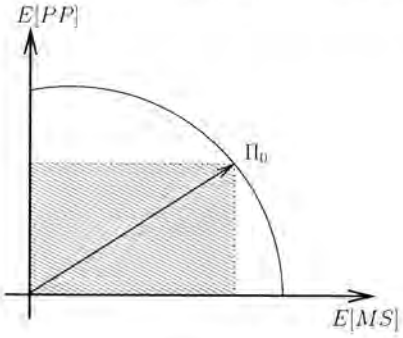


Figure 6.6: Values of $E[PP]$ and $E[MS]$ for two different partitions.

quality of a partition. Since not all the vectors in the shaded area define the same quality, we need to define a measure to pick the optimal one from the shaded area. We use the norm of the vector as a quality measure because it tries to minimize all the

components at the same time. Formally, the function q_{Π_0} that we minimize is defined as follows:

$$q_{\Pi_0}(\Pi) = \begin{cases} \|X\| + C & \text{if } E[PP^R_{\Pi_1}] > E[PP^R_{\Pi_0}] \\ \|X\| + C & \text{if } E[MS^R_{\Pi_1}] > E[MS^R_{\Pi_0}] \\ \|X\| + C & \text{if } E[PP^L_{\Pi_1}] > E[PP^L_{\Pi_0}] \\ \|X\| + C & \text{if } E[MS^L_{\Pi_1}] > E[MS^L_{\Pi_0}] \\ \|X\| & \text{otherwise,} \end{cases}$$

where $X = (E[PP^R_{\Pi_1}], E[MS^R_{\Pi_1}], E[PP^L_{\Pi_1}], E[MS^L_{\Pi_1}])$, C is a constant number and $\|(x_1, x_2, \ldots, x_n)\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$. We use the constant C to penalize the vectors outside the shaded area of Figure 6.6. We drop the reference partition subscript from q whenever the reference vector is clear from the context.

The measure defined this way, is a measure that depends on a starting configuration. In Figure 6.7 we show an example in which $q_{\Pi_0}(\Pi_2) < q_{\Pi_0}(\Pi_1)$, while Π_1 and Π_2 are incomparable using measures q_{Π_1} and q_{Π_2}. It is also interesting to note that whenever partitions Π_0, Π_1 and Π_2 are such that $q_{\Pi_0}(\Pi_1) < q_{\Pi_0}(\Pi_0)$ and $q_{\Pi_1}(\Pi_2) < q_{\Pi_1}(\Pi_1)$ then $q_{\Pi_0}(\Pi_2) < q_{\Pi_0}(\Pi_1)$.

In our experiments we use the trivial partition, i.e., the partition containing one and only one component containing all the training material, as reference point. Note that the reference point coincides with the partition we use in Chapter 5, consequently the results presented in this chapter are comparable to the experiments we performed in that chapter.

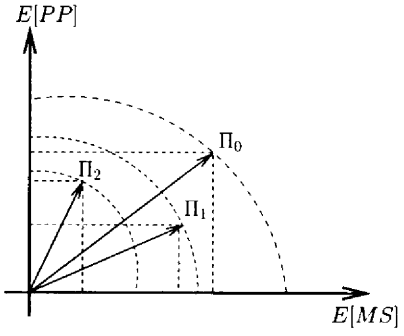


Figure 6.7: Two incomparable solutions.

We now apply this optimization technique to our specific problem, i.e., to find a merging of components that minimizes values of $E[MS]$ and $E[PP]$. We apply the procedure to different initial partitions: Table 6.5 shows the values of $E[PP]$, $E[MS]$, $E[PP]$, $E[MS]$, and q for all the grammars we build. Since all the experiments we carried out share the same reference partition, their values of q are comparable.

	left		right			
grammar	$E[PP]$	$E[MS]$	$E[PP]$	$E[MS]$	q	alpha
Baseline	1.189	0	9.633	6	10.911	0.00020
rSibling	1.189	0	9.633	6	10.911	0.00020
NumRightDep	1.189	0	9.633	6	10.911	0.00020
Father	1.188	0	9.743	5	9.432	0.00036
gFather	1.189	0	9.652	6	9.413	0.00034
Depth	1.180	1	9.783	5	9.065	0.00039
FstRightDep	1.188	0	2.950	1.431	3.337	0.00035
WordStem	1.195	0	9.647	5	4.705	0.00039

Table 6.5: Results on q for all the grammars we built.

From Table 6.5 we can see that the rSibling and NumRightDep do not suggest any partition that outperforms the value of q of the baseline; for this particular case genetic algorithms exhaustively searched the whole space of possibilities. This was possible because the space itself is not very big. The number of components in Table 6.4 gives a hint about the size of the space of possible merges.

To understand the meaning of the measure q in the context of two-level parsing, it is important to recall from Chapter 5 the meaning of PP and MS in this context. Recall that we are building PCW-grammars, and parsing with such grammars can be viewed as a two-phase procedure. The first phase consists in creating the rules that will be used in the second phase. The second phase consists in using the rules created in the first phase as a PCFG and in parsing the sentence using a PCF parser. Since automata are used to build rules, the values of PP and MS quantify the quality of the set of rules built for the second phase: MS gives us a measure of the number of rule bodies that should be created but that will not be created, and, hence, it gives us an indicator of the number of "correct" trees that will not be produced. PP tells us how uncertain the first phase is about producing rules. Now, q tries to minimize these two aspects: a partition that outperforms the baseline means that the automata we induced missed, on average, a smaller number of bodies of rules and that the bodies of rules that are created, on average, are created with lower perplexity.

Searching for the Optimal Partition

Let Π be an initial partition of T built as described in Section 6.4.1. Let Π_0 be the reference partition, i.e., the partition containing one component in which the whole of the training material is found. For each of the initial partitions we defined in Section 6.4.1, we search for the merging that optimizes the quality function q.

Formally, the search space is defined as the set of possible partitions containing the

initial one. Let Π and Π' be two partitions over T, and let a and b be two elements in T. Recall that a partition Π contains another partition Π' if all components in Π result from merging components in Π'. Consequently, a partition containing Π' can be easily generated by merging some of its components.

In order to search for the partition giving the minimum value of q we use Genetic Algorithms (GAs). We use GAs because our problem can naturally be re-phrased as a GA optimization problem.

In GAs, a population of individuals competes for survival. Each individual is designated by a bag of genes that define its behavior. Individuals that perform better (as defined by the fitness function) have a higher chance of mating with other individuals. A GA implementation runs for a discrete number of steps, called *generations*. What happens during each generation can vary greatly depending on the strategy being used. Typically, a variation of the following happens at each generation:

1. **Selection.** The performance of all the individuals is evaluated based on the fitness function, and each is given a specific fitness value. The higher the value, the bigger the chance of an individual passing its genes on to future generations through mating (crossover).

2. **Crossover.** Selected individuals are randomly paired up for crossover (also known as sexual reproduction). This is further controlled by the crossover rate specified and may result in a new offspring individual that contains genes from each of its parents. New individuals are injected into the current population.

3. **Mutation.** Each individual is given the chance to mutate based on the mutation probability specified. Each gene of an individual is looked at separately to decide whether it will be mutated or not. Mutation is decided based upon the mutation rate (or probability). If a mutation is to happen, the value of the gene is switched to some other possible value.

In order to use GAs for our purposes we have to provide the following:

1. **A definition of individuals:** We design our individuals to codify two things. First, a value of `alpha` to be used for building the automata and second a partition of the training material. `alpha` is simply codified as the first gene in the vector; the partition containing the initial partition is codified as follows. Note that in order to describe a partition for the training material it is enough to describe a way to merge components in the initial partition. Individuals in the population specify a way to merge components belonging to the initial partition into new components. A number k in the i-th position in the vector V indicates that component i in the original partition should be added to the new component

k. Formally, let $V' = \langle a_1, \ldots, a_n \rangle$ be a vector with $i \leq a_i \leq n$. The vector V' defines the partition $\Pi' = \langle \pi'_1, \ldots, \pi'_{n'} \rangle$ such that $\pi'_i = \bigcup \{\pi_k : V'[k] = i\}$. Intuitively, the number of components in the resulting partition is equal to the number of different values stored in vector V'. E.g., if all entries in V' are the same, the new partition defined by V' contains only one component.

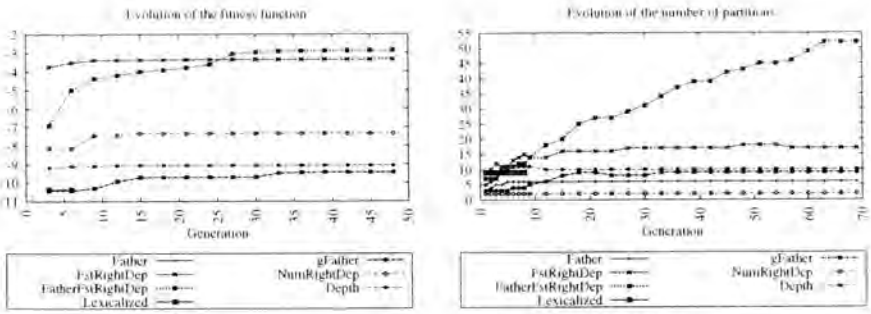
2. **A fitness function defined on individuals:** The fitness function for an individual is defined as the quality measure q we defined in Section 6.4.2.

3. **A strategy for evolution:** The strategy we follow is defined as follows. We apply two different operations to genes, namely crossover and mutation. We decide which operation to apply by flipping a biased coin. Crossover get 0.95 probability of being applied while mutation gets 0.05. Once the operation is chosen, genes to which the operations apply are selected from the population. We select individuals using the *roulette wheel strategy* (Gen and Cheng, 1997), in which the probability for an individual to be selected is proportional to its fitness score. Crossover is implemented as follows, two points are selected along the chromosomes of both parents. The chromosomes are then cut at those points, and the middle parts are swapped, creating two child chromosomes. If mating occurs, two new genes are added to the population. If no mating occurs, no new gene is add to the population. Mutation is implemented as follows. Each gene of an individual is looked at separately to decide whether it will be mutated or not. Mutation is decided based upon the mutation rate (or probability). If a mutation is to happen, then the value of the gene is switched to some other possible value. For further details on the implementation of GAs we used see (Qumsieh, 2003)

Finally, the population of each of our generation consists of 50 individuals; we let the population evolve for 100 generations. We decided to use 100 generations because the computation of the quality of partition q is time consuming and, moreover the quality measure and the number of partition are stable around generation 65 as pictured in Figure 6.8.

6.5 Parsing the Penn Treebank

Finally, we report on the accuracy in parsing. As in Chapter 5 we use two measures, %Words and %Pos. The former computes the fraction of the words that have been attached to their correct father, the latter computes the fraction of words that were attached to the correct word-class. As explained in Chapter 5 the two measures try to capture the performance of the PCW-parser in the two phases procedure described in



(a) (b)

Figure 6.8: (a) The value of the quality measure q across generations for 7 different grammars. (b) The number of components across generations. In both plots, plotted values at population i correspond to the value of q and the number of components of the individual with the highest q among all individuals in i.

Chapter 5: (%POS) tries to capture the performance in the first phase, and (%Words) in the second phase.

The measures reported in Tables 6.6 and 6.7 are the mean values of (%POS) and (%Words) computed over all sentences in section 23 having length at most 20. We parsed only those sentences because our baseline, coming from Chapter 5, was computed on these sentences. The grammar names in Tables 6.6 and 6.7 refer to the features used for splitting the training material and building the grammars; see Table 6.2 for an explanation of each feature.

Since in our approach we split only the training material for VB we start by commenting the scores on parsing sentences containing the VB tag. Table 6.6 columns 4 and 5 shows the results. We have identified two sets of grammars. For those in the up-

			with VB tag		without VB tag	
Grammar	q	# components	%Words	%POS	%Words	%POS
Baseline	10.911	1	0.8484	0.8738	0.8493	0.8806
rSibling	10.911	1	0.8484	0.8738	0.8493	0.8806
NumRightDep	10.911	1	0.8484	0.8738	0.8493	0.8806
Father	9.432	6	0.8601	0.8846	0.8494	0.8818
gFather	9.413	9	0.8502	0.8758	0.8477	0.8798
Depth	9.065	10	0.8608	0.8855	0.8520	0.8829
FstRightDep	3.337	17	0.8623	0.8876	0.8521	0.8834
WordStem	4.705	57	0.8498	0.8758	0.8493	0.8818

Table 6.6: Results on parsing sentences containing and not containing the VB tag.

per part of the table there is a correlation between the quality measure and the parsing score given that for lower values of q we obtain greater values of parsing scores. This does not seem to be the case for the grammar in the lower part of the table. For the latter, even though the quality measure q is quite a bit smaller than the baseline (4.705 vs. 10.911), its parsing score is only marginally different from the baseline score. As q gives us the expected values of PP and MS, intuitively this states the grammar in the lower part should have less ambiguity when choosing dependents of words tagged VB, but it produces virtually identical parsing scores even though its q values are lower than for nearly all grammars in the upper part. We think that is due to the number of components in the partitions. Note that the grammars in the upper part are induced from partitions containing less than 20 components, while the grammar in the lower part is induced from a partition containing no less than 57 components. As a consequence of having a large number of components each component contains a small amount of training and testing material and the values of PP and MS become unreliable. A better version of q should take into consideration the number of components in each partition and it should punish those components containing a small number of instances. Obviously, another way to overcome this problem is to use more training and testing material. Alternatively, a possible solution is to adopt a similar approach to the use in many clustering techniques where the number of components is required to be fixed and to search for the partition that optimizes q among all partitions having a fixed number of components.

With the experiments we carried out, it is possible to draw some conclusions for the grammars in the upper part of Table 6.6. Note that for those grammars, the ranking correlates (more correctly: inversely correlates) with the parsing score. This suggests that q is an indicator of the parsing score and that q can be used to quantify the quality of grammars without having to parse the whole gold standard. For the grammars in the upper part of Table 6.6, columns 4 and 5, we computed Pearson's product-moment correlation (NIST, 2004; Wright, 1997). We computed the correlation between q and %Words and between q and %POS for the case of sentences containing the VB tag. Pearson's correlation coefficient is usually signified by ρ, and can take on the values from -1.0 to 1.0. Here, -1.0 is a perfect negative (inverse) correlation, 0.0 is no correlation, and 1.0 is a perfect positive correlation. The statistical significance of ρ is tested using a t-test. The t-test returns a p-value, where a low p-value (less than 0.05 for example) means that there is a statistically significant relationship between the two variables.

Now, Pearson's product-moment correlation test shows a correlation value of $\rho = -0.821$, $p = 0.04484$ and $\rho = -0.835$, $p = 0.03832$, for q vs %Words and q vs %POS, respectively. The correlation values suggest that q is a measure that only takes into consideration the way a grammar was built in order to predict its parsing performance.

Grammar	q	#components	%Words	%POS	%POS – %Words
Baseline	10.911	1	0.8491	0.8787	0.0296
rSibling	10.911	1	0.8491	0.8787	0.0296
NumRightDep	10.911	1	0.8491	0.8787	0.0296
Father	9.432	6	0.8525	0.8826	0.0301
gFather	9.413	9	0.8484	0.8787	0.0303
Depth	9.065	10	0.8545	0.8836	0.0291
FstRightDep	3.337	17	0.8550	0.8846	0.0296
WordStem	4.705	57	0.8494	0.8801	0.0307

Table 6.7: Results on parsing the PTB (all sentences).

Note that the values of p are small, they are below 0.05 which is usually the weakest evidence that is normally accepted in experimental sciences. However, the correlation was computed only on a few sample points; in order to get more reliable values of correlation it is necessary to use bigger collections as training material and to define and compute q for a larger number of grammars. Nevertheless, the correlation values found suggest that the observed differences are significant and that they are not the product of a random improvement.

Since all the training material is retagged according to the components induced for VB, automata induced for POS other than VB might alter their quality. In order to get a quantitative picture of the impact of the splitting in POS other than VB we separately parse sentences that do *not* contain the VB. Columns 6 and 7 in Table 6.6 show the results. Parsing scores are close to the baseline, whenever the sentences do not contain the VB tag. Phrased more positively, while optimizing for sentences containing words tagged VB, parser performance on sentences not containing words tagged VB did not decrease. Indeed, the Pearson's product-moment correlation tests for these columns show a correlation value of $\rho = -0.2860$, $p = 0.5816$ and $\rho = -0.5369$, $p = 0.271$, for q vs %Words and q vs %POS, respectively. These correlation values suggest that q does not (inversely) correlate with parsing performance for sentences not containing the VB tag.

We can speculate that splitting material for one particular POS tag does not hurt the parser performance on other POS tags. This suggests that we could proceed by splitting different POS separately and then combine them in one grammar. The parsing performance for the final grammar should gain from all the gains in performance for each of the non-trivial splitting.

Finally, for the sake of completeness Table 6.7 presents the parsing scores for all the sentences in the test set. Observe that, indeed, the scores over all sentences do improve, even if we only optimized for a single POS.

6.6 Related Work

There are several perspectives from which we can analyze the approach and the experiments in this chapter. A first analysis sees our procedure to find optimal grammars as a way to induce preterminal symbols from the PTB. They are optimized for parsing and they provide information about the behavior of words tagged with VB. These preterminal symbols define a classification of verbs based on their syntactic behavior. There is a large collection of work on classification of verbs; most of them try to induce a classification of verbs using syntactic features, in some cases the resulting classification is evaluated (Merlo and Stevenson, 2001; Stevenson and Merlo, 2000), while in others the resulting classification (Decadt and Daelemans, 2004) is compared to the hand-crafted classification made by Levin (1993). We ran some experiments on trying to classify verbs according to the components they belong to. We checked the match between our classification and Levin's manual classification of verbs. Unfortunately, we did not see any clear match between the two. We also explored manually the classification induced by our procedure, but we could not detect any linguistic explanation of the classification. We think that in order to get a linguistically meaningful classification, more training material and material tagged with other verb tags should be used.

A second analysis sees our procedure as a method for finding labels for estimating better probabilities. We can replace words by more general categories, like POS tags, in order to induce better parameters. The clusters we found can be viewed as new labels because these labels group words having comparable syntactic behavior. To use our labels in order to obtain better probabilities we need to retag not only the training material but also the gold standard. When the gold standard is retagged new tags codify some structural information. The whole approach is a simplified version of supertagging (Joshi and Srinivas, 1994; Srinivas, 1997) for PCW-grammars.

A third analysis considers the procedure as a way to induce sub-categorization frames (Manning, 1993; Carroll and Fang, 2004) for words tagged with VB. Our sub-categorization frames have the peculiarity that there is an infinite number of them given that each string accepted by our automata is a possible sub-categorization frame. Our induced sub-categorization frames are used for improving the parsing performance and are induced specially for this purpose. Only recently (Carroll and Fang, 2004; Yoshinaga, 2004; Hara et al., 2002) some work appeared where the induced sub-categorization frames are used for improving the parsing task.

Outside the context of parsing, the methodology we presented in this chapter can also be used for inducing regular languages. The idea of using clustering before inducing automata is not new. Dupont and Chase (1998) clustered symbols using standard clustering techniques (Brown et al., 1992; Ney and Kneser, 1993) before inferring the

automata. The main difference between our approach and theirs is that our algorithm presupposes that the target language is the union of different languages and the method tries to automatically detect the different components. Our algorithm also tries to detect the number of components automatically, while in (Dupont and Chase, 1998) the number of components is a parameter of the algorithm. From a more technical point of view, and still within the setting of inducing probabilistic regular languages, the procedure of, first, splitting the training material and, second, inducing as many regular languages as there are components, is a technique that guides the merging of states in the MDI algorithm and that disallows some of the possible merging. Recall from Section 2.2.2 that the MDI algorithm builds an automata, first, by building an initial automaton and, second, by merging some of the states in the initial automaton. When the material is split, not one but many initial automata are built. For this case, the MDI algorithm searches for candidate merges within each of the initial automaton. As a consequence, some candidate merges that were possible when inducing one automaton are not available any more in the case of many automata. Clearly, there are two questions that the splitting approach has to address, the first one is how to recombine the different automata in one single regular language, and the second, what criteria should we follow for splitting the training material? For our particular case, we use PCW-grammars for recombining the automata and syntactic information for splitting the training material.

From the point of view of optimization, we present a solution for optimizing two functions at the same time. The problem of optimizing more than one function at the same time is known as *multiobjective optimization* (Coello Coello, 1999). Briefly, multiobjective optimization techniques try to optimize a combination of many functions, called objectives, by finding a trade-off between the objectives. Under the multiobjective optimization perspective, it is possible to optimize the combination of objectives by optimizing one of them while others are not optimized.

6.7 Conclusions and Future Work

We presented an approach that aims at finding an optimal splitting of the training material, which in turn, is meant for improving parsing performance. For this purpose we defined a quality measure that quantifies the quality of partitions. Using this measure, we search among a subset of all possible partitions for the one maximizing the proposed quality measure. Our measure combines a quality measure defined for each component in a partition. To measure each component's quality, we compute an automaton for each of component and we computed the automaton's MS and PP. The measure we presented combines values of PP and MS for all resulting automata, one

per component, and it uses the resulting components to build grammars that are subsequently used for parsing the PTB.

For our particular case, it is not clear how to combine the functions $E[PP]$ and $E[MS]$ in such a way that the optimization of the combined values produces better parsing scores. What we know, is that if we optimize the four values ($E[PP]$ and $E[MS]$ for the left and right side) at the same time, we gain in parsing performance. While searching for the definition of the optimal function q, we noticed that there might be a measure that uses only a subset of these four values. It seems that the four functions are not fully independent but the underlying relation remains an open problem.

We have shown that the quality measure we defined can be used for comparing parsing scores of two grammars whenever the grammars are built from partitions having a similar number of components. It would be interesting to define a measure that correlates with parsing performance independent of the number of components in the partition. The natural next step is, then, to define a measure that takes into account the number of components and the number of elements in each component. It is also important to investigate the impact a bigger corpus has in the measure we defined.

In this chapter we used PCW-grammars as the backbone for our experiments. They provide us with the appropriate level of abstraction for carrying out the experiments, and an easy way to combine all automata we induced for the different components into one single grammar. In contrast to the grammars in Chapter 5, the grammars we built in this chapter are not bilexical grammars. But, since the parser we implemented (see Appendix A for details) is a parser for PCW-grammars, it can handle both types of grammars. Grammars in Chapter 5 and the grammars in this chapter have in common that they search for unlabeled links.

Finally, this chapter changes the way the parsing task is usually addressed. Parsing is usually treated more as a modeling task than as an optimization task. A modeling task is a task where a model is designed and its parameters estimated from training material. Once these parameters are estimated, the model is tested on the parsing task and its results reported. In contrast, an optimization task is a task where a model is designed and its parameters are optimized according to the performance of the model and parameters in the final task. The difficulty of treating parsing as an optimization task resides in the time it takes to test a set of candidate values for the parameters. Since our measure q is a good indicator of the parsing performance we can treat parsing as an optimization task without having to parse. As a consequence, the procedure we defined is a procedure for building optimal grammars.

Chapter 7
Sequences as Features

7.1 Introduction

In the parsing activities and methods discussed so far in this thesis, we set out to discover syntactic structure, and in particular word dependents, using only sequences of POS tags. In contrast, nearly all other parsing approaches discussed in the literature use both sequences of POS tags and sequences of grammatical relations (GRs). Grammatical relations are labels describing the relation between the main verb and its dependents and they can be viewed as a kind of non-terminal labels. This observation suggests an obvious research question: which of the two types of information helps more for the discovery of word dependents, sequences of POS tags or sequences of GRs? This is our main research question in this chapter. Let us make more precise what it means.

In order to obtain phrase structures like the ones retrieved in (Collins, 1999), the dependents of a POS tag should consist of pairs of POS tags and non-terminal labels instead of sequences of POS tags alone (Markovian rules capture such pairs; see Chapter 4 for details on how PCW-grammars capture them). Like sequences of POS tags, sequences of pairs of POS tags and non-terminal labels can be viewed as instances of a regular language: a regular language whose alphabet is the product of the set of possible POS tags and the set of possible non-terminal labels. Moreover, they can be viewed as instances of the combination of two different regular languages: one regular language modeling sequences of POS tags, and another regular language modeling sequences of non-terminal labels. Under this perspective, it is clear that Chapters 5 and 6 only use the first regular language, while non-lexicalized approaches use the second regular language, and Markovian rules use a combination of the two.

From the literature, it is clear that combining the regular language of POS tags and the regular language of non-terminal labels boosts parsing performance, but it is not clear why this is the case. Chapter 5 suggests that lexicalization improves the quality

of the automata modeling sequences of POS tags, but it does not provide any insight about the differences or the interplay between these two regular languages.

In this chapter we design and implement experiments for exploring the differences between the regular language of POS tags and the regular language of non-terminal labels in a parsing setup. Our research aims at quantifying the difference between the two and at understanding their contribution to parsing performance. In order to clearly assess the contribution of these two features, we need to carry out an evaluation in terms of a task that clearly isolates the two regular languages. We decided to use the task of detecting and labeling dependents of the main verb of a sentence. Labels describing the relation between the main verb and its dependents are what we call *grammatical relations* (GRs), and they can be viewed as a kind of non-terminal labels.

We present two different approaches for dealing with the task of finding grammatical relations. In the first approach, we develop two grammars: one for detecting dependents and another for labeling them. The first grammar uses sequences of POS tags as the main feature for detecting dependents, and the second grammar uses sequences of GRs as the main feature for labeling the dependents found by the first grammar. The task of detecting and labeling dependents as a whole is done by cascading these two grammars. In the second approach, we build a single grammar that uses sequences of GRs as the main feature for detecting dependents and for labeling them. The task of detecting and labeling dependents as a whole is done in one go by this grammar. The two approaches differ in that the first one uses sequences of GRs and sequences of POS tags, while the second only uses sequences of GRs.

We show that English GRs follow a very strict sequential order, but not as strict as POS tags of verbal dependents. We show that the latter is more effective for detecting and labeling dependents, and, hence, it provides a more reliable instrument for detecting them. We argue that this feature is responsible for boosting parsing performance.

The rest of the chapter is organized as follows. Section 7.2 details the task we use for testing the two features. Section 7.3 explains how to encode this task as parsing. Section 7.4 discusses how the training material used in our experiments is obtained. Section 7.5 explains how the grammars we use are built. Section 7.6 discusses the optimization phase for those grammars. Section 7.7 shows the experimental results; Section 7.8 discusses related work and, Section 7.9 concludes the chapter.

7.2 Detecting and Labeling Main Verb Dependents

The task we use for our experiments is to find main verbs dependents and to determine their GR. In this section we describe the selected task as a black-box procedure. We specify its input and its output. The input of the task consists of the following items:

1. the main verb of the sentence,
2. the head word for each of the chunks into which the sentence has been split, and
3. the POS tags for the heads of the chunks.

The definition of chunks becomes clearer in the next section. For now it is enough to know that the sentences is chunked and that not all the words are used. Figure 7.1 shows an example of the input data.

[Pierre Vinken],	[61 years]	[old],	[joined]	[the board]	[as]	[a nonexecutive director]	[Nov. 29].
NNP	NNS	JJ	VBD*	NN	PP	NN	CD

Figure 7.1: Example of the information to be parsed by the grammars we build.

The output consists of a yes/no tag for each element in the input string. A POS tag marked yes implies that the tag depends on the main verb. If a POS tag is marked yes, the outputs has to specify the GR between the POS tags and the main verb.

The desired output for the input in Figure 7.1 is shown in Figure 7.2. Tags labeled yes have been replaced by links between the POS tags and the main verb.

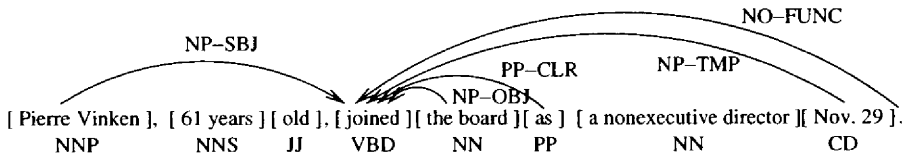


Figure 7.2: Information we use from each tree in the PTB.

Note that not all POS tags in our example sentence bare a relation to the main verb. More generally, there might be POS tags that depend on the main verb but whose relation cannot be labeled by any of the labels we define later in this chapter. These links receive the NO-FUNC label. It is important to distinguish between the POS tags that do not have a relation to the main verb and those that depend syntactically on the main verb but whose relation cannot be labeled. The former are marked with the no tag, while the latter are marked with the yes tag and the GR is NO-FUNC. See Figure 7.2 for an example.

7.3 PCW-Grammars for Detecting and Labeling Arguments

In order to determine the contribution of the two kinds of information (sequences of POS tags and sequences of GRs), we set up the task of detecting and labeling as a

combination of two independent tasks. The first one is to find the dependents of the main verb, and the second to label them.

In order to try to use sequences of POS tags and sequences of GRs as features, we codify GRs in pre-terminal symbols. Figure 7.3 shows an example. It shows the verb dependents from Figure 7.1: `nnp nn pp`, and `cd`, with labels as pictured, while `nns jj`, and `nn` do not hold any relation to the main verb and, consequently, they are not linked or labeled and not shown in Figure 7.3.

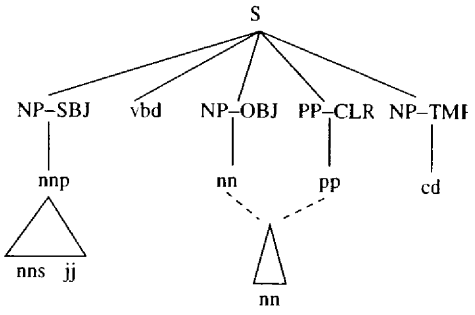


Figure 7.3: The tree we want to retrieve.

In Figure 7.3, we can clearly distinguish the two regular languages that can be used for detecting dependents of verbs: the sequences `NP-SBJ` and `NP-OBJ PP-CLR NP-TMP` are instances of the regular languages whose alphabet is the set of possible GRs, while the sequences `nnp` and `nn pp cd` are instances of the regular language whose alphabet is the set of possible POS tags.

We build 3 different grammars:

1. A grammar G_D that aims at *detecting* main verb dependents. This grammar uses automata that model sequences of POS tags. The parser that uses this grammar is fed as with all the POS tags.

2. A grammar G_L that aims at *labeling* dependents. This grammar uses automata that model sequences of GRs. The parser that uses this grammar is fed with the POS tags that are believed to depend on the main verb. The result is a GR name for each POS tag in the input sentence.

3. A grammar G that aims at *detecting* and *labeling* main dependents. This grammar uses automata that model sequences of GRs together with automata that models sequences of POS tags. The input and output of parsing with this grammar is as described in Section 7.2.

With these three grammars we can achieve the task described in Section 7.2 in two different ways:

1. We use G_D for detecting dependents, and G_L for labeling the dependents that G_D outputs.

2. We use G for detecting and labeling the main dependents.

Each of these three grammars are PCW-grammars (see Section 6.3), and they all are built using automata, just like the grammars we built in Chapters 5 and 6. In order to build them we have to carry out the following three steps:

1. generate the training material for training the automata,

2. optimize the automata, and

3. build the grammar.

Sections 7.5.1, 7.5.3 and 7.5.2 describe steps 1 and 3 for grammars for detecting dependents, labeling dependents and detecting and labeling dependents respectively. Section 7.6 explains which automata are optimized, and in which way.

For all grammars, Step 1 uses training material extracted from a labeled dependency grammar version of the PTB. The following section gives an overview of how we transform the PTB into labeled dependency trees.

7.4 Transforming the Penn Treebank to Labeled Dependency Structures

The set of GRs we aim to capture is a fixed set that is defined by the annotation schema followed in the Penn Tree-bank (PTB). We transformed the PTB into labeled dependency structures from which we induced our grammars.

All the training material we used comes from the PTB, hence the grammatical relations we are able to retrieve are those that are marked in the training material. We used `chunklink.pl` for transforming the PTB to labeled dependency structures and for marking all the information we need in the PTB (Buchholz, 2002). For detailed information on `chunklink.pl`, the reader is referred to the latter publication. In order to better understand the nature of the GRs to be found, we briefly describe how GRs are marked by `chunklink.pl` in the PTB.

The procedure for identifying links, chunks, and labels consists of four steps:

1. detecting words that may be heads of chunks,

2. drawing dependency relations (links) between these words,

3. assigning labels to these relations and

4. detecting chunks.

As to step (1) `chunklink.pl` detects heads using a head table, pretty much as explained in Section 2.1.1. Still, the application of tables described in Section 2.1.1 is different from the strategy used by `chunklink.pl`. In the latter, the head is either the right-most pre-terminal child that matches the (regular expression) POS tags list in the table, or all non-terminal children that match the (regular expression) constituent list. Consequently, there is a preference for lexical over non-lexical head children, but no preference within these groups. In the approach presented in Section 2.1.1, the list is ordered by preference and it also has an associated direction (starting left or right depending on the index of the table). The head detection algorithm first tries to find a child of the kind indicated by the first element of the list in the indicated direction, and stops as soon as it finds one. If no child of this kind can be found, the algorithm next looks for a child of the kind of the second element of the list, and so on, down to the last. If even the last kind of child cannot be found, the algorithm takes the left/rightmost child (of any kind) to be the head. The two different approaches, according to (Buchholz, 2002), differ mainly with respect to coordinated structures. For us, this difference is irrelevant, because we compare our approaches in each of the cases using the corresponding training material. Buchholz (2002) has an extensive description of the strategy followed by `chunklink.pl` for handling special cases that can not be described in the table. Clearly, the result of both heuristics may be "unknown." For these cases `chunklink.pl` returns an unknown head. In our experiments we discarded all sentences that contain at least one "unknown" head.

Once heads have been marked, the algorithm proceeds with step (2). Links (dependency relations) are drawn in a bottom-up fashion. In the process of drawing links, the tree ends up in an intermediate structure, which is then used in the third step. We describe here how the tree is transformed into a directed graph; the resulting graphs contain a partial dependency structure like the one used in previous chapters, plus links between non-terminals in the original tree and words in their yield. Step (3) uses the latter links for labeling relations in the dependency tree and completing the dependency tree; it finally eliminates them from the the graph producing a labeled dependency tree.

Step (2) starts by adding a link for each word in the yield of the tree that links the word to itself. Recall that, for each non-terminal, the algorithm knows which of its child nodes contains the head word. Step (2) traverses the tree in a bottom-up fashion: it adds a link between the current non-terminal and whatever the head child is pointing at. The result is a link between the non-terminal and a word in the yield. Step

(2) also redirects all links outgoing from the current non-terminal child to the same word the current non-terminal is pointing to. Note that step (2) outputs an incomplete dependency structure at word level, together with a graph where all non-terminals point to a word in the yield.

Step (3) uses the structure that step (2) outputs to add labeled links to words that remain to be linked in the dependency structure output by (3). A link between a non-terminal T and a word w in the yield encodes a dependency between w and the word that is found by descending in the tree from T to the yield, always following head children. These dependencies are the ones that step (3) draws. Recall that each non-terminal points to a word w in the yield. Step (3) adds a link for each non-terminal NT in the tree. The link goes from the word in the yield resulting from going down the non-terminal, always following the head children and the word w in the yield.

The pointer introduced by step (3) indicates dependency relations between syntactic constituents and head words. We are interested in relations between constituents and words, but to establish relations between pairs of words, step (3) traverses again the tree in a top-down fashion, and pushes the labeled pointer of the parent to the head child. If the syntactical part of the pointer label and of the non-terminal head children are identical, the pointer label stays the same. However, if the syntactic part is different, the head child label is prefixed to the pointer label, separated by the "/" symbol. When "pushing down" the function pointers, we lose the information about the level at which they were originally attached. In most cases this information is not relevant, as the function is defined by the combination of syntactic category and function tag. However, in the case of NPs without function tag, the level of attachment makes the difference between a complement (object) and an adjunct. In order to preserve this distinction, Buchholz (2002) adds the new function tag `-OBJ` to the following constituents if they occur without function tags and as siblings of lexical heads: `NP`, `VP`, `ADJP`, `S`, `SBAR`, `SINV`, `SQ`, `SBARQ`. These is done during phase (2).

Step (4) uses the links between words that were already present at step (2) for finding chunks. Buchholz (2002) defines a chunk to consist of a head, i.e., any word that has a labeled pointer, plus the continuous sequence of all words around it that have an unlabeled pointer to this head. Since labeled links between words were introduced by step (3), chunks are defined by all links between words that appear up to step (2) in the algorithm. This chunk correspond to the projection of the pre-terminal level in the original tree.

Since our experiments use the transformed version of the PTB and try to compare two different aspects of syntax, there are two issues we should discuss. The first is related to the nature of the transformation defined by `chunklink.pl`. Clearly, `chunklink.pl` takes a phrase structure as input and returns a dependency tree: it might be that this transformation discards some information from the PTB and that this

loss of information produces misleading results in our experiments. It seems to us that `chunklink.pl` does *not* define an invertible procedure, i.e., the dependency trees returned by it can *not* be transformed back to the original phase structure tree, because labels of some of the intermediate constituents are deleted during pruning (Buchholz, 2002, page 60). Buchholz (2002, page 59) also mentions loss of information regarding the original attachment position of grammatical functions. Despite all this, we think that `chunklink.pl` does not discard too much information and that the structures it produces are still meaningful.

The second issue to discuss is that, in theory, the transformation might be more beneficial for one of our experiments than for the other. It is not clear to us that this is indeed the case. All of our experiments are close to each other in that they use the same type of information and that the transformation does not favor a particular experiment.

7.5 Building the Grammars

For each of the tree grammars we build, we have to follow the same 3 steps:

1. extract the training material,
2. find the best automata, and
3. use the automata to build the grammar.

The optimization procedure we use for selecting the best automata, step (2), is the same for all grammars, while steps (1) and (3) are different for each particular grammar. Sections 7.5.1, 7.5.2, and 7.5.3 describe steps (1) and (3) for each of the grammars, while Section 7.6 describes step (2).

7.5.1 Grammars for Detecting Main Dependents

The grammar for detecting dependents is very similar to the grammars we built in Chapters 5 and 6. For each sentence parsed with this grammar, the parser outputs a dependency structure; the main verb dependents are found in this dependency structure.

Extracting Training Material

In order to obtain training material we transformed the PTB, sections 11–19, as explained in Section 7.4. For each dependency tree in the transformed TB, we extracted a sample set of right and left sequences of dependents. Figure 7.4 shows an example of a dependency tree, and Table 7.1 shows the sample sets of right and left dependents we extracted from it. We built two different sample bags per POS tag, one containing

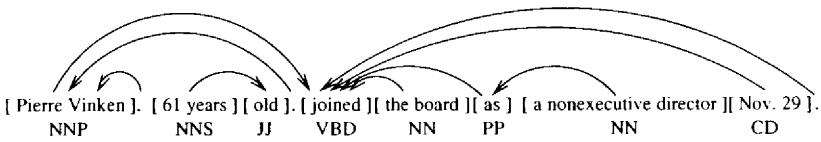


Figure 7.4: A dependency tree from which we extracted training material.

all instances of left dependents and one containing all instances of right dependents. For each of the bags we built an automaton. The description of how to build an automaton from a bag of samples and the steps we follow for optimizing all automata are discussed in Section 7.6.

POS	Left	Right
NNP	NNP COMMA COMMA	NNP
COMMA	NNP	NNP
NNS	NNS	NNS
JJ	JJ NNS	JJ
COMMA	COMMA	COMMA
VBD	VBD NNP	VBD NN PP CD DOT
NN	NN	NN
PP	PP	PP NN
NN	NN	NN
CD	CD	CD
DOT	DOT	DOT

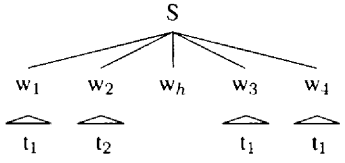
Table 7.1: Instances of left and right dependents extracted from the tree in Figure 7.4.

Building the Grammar

Once the training material has been extracted, we build two different automata per POS tag, one modeling left dependents and one modeling right dependents. Let POS be the set of possible POS tags, and let w be an element in POS; let A_L^w and A_R^w be the two automata associated to it. Let G_L^w and G_R^w be the PCFGs equivalent to A_L^w and A_R^w, respectively, following (Abney et al., 1999), and let S_L^w and S_R^w be the start symbols of G_L^w and G_R^w, respectively. We build a grammar G_D with start symbol S, by defining its metarules as the disjoint union of all rules in G_L^w and G_R^w (for all POS w), its set of pseudorules as the union of the sets $\{S \xrightarrow{s}_1 S_L^v v^* S_R^v : v \in \{\texttt{VB}, \texttt{VBD}, \texttt{VBG}, \texttt{VBN}, \texttt{VBP}, \texttt{VBZ}\}\}$. The grammar is designed in such a way that the grammar's start symbol S only yields the head words of the sentences which are marked with the $*$ symbol. The main difference between the grammar we built in this section and the grammars we built in

Chapters 5 and 6 is that the sentences that are parsed with this grammar have the main head verb marked. We design the grammar to take advantage of this information.

To understand our experiments, we need to take a closer at the probabilities, and specially assigned by the grammars to the tree languages involved. Here is an example. Figure 7.5 shows a tree generated by G_L together with its probability. Here, $p(w_h w_1 w_2)p(w_h w_3 w_4)$ is the probability assigned by the automata to $w_h w_1 w_2$ and $w_h w_3 w_4$. In fact, this is a simplification of the probability; it does not affect the analysis we carry out later but it makes the analysis clearer.



(a)

$$p_D(t) = p(w_h w_1 w_2)p(w_h w_3 w_4) \times p(t_1)\ldots p(t_4)$$

(b)

Figure 7.5: (a) An example of a structure retrieved by the grammar G_D, and (b) the probability value G_D assigns to it.

7.5.2 Grammars for Labeling Main Dependents

The second grammar we build is for labeling dependents. The sentences this grammar process are supposed to be only the dependents of a verb. In other words, the grammar assumes that somehow the right dependents have been identified, the task of this grammar is to assign the correct label to the dependents. It assigns a label to all elements in the the input string. The grammar built in the previous sections selects a set of candidate dependents, and the selected dependents are fed to grammars described in the present section.

Sequences of GRs are modeled as instances of regular languages. Every verb tag has two automata associated to it: one modeling the sequence of left GRs and one modeling sequences of right GRs. A sequence of left (right) GRs is then an instance of the left (right) automata.

The grammar we build in this chapter is similar to the grammars built in Chapters 5 and 6 in that automata are used for building meta-rules. In contrast, automata are used to model sequences of GRs instead of sequences of POS tags. Figure 7.6 shows an example of a possible tree. From the figure, it is clear that GRs are encoded in preterminal symbols. All trees in the tree language defined by this grammar are flat trees of depth two. GRs are at depth one and they are modeled with automata and meta-rules. The yield of the tree is at depth two and it is modeled using pseudo-rules. These

pseudo-rules rewrite GR names into a POS tag and they are read from the tree-bank; their probabilities are computed using the maximum likelihood estimator (Manning and Schütze, 1999).

Observe that these are w-trees, and not CFG trees; all meta-derivations that took place to produce nodes at depth 1 remain hidden. Hence, the sequence of GRs to the right and to the left of the main verb are instances of the regular languages modeling right or left GRs, respectively.

Summing up, we build a grammar for labeling GRs by combining two techniques for estimating probabilities for rules: the techniques presented in Chapters 5 and 6 for estimating probabilities for meta-rules, and maximum likelihood estimators for estimating probabilities for pseudo-rules.

Extracting Training Material

For this grammar we build two automata per verb POS tag, one modeling left GRs and one modeling right GRs. In order to extract the training material required for this grammar, we discarded all information not related to GRs from the transformed PTB. Figure 7.6 shows an example of the information we kept from the tree in Figure 7.2.

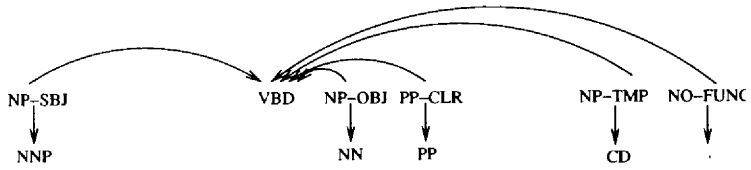


Figure 7.6: The tree representation we use, extracted from tree in Figure 7.2.

From the tree in Figure 7.6 we extract two kinds of information. The first kind is used to model meta-rules yielding GRs, i.e., the first level of the output trees, while the second kind of information is used to model pseudo-rules that rewrite names of GRs into POS tags, i.e., the third level of the output tree.

We first discuss the extraction of the material to build the automata. For this purpose we build two training bags, one containing right GRs and the other containing left GRs. Table 7.2 shows all instances to be added to the training material extracted from the tree in Figure 7.2.

We build two sample bags per verb POS tag, one containing all instances of left sequences of GRs and one containing all instances of right sequences of GRs. For each of the resulting bags we build one automaton. The description of how to build an automaton from a bag of samples, and the steps we follow for optimizing all automata, are discussed in Section 7.6.

VBD	
Left	Right
NP-SBJ VBD	VBD NP-OBJ PP-CLR NP-TMP NO-FUNC

Table 7.2: Data extracted from the tree in Figure 7.2. Left dependents should be read from right to left.

In contrast to Chapters 5 and 6, where probabilities for pseudo-rules were hand-coded, for this grammar probabilities for pseudo-rules have to be estimated from the training material. This is the case because pseudo-rules in Chapters 5 and 6 could be rewritten in only one way. For the present grammar, this is no longer the case. Left hand symbols of pseudo-rules are GRs, and these names can yield different POS tags. In order to estimate probabilities, we extracted all pairs of (GR, POS) from the training material and put them aside in only one bag. Table 7.3 shows the instances of pairs extracted from the tree in Figure 7.2.

GR	POS tag
NP-SBJ	nnp
NP-OBJ	nn
PP-CLR	pp
NP-TMP	cd
NO-FUNC	dot

Table 7.3: Pairs of GRs and POS tags extracted from tree in Figure 7.2.

Building a Grammar for Labeling

For building a grammar for labeling, we have to estimate two sets of probabilities and rules. On the one hand, we have to estimate meta-rules and their probabilities, and on the other hand we have to estimate the probabilities for pseudo-rules. The estimation of meta-rules and their probabilities is done by inducing an automaton for each of the sample sets of sequences of GRs.

Once the training material for meta-rules has been extracted, we build two automata per POS tag, one modeling left sequences of GRs and one modeling right sequences of GRs. Let VS be the set of possible verb tags, let v an element in VS; and A_L^v and A_R^v the two automata associated with it. Let G_L^v and G_R^v be the PCFGs equivalent to A_L^v and A_R^v, respectively, and let S_L^v and S_R^v be the start symbols of G_L^v and G_R^v, respectively. We build a grammar G_L with start symbol S, by defining its meta-rules as the disjoint union of all rules in G_L^v and G_R^v (for all verb POS tags v), and its set of

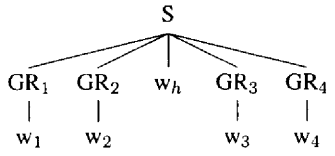
pseudo-rules as the union of the two sets. One set, given by

$$\{S \xrightarrow{s}_1 S_L^v v^* S_R^v : v \in VS\},$$

is in charge of connecting the automata modeling left sequences of GRs with the automata modeling right sequences of GRs. The second set, given by

$$\{GR \xrightarrow{s}_p w : w \in POS\},$$

where GR is the name of a GR, w is a POS tag, and p the probability associated to the rule, is computed using the pairs of (GR, POS) extracted from the training material, using the maximum likelihood estimator.



(a)

$$\begin{aligned} p_L(t) &= p(GR_1 \ldots GR_4) \times \\ &\quad p(GR_1 \xrightarrow{s} w_1) \times \\ &\quad \vdots \\ &\quad p(GR_4 \xrightarrow{s} w_4) \end{aligned}$$

(b)

Figure 7.7: (a) An example of a structure retrieved by the grammar G_L, and (b) its probability value.

Again, the key to understanding our experiments lies in the way G_L assigns probability mass to its grammatical trees. Figure 7.7 shows a tree generated by G_L together with the probability assigned to it. In the figure, $p(GR_i \xrightarrow{s} w_i)$ refers to the probability assigned to the rule $GR_i \xrightarrow{s} w_i$ and $p(GR_1 \ldots GR_4)$ is a simplification of the probability associated to the string $GR_1 \ldots GR_4$ by the automata modeling sequences of GRs.

If the grammar for labeling dependents is fed with the dependents found by the grammar for detecting dependents, the probability associated to a tree like the one pictured in Figure 7.8 is

$$\begin{aligned} p_{cascading}(t) &= p_D(t) \times p_L(t) \\ &= p(GR_1 \ldots GR_4) \times \\ &\quad p(GR_1 \xrightarrow{s} w_1) \ldots p(GR_4 \xrightarrow{s} w_4) \times \\ &\quad p(w_h w_1 w_2) p(w_h w_3 w_4) \times \\ &\quad p(t_1) \ldots p(t_4) \end{aligned} \tag{7.1}$$

That is, $p_{cascading}$ is the product of the probability assigned by G_L and G_D, the main difference between this probability and $p_{one\text{-}go}$ (the one assigned by the grammar G defined in Section 7.5.3 below), is that $p_{cascading}$ uses the probability of the sequences $w_h w_1 w_2$ and $w_h w_3 w_4$ for detecting and labeling dependents.

Summing up, we have two probability distributions for the very same task, one of the distributions uses one more feature. An empirical comparison of these two distributions would provide us with information about the value of the extra feature; this is what we turn to in Section 7.7.

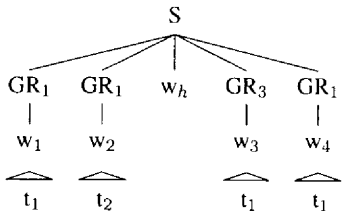


Figure 7.8: The result of cascading the grammars for detecting and labeling dependents.

7.5.3 Grammars for Detecting and Labeling Main Dependents

This grammar to be defined in this section does in one go what the two grammars in Section 7.5.1 and 7.5.2 do in two steps.

Extracting Training Material

The training material we used for building this grammar is the union of the training materials we used for building the two previous grammars.

Building the Grammar

The automata we used for building this grammar are the same as the automata used in the previous two grammars, but the set of rules is different. Let POS be the set of possible POS tags, let w be a an element in POS; let A_L^w and A_R^w be the two automata built for each POS tag in Section 7.5.1. Let VS be the set of possible verb tags, v an element in VS; let A_L^v and A_R^v the two automata built for verb tags in Section 7.5.2. Let G_L^v, G_R^v, G_L^w, and G_R^w be the PCFGs equivalent to A_L^v, A_R^v, A_L^w and A_R^w, respectively, and let S_L^v, S_R^v, S_L^w and S_R^w be the start symbols of G_L^v and G_R^v, respectively. We build a grammar G with start symbol S, by defining its meta-rules as the disjoint union of

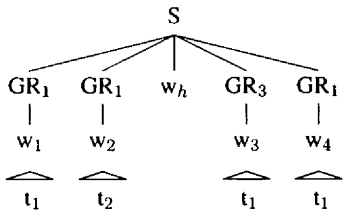
all rules in G_L^v, G_R^v, G_L^v and G_R^v, for all POS tags and all verbs tags, while its set of pseudo-rules is the union of the following sets:

$$\{S \xrightarrow{s}_1 S_L^v v^* S_R^v : v \in VS\}, \{W \xrightarrow{s}_1 S_L^w w S_R^w : w \in POS\}, \text{ and}$$
$$\{GR \xrightarrow{s}_p S_L^w w S_R^w : w \in POS\},$$

where p is the probability assigned to the rule $\{GR \xrightarrow{s}_p w : w \in POS\}$ in Section 7.5.2.

The difference between this grammar and the grammar for detecting dependents is that this grammar uses sequences of GRs for detecting the main dependents, while the grammar for detecting dependents uses sequences of POS tags.

Figure 7.9 shows an example of a tree accepted by G together with the probability G assigns to it.



(a)

$$\begin{aligned} p_{one\text{-}go}(t) &= p(GR_1 \dots GR_2) \times \\ & \quad p(GR_1 \rightarrow w_1) \times \\ & \quad \dots \\ & \quad p(GR_4 \rightarrow w_4) \times \\ & \quad p(t_1) \dots p(t_4) \end{aligned}$$

(b)

Figure 7.9: (a) An example of a structure retrieved by the grammar G, and (b) its probability value.

Now that we have the tree probability distributions we can establish the relation between the two. Let $p_{cascading}$ be the probability distribution generated over trees by cascading the two first grammars, and let $p_{one\text{-}go}$ be exactly the probability distribution generated by G. The probability distributions $p_{one\text{-}go}$ and $p_{cascading}$ assign probabilities to the same set of trees, and the two are related as follows

$$p_{cascading}(t) = p_{one\text{-}go}(t) \times p(w_h w_1 w_2) p(w_h w_3 w_4) \tag{7.2}$$

As is clear from Equation 7.2, the difference between the two distributions is the probability of the sequence of POS tags $w_1 \dots w_4$.

7.6 Optimizing Automata

Let T be a bag of training material extracted from the transformed tree-bank. The nature of T depends on the grammar we are trying to induce. But since we use the

same technique for optimizing all automata, we describe the procedure for a general bag.

Recall from Section 2.2.2 that we use two different measures for evaluating the quality of automata. Let Q be a test bag extracted as T. As before, we use perplexity (PP) and missed samples (MS) to evaluate the quality of a probabilistic automaton. A PP value close to 1 indicates that the automaton is almost certain about the next step while reading the string. MS counts the number of strings in the test sample Q that the automaton failed to accept.

We search for the value of `alpha` that minimizes $q = \sqrt{PP^2 + MS^2}$ (see Chapter 6.4.2, page 103, for the motivation of this function). In Figures 7.10 and 7.11 we have plotted `alpha` vs. PP, MS and q for all verb POS tags used in the grammar for detecting the main dependents of verbs. Table 7.4 shows the values of `alpha` that produce the minimum value of q.

	Alpha	
POS tag	Left	Right
VB	0.0004	0.0004
VBD	0.0004	0.0002
VBG	0.0002	0.0004
VBN	0.0005	0.0004
VBP	0.0004	0.0004
VBZ	0.0004	0.0004

Table 7.4: Optimal values of PP and MS for automata used for labeling dependents.

In Figures 7.12 and 7.13 we have plotted `alpha` vs. PP, MS and q for all verb POS tags used in the grammar for labeling dependents, an instance of the regular languages these automata model are GRs sequences. Table 7.5 shows the values of `alpha` that produce the minimum value of q. Recall from Section 7.5.2 that we build one automaton per verb POS tag.

Note that, even though the PP values for automata modeling sequences of GRs and the PP values for automata modeling POS tags are close to each other, the difference between their MSs is remarkable. We think that data sparseness affects the modeling of GRs much more than the modeling of POS tags. This sparseness prevents the MDI algorithm from inducing a proper language for GRs.

7.7 Experiments

For our experiments we shuffle the PTB sections 10 to 19 into 10 different sets. We run the experiments using set 1 as the test set and sets 2 to 10 as training sets. The tuning

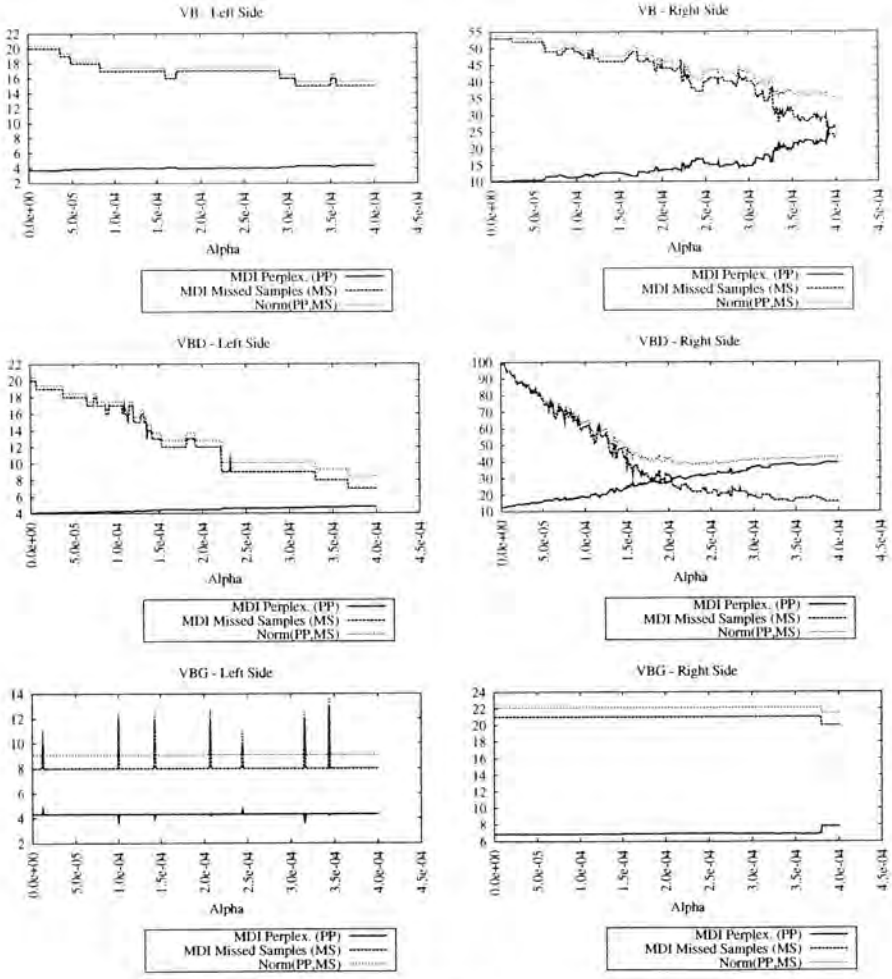


Figure 7.10: Values of PP and MS for automata used for labeling.

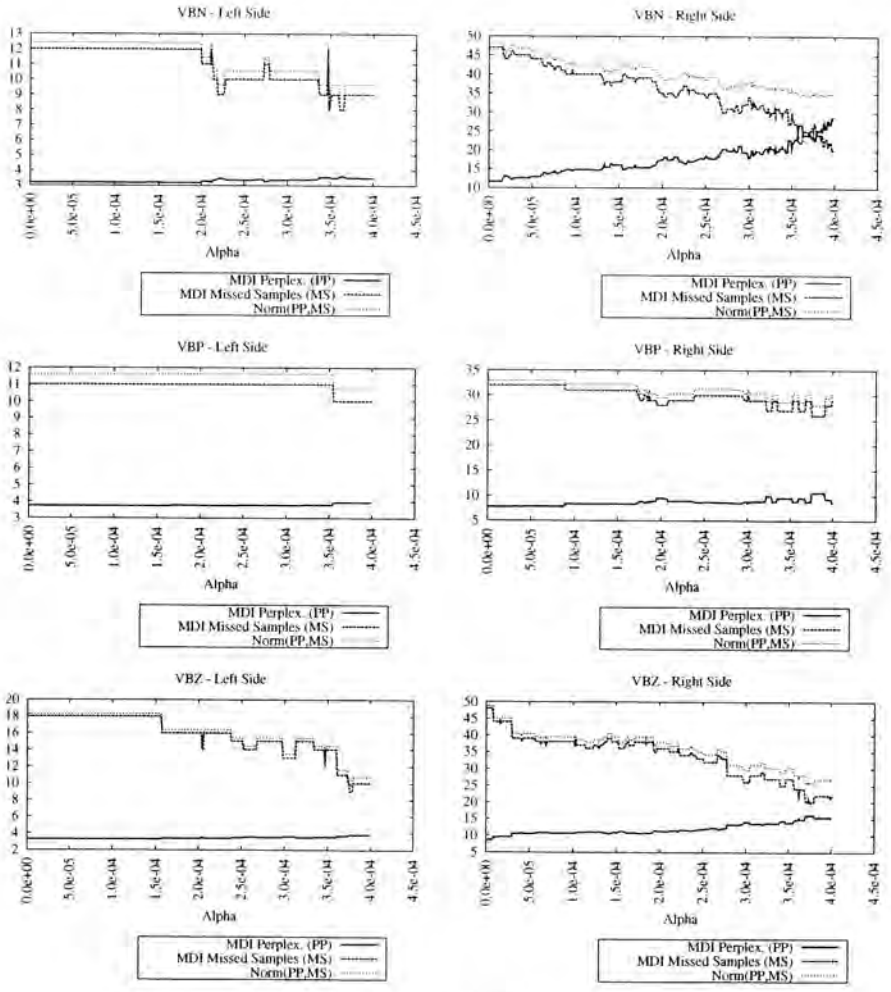


Figure 7.11: Values of PP and MS for automata used for labeling.

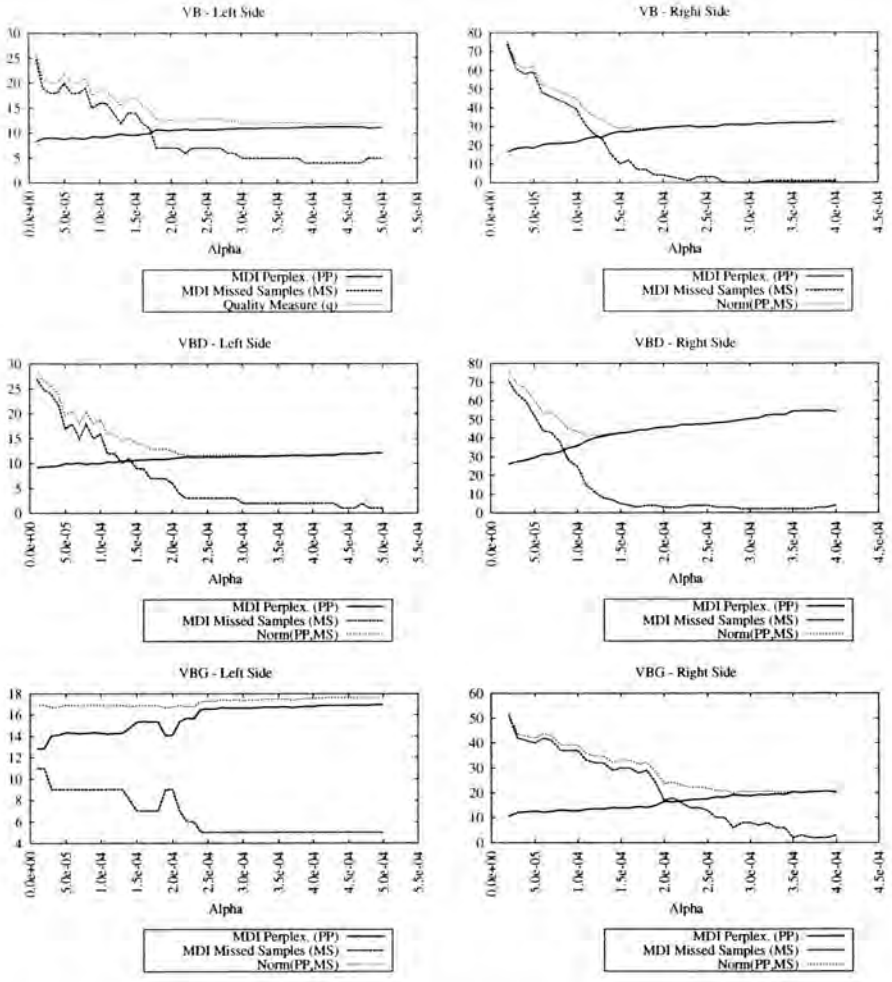


Figure 7.12: Values of PP and MS for automata used for detecting dependents.

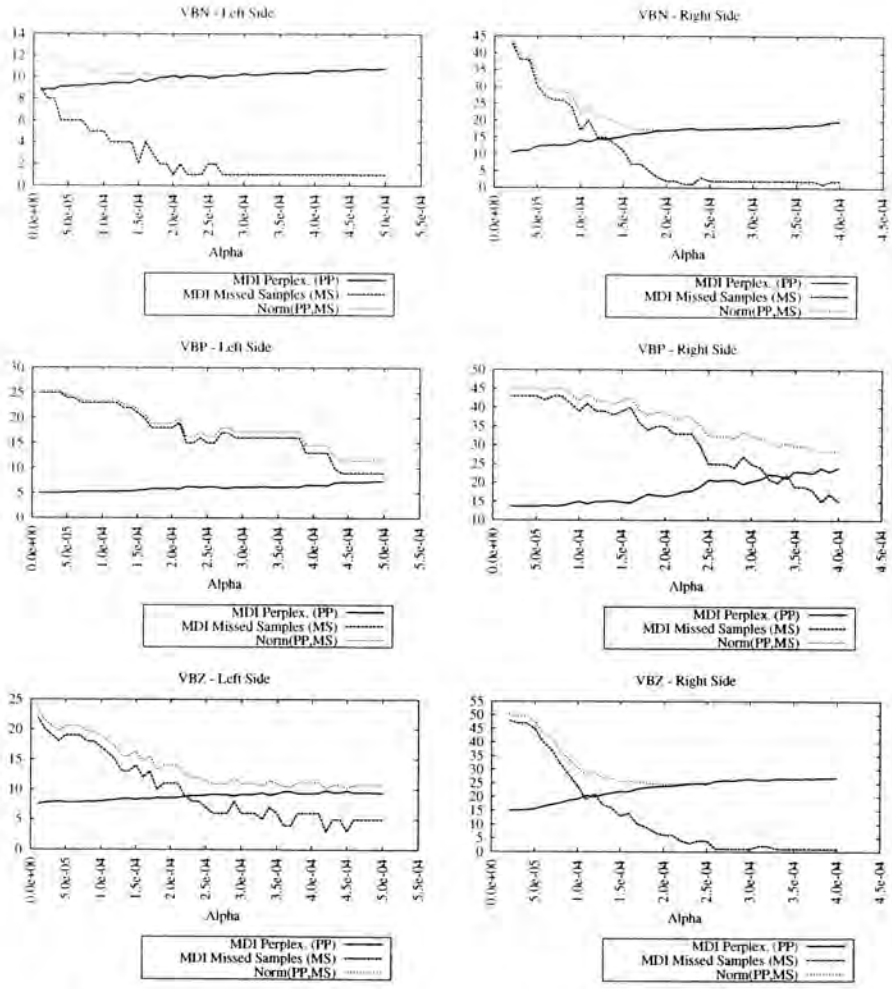


Figure 7.13: Values of PP and MS for automata used for detecting dependents.

	Alpha	
POS tag	Left	Right
VB	0.00015	0.00015
VBD	0.00020	0.00010
VBG	0.00030	0.00020
VBN	0.00020	0.00020
VBP	0.00050	0.00050
VBZ	0.00030	0.00015

Table 7.5: Optimal values of `alpha` for automata used for detecting dependents.

samples were extracted from Section 00. All the sentences we fed to the parser have the main head marked; all sentences whose main head was not tagged as a verb were filtered out.

We start by performing the whole task (detecting dependents and labeling their relation with the main verb) by the two different approaches; results are shown in Table 7.7. These results, together with Equation 7.2, answer one of our main research questions, namely what is the importance of the sequences of POS tags for parsing.

Recall from Equation 7.2 that the only difference between the two probability distributions $p_{one\text{-}go}$ and $p_{cascading}$ is the probability that $p_{cascading}$ associates to sequences of POS tags. Note also that the grammar that does not take it into account, namely G, performs significantly worse than the one that does take this sequence into account. From this we can conclude that the 10% jump in performance is due to the use of this specific information. The grammar G_L for labeling dependents allows us to quantify

Approach	Precision	Recall	$f_{\beta=1}$
Cascading	0.73	0.73	0.73
One Go	0.65	0.67	0.66

Table 7.6: The results on detecting and labeling dependents of main verbs.

how effective are the sequences of GRs together with pseudo-rules $GR \xrightarrow{s} w$ for labeling GRs. To isolate these features, we used grammar G_L for labeling dependents that are known to be the right dependents. We extracted the correct sequences of dependents from the gold standard and used grammar G_L for labeling them. Table 7.7 shows the results of this experiment.

Precision	Recall	$f_{\beta=1}$
0.76	0.76	0.76

Table 7.7: Results of the experiment on labeling gold standard dependents.

The experimental results show that labeling is not a trivial task. The score obtained is low, even more so if we take into account that the sentences we fed to the parser consisted only of correct dependents. We think that the poor performance of this grammar is due to the data sparseness problem mentioned above, because there is a high amount of MS in the automata that model GRs.

The two grammars in the first approach allow us to quantify how the errors percolated from detecting dependents to labeling them. Now, the only aspect of the task that is left is to study is the detection of dependents. Table 7.8 shows the results of the experiment to assess the goodness of G_D for detecting dependents.

Precision	Recall	$f_{\beta=1}$
0.85	0.88	0.86

Table 7.8: Results of the experiment on detecting dependents.

We can see how sensitive the task of labeling dependents is to errors in its input. Table 7.7 states that the labeling precision drops from 0.76 to 0.73 when only the 85% of the arguments fed to the labeling grammar are correct.

7.8 Related Work

The task of finding GRs has usually been considered as a classification task (Buchholz, 2002). A classifier is trained to find relations and to decide the label of the relations that are found. The training material consists of sequences of 3-tuples (main verb, label, and context). In order to have a better impression of the difficulty of the task, Table 7.9 shows some baselines extracted from (Buchholz, 2002). To understand the table, it is important to note that "no relations" refers to the absence of the predicted relation and that 0 divided by 0 is defined as 1. In contrast to approaches based on classifiers, we

Description	Precision	Recall	F_β
always predict "no relation"	100	0	0
always predict NP-SBJ	6.85	30.73	11.20
most probable class for focus chunk type/POS	100	0	0
most probable class for focus word	31.21	1.07	2.07
most probable class for distance	49.43	37.30	42.51

Table 7.9: Some possible baselines. Results extracted from Table 3.2 in (Buchholz, 2002).

consider the task of finding GRs as a parsing task. We build grammars that specifically

try to find GRs. It is possible to find GRs as a side product of full parsing because full trees output by a parser can be transformed as we transformed PTB. In order to give an impression of state-of-the-art methods for finding and labeling main dependents, we compare experiments to the approach presented in (Buchholz, 2002).

Approach	Precision	Recall	F_3
Cascading	0.73	0.73	0.73
One-go	0.65	0.67	0.66
Memory Based Approach	0.86	0.80	0.83

Table 7.10: Comparison to state-of-the-art techniques for detecting and labeling main verb dependents.

The main difference in the scores obtained by Buchholz (2002) and our own approach is probably due to the little information we used for performing the task. In contrast to our approach, Buchholz (2002) uses all kinds of features for detecting and labeling dependents.

7.9 Conclusions and Future Work

In this chapter we designed and implemented experiments for exploring the differences between the regular language of POS tags and the regular language of non-terminal labels in a parsing setup. Our research aimed at quantifying the difference between the two and at understanding their contribution to parsing performance. In order to clearly assess the contribution of these two features, we needed to carry out an evaluation in terms of a task that clearly isolates the two regular languages. We used the task of detecting and labeling dependents of the main verb of a sentence.

We presented two different approaches for dealing with the task of finding grammatical relations. In the first approach, we developed two grammars: one for detecting dependents and another for labeling them. The first grammar used sequences of POS tags as the main feature for detecting dependents, and the second grammar used sequences of GRs as the main feature for labeling the dependents found by the first grammar. The task of detecting and labeling dependents as a whole was done by cascading these two grammars. In the second approach, we built a single grammar that uses sequences of GRs as the main feature for detecting dependents and for labeling them. The task of detecting and labeling dependents as a whole was done in one go by this grammar. The two approaches differ in that the first one used sequences of GRs and sequences of POS tags, while the second only used sequences of GRs.

We showed that English GRs follow a very strict sequential order, but not as strict as POS tags of verbal dependents. We showed that the latter is more effective for de-

tecting and labeling dependents, and, hence, it provides a more reliable instrument for detecting them. Moreover, we have shown that sequences of POS tags are fundamental for parsing performance: they provide a reliable source for predicting and detecting dependents. Our experiments also show that sequences of GRs are not as reliable as sequences of POS tags.

The usual perspective on parsing is that all features that improve parsing performance are used without clearly stating why these features improve. Our approach aims at changing this perspective; we designed grammars and experiments for isolating, testing and explaining two particular features: sequences of POS tags and sequences of GRs, both for detecting and labeling and labeling dependents. PCW-grammars allow us to do these. Note that trees returned by our parser are flat trees and they can not be modeled either by PCFGs nor by bilexical grammars. This is the case because some of the dependencies we model using automata in this chapter do not yield terminals but preterminal symbols.

Chapter 8

Conclusions

In this thesis we have applied an abstract model to learn more about natural language parsing. We have surveyed three state-of-the-art probabilistic parsers and we have identified their characteristic features. We have abstracted from these particular features and models to produce and formalize a general, and abstract, language model. This abstract language model provides us with a suitable framework to carry out principled investigations of new directions into parsing based on different parameterizations of the general model.

While reviewing state-of-the-art parsers, we focused on crucial issues like the role of probabilities in the context of parsing, their importance and their possible uses. We have shown that, when used as a filtering mechanism, probabilities can add expressive power to context free grammars, defining a class of tree languages beyond the expressivity of context free grammars. We have shown that it is not possible to decide whether probabilities solve all ambiguities in a language. In addition, we have shown that probabilities can be used for purposes other than the mere filtering of unwanted trees. We have illustrated this claim with some examples, like using probabilities to evaluate the quality of PCFGs and to boost the performance of parsers.

The general language model we have presented is based on W-grammars. We have introduced constrained W-grammars and have augmented them with probabilities. The resulting formalism, probabilistic constrained W-grammars, is the backbone formalism for all our experiments. Like every general model, the suitability of PCW-grammars is mainly given by two aspects: that they can capture multiple existing formalisms in a single formalism, and that they provide a structured framework where new directions of research can be identified and pursued in a principled way.

8.1 PCW-Grammars as a General Model

We have shown that PCW-grammars provide an encompassing formalism for explaining three state-of-the-art language models. PCW-grammars are based on a well-known grammatical framework, and their computational properties are well-understood. It is true that the expressive power of PCW-grammars is significantly lower than that of W-grammars, but their expressiveness is perfectly adequate to capture grammatical formalisms underlying state-of-the-art parsers.

For example, probabilistic constrained W-grammars are capable of capturing bilexical grammars, Markovian context free rules, and stochastic tree substitution grammars. We have described the expressive power of these three formalisms, together with some conditions under which grammars inferred from treebanks are consistent. Despite the similarities between PCW-grammars and PCFGs, there is a fundamental difference between the two: the two-level mechanism of PCW-grammars. This mechanism allowed us to capture these three state-of-the-art natural language models mentioned above, which cannot be done using standard PCFGs only.

The suitability of the general language model provided by PCW-grammars is that it allowed us to compare three apparently different formalisms within the same formal perspective. We have shown that the essence of bilexical grammars and Markovian context free grammars is quite comparable: both are based on approximating bodies of rules using Markov models. We also found similarities between STSGs and Markov rules: both suppose that rule bodies are obtained by collapsing hidden derivations. More concretely, for Markovian rules, a rule body is a regular expression (or a Markov chain, which is equivalent) and STSGs take this idea to the extreme by considering the whole sentence as the yield of a hidden derivation.

8.2 PCW-Grammars as a New Parsing Paradigm

PCW-grammars are not only useful for capturing the formalisms underlying state-of-the-art parsers, but also for suggesting new research directions. These come as a consequence of different instantiations of the parameters of the general model, or by rethinking the set of assumptions the particular instances have made. A brief description of the directions explored in this thesis follows.

Explicit Use of Probabilistic Automata: PCW-grammars allowed us to use general methods for inducing regular languages instead of the usual n-based algorithms. Our experiments along this line lead to two types of conclusions. First, that modeling rules with algorithms other than n-grams does not only produce smaller grammars, but also better performing ones. Second, that the procedure used for

optimizing the parameters of the parser reveals that some POS behave almost deterministically for selecting their dependents, while others do not. This conclusion suggests that splitting classes that behave non-deterministically into homogeneous ones could improve the quality of the inferred automata. We argued that lexicalization and head-annotation seem to take advantage of the properties of splitting.

Splitting the Training Material: We have presented an approach that aims at finding an optimal splitting of the material before inducing a PCW-grammar. The splitting was aimed at improving parsing performance. For this purpose, we defined a quality measure to quantify the quality of different partitions of the material. Using this measure, we searched among a subset of all possible partitions for the one maximizing the proposed quality measure. This measure is a combination of a quality measure defined for each component in a partition. For each component, we built an automaton and computed the automaton's missed samples and perplexity. The measure we presented combines the values of perplexity and missed samples for all resulting automata. We used the resulting automata to build grammars that were subsequently used for parsing the Penn Treebank. We have shown that the quality measure we defined can be used for comparing two grammars' parsing scores if the grammars are built from partitions having a similar number of components. Since our measure q is a good indicator of the parsing performance, the process of inferring grammars can be treated as an optimization task. This implies that this procedure spares us the need to assess the performance of a particular grammar by parsing.

Sequences as Features: The usual perspective on parsing is that all features that improve parsing performance are used for parsing, without a clear study of *how* these features improve parsing. Our approach is aimed at changing this perspective; we have designed grammars and experiments for isolating, testing and explaining the value of two particular features that are known to improve parsing performance: sequences of POS tags and sequences of GRs. We have shown that sequences of POS tags are fundamental for parsing performance, because they provide a reliable source for predicting and detecting dependents. Our experiments have also shown that sequences of GRs are not as reliable as sequences of POS tags. We think this is the case because the training material for GRs is small compared to the training material for sequences of POS.

PCW-grammars are versatile enough to allow us to address the variety of experimental questions and aspects we tried out in this thesis. For all of them we used only one parsing algorithm and a unified mathematical theory. PCW-grammars have reduced

the design cycle for all of our experiments; we only need to focus on very specific aspects of parsing, and we could leave aside all conditions on expressive power, complexity of parsing, and parsing algorithms. We think that PCW-grammars have proven their suitability: the formalism is abstract enough to capture the formalisms underlying state-of-the-art parsers and to suggest new research directions in parsing.

8.3 Two Roads Ahead

This thesis presents a rather unusual perspective on parsing. The usual perspective aims at designing and building parsers that produce better scores on parsing the Penn treebank. In contrast, we presented measures, grammars, tasks and experiments that were designed for testing particular aspects of syntax, language modeling and parsing.

In my opinion, these two approaches are exponents of two different research directions. The first one focuses on understanding why particular features improve parsing performance, while the second focuses on finding new features that can improve parsing performance. The second direction has reached a plateau; different approaches do not differ substantially in terms of their parsing scores and it is hard to identify the key features that may produce a future jump in performance scores. It is also difficult to determine which differences in performance are statistically significant. I think that in the forthcoming years the research focus will shift from the second line of research to the first one. I think that this shift will result in deeper, more detailed knowledge of the structure of human language, and its impact on parsing performance.

The ultimate aim of parsing is twofold: understanding human language and producing parsers that process naturally ocurring sentences with an acceptable error rate. Depending on the scientific discipline, one of these interests might be more important than the other: linguistics aims at understanding human language, while statistical parsing aims at producing computational models that process natural language with an acceptable level of performance. If there is any flow of ideas here, it seems to go mostly from the linguistic side to the statistical one. The reverse direction is blocked to the point that some linguists claim that the knowledge that can be inferred by statistical methods cannot be considered as a reliable model of language (Andor, 2004). I hope this situation will change in the years to come. I think that the parsing community has to focus on two main research areas. One area will focus on identifying, understanding and testing particular aspects of features used for parsing which will provide linguists with interesting data about language as they understand it. The second stream will try to incorporate insights suggested by the first line of research in parsers. I hope that this thesis has suggested new directions; translating them into improvements in state-of-the-art parsing performance is the next step.

Appendix A
Parsing PCW-Grammars

A.1 Introduction

In this appendix we focus on two aspects of our PCW-grammar parsing algorithm. One aspect, the most theoretical one, is related to the study of the capacity and computational complexity of our implementation for handling PCW-grammars. Since various meta-derivations can produce the same w-rule, it is important to distinguish between the most probable derivation tree and the most probable w-tree. As described in Chapter 4, different meta-derivations can yield the same w-rule and consequently the same w-tree. A parser returning the most probable derivation tree considers the probability of a w-rule as the probability value of the most probable meta-derivation. In contrast, a parsing algorithm searching for the most probable tree considers the probability of a w-rule as the sum of all probabilities assigned to the meta-derivations producing it. In this appendix we investigate the differences between these two approaches, focusing on necessary and sufficient conditions for both approaches to return the same tree.

The second aspect we focus on, is related to technical issues of our PCW-grammar parsing algorithm implementation. In some of the experiments we performed, the parser had to handle grammars containing a number of rules close to one million. The parsing algorithm is an optimization algorithm, it searches for the best solution among a set of possible solutions. At each step in the optimization process, the algorithm builds possible solutions retrieving new rules from the grammar. When working with large grammars, as we did, the complexity of the parsing algorithm becomes unmanageable if the retrieval step takes more than constant time. In this appendix, we briefly describe the approach we followed to minimize the computational costs of this step.

The rest of the appendix is organized as follows. Section A.2 discusses the theoretical issues related to the parsing algorithm, Section A.3 discusses the Java implementation, and Section A.4 concludes the appendix.

A.2 Theoretical Issues

Recall from Chapter 4 that a PCW-grammar is a 6-tuple $(V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ such that:

- V is a set of symbols called *variables*. Elements in V are denoted with over-lined capital letters, e.g., $\overline{A}$,$\overline{B}$,$\overline{C}$.
- NT is a set of symbols called *non-terminals*; elements in NT are denoted with upper-case letters, e.g., X, Y, Z.
- T is a set of symbols called *terminals*, denoted with lower-case letters, e.g.: a, b, c, such that V, T and NT are pairwise disjoint.
- S is an element of NT called *start symbol*.
- $\xrightarrow{m}$ is a finite binary relation defined on $(V \cup NT \cup T)^*$ such that if $x \xrightarrow{m} y$, then $x \in V$. The elements of $\xrightarrow{m}$ are called *meta-rules*.
- $\xrightarrow{s}$ is a finite binary relation on $(V \cup NT \cup T)^*$ such that if $u \xrightarrow{s} v$ then $u \in NT$, $v \neq \epsilon$ and v does not have any variable appearing more than once. The elements of $\xrightarrow{s}$ are called *pseudo-rules*.

Meta-rules and pseudo-rules have probabilities associated to them, see Example A.2.1 for an example of a w-grammar.

A.2.1. EXAMPLE. Let $W = (V, NT, T, S, \xrightarrow{m}, \xrightarrow{s})$ be a W-grammar, where $V = \{\overline{A}, \overline{C}\}$, $NT = \{B, S\}$, $T = \{a, c\}$, $\xrightarrow{m}$ and $\xrightarrow{s}$ as described in Table A.1.

pseudo-rules ($\xrightarrow{s}$)	meta-rules ($\xrightarrow{m}$)
$S \xrightarrow{s}_{0.5} \overline{A}$	$\overline{A} \xrightarrow{m}_{0.5} a\overline{C}$
$S \xrightarrow{s}_{0.5} B$	$\overline{A} \xrightarrow{m}_{0.5} \overline{C}a$
$B \xrightarrow{s}_{0.75} aa$	$\overline{C} \xrightarrow{m}_{1} a$
$B \xrightarrow{s}_{0.25} cc$	

Table A.1: A W-grammar that has a best derivation tree that does not correspond to the most probable tree.

As described in Chapter 4, there are two types of derivations depending on the type of the rules used to produce them. *Meta-derivations* are derivations in which only meta-rules are used, while *w-derivations* are derivations in which only w-rules are used. Since w-rules are built by meta-deriving all variables in a pseudo-rule, there might be

w-rules that are the product of different meta-derivations. We can think of a w-rule as a way to pack all meta-derivations that yield the w-rule, because the probability assigned to the w-rule is the sum of all the possible meta-derivations it covers.

Since a w-rule covers many meta-derivations, the underlying PCFG can not be used for parsing PCW grammars. A parser for PCFGs can be used for parsing PCW-grammars if and only if all possible w-rules cover one and only one meta-derivation. If this is not the case, the w-tree resulting from parsing with the underlying PCFG plus hiding its meta-derivations might not be the w-tree with the highest probability.

In order to better understand this phenomenon, we use the grammar in Example A.2.1. This grammar produces the two w-trees pictured in Figure A.1.a and Figure A.1.b, both of them yielding "aa". Clearly, the most probable w-tree is the tree in Figure A.1.a, given that it is the one with the highest probability.

While trees in (a) and (b) are trees belonging to the forest of the PCW-grammar, trees in (c), (d) and (e) are trees that belong to the forest of the underlying PCFG. The procedure for hiding meta-rules maps trees (c) and (d) to tree (a), and (e) to (b).

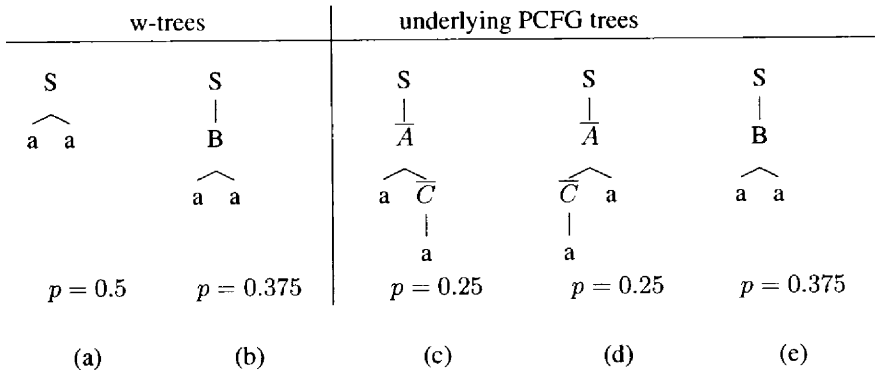


Figure A.1: Trees (a) and (b) belong to the forest of the W-grammar in Example A.2.1, while trees in (c), (d) and (e) are trees in the forest of the PCFG underlying the same W-grammar.

The PCFG parser using the PCFG underlying searches for the best parser among the trees that belong to the forest generated by the PCFG underlying, i.e., the parser searches for the best among the trees in the right-hand side of Figure A.1. Once the best tree is found, it is mapped to a w-tree by hiding all meta-derivations. In this example, the most probable tree in the forest generated by the PCFG underlying is the tree in part (e), which is mapped to the w-tree in part (b). Clearly, the w-tree with the highest probability is the tree in part (a) and not the one in part (b). The algorithm failed in returning the most probable tree. In other words, the PCW parser defined as

the procedure of, first, searching for the most probable tree in the forest generated by the PCFG underlying and, second, hiding all of its meta-derivations, might not return the most probable w-tree.

Clearly, if the hiding procedure maps one tree in the forest generated by the PCFG underlying to one tree in the forest generated by the W-grammar, then a parser for W-grammar is equivalent to the process of using a PCFG parser plus post tree-processing.

There are two configurations for which the mapping between the two forests is not a one-to-one map. The first one occurs when there is at least one meta-variable that can be instantiated with a value that can be meta-derived in two different ways. The second one occurs when there is a pseudo-rule that has only one terminal in its body, and that body can be generated with another w-rule. For the grammar in Example A.1, the mapping between two forests is not a one-to-one mapping because the variable $\overline{A}$ in pseudo-rule $S \xrightarrow{s}_{0.5} \overline{A}$ can be instantiated with two different meta-derivations.

Note that in all our experiments, meta-rules come from probabilistic deterministic automata. Since they are deterministic, all possible variable instantiations have a unique way to derive them. Also, not all the pseudo-rules we used in our grammars have a variable in their body. Consequently, since, for all the grammars we developed in this thesis, there is a one-to-one mapping between the forest generated by the w-grammar and the forest generated by the PCFG underlying, we decided to implement our parser as a Cocke-Younger-Kasami (CYK) parsing algorithm plus a procedure that hides all meta-derivations from the tree returned by the CYK algorithm.

The parser we implemented can be used as a parser that returns the most probable tree, because we know that the most probable tree corresponds to the most probable derivation tree. The problem of knowing when these two trees are the same is not a trivial one. As we discussed before, the two trees are the same tree if there are no variables that can be instantiated in the same way with two different derivations. Since variables are instantiated through a context free like system, the problem of knowing whether there are two ways to derive the same string becomes equivalent to the problem of knowing whether a context free grammar is unambiguous. It is well-known that the latter is an undecidable problem, which implies that it is is undecidable whether the most probable tree is the same as the most probable derivation tree for a given grammar.

A.3 Practical Issues

Conceptually, our parsing algorithm consists of two different modules. One module, a CYK parser, searches for the most probable derivation in the underlying PCFG (see Chapter 4 for the definition of the underlying PCFG). The second module, the function

devoted to hiding meta-derivations, is in charge of transforming the most probable derivation tree into a w-tree.

The rules handled by the version of the CYK algorithm we implemented have an number associated to them. This integer, called *level of visibility*, is a generalization of the concept of meta-rules and pseudo-rules. The tree returned by the CYK algorithm is transformed to many different trees, depending on the visibility level of the rules to be hidden.

A.3.1 Levels of Visibility

Trees to be transformed can be thought of as trees in which, for each node, there is an integer marking the node's level of visibility. In order to transform a tree by hiding a level of visibility, we implemented a function that takes two arguments, one argument is the tree to be transformed and the second argument is the level of visibility to be hidden. The algorithm traverses the tree in a bottom up fashion and, for each node having the visibility level to be replaced, it replaces the node itself with the sub-trees hanging from that node in the original tree. Figure A.2 shows an example of hiding operations for different levels of visibility. Since the hide operation can be applied to a tree which was already transformed, a tree has many possible sets of visibility levels to hide. For example, the tree in Figure A.2.d is the result of hiding level of visibility 1 from the tree in Figure A.2.c, which is in turn the result of hiding visibility level 2 from the tree in Figure A.2.a. Note that the order in which the visibility levels are applied does not matter.

The PCW parser is a particular case of the parser we implemented. In order to obtain a PCW-parser, we marked meta-rules with visibility level 1 and pseudo-rules with visibility level 0. In order to obtain a w-tree from the tree output by the CYK component, we hide nodes whose level of visibility equals 1.

A.3.2 Optimization Aspects

The core of our algorithm is a Probabilistic CYK parsing algorithm capable only of parsing grammars in Chomsky Normal Form (CNF). Probabilistic CYK parsing was first described by Ney (1991), but the version we discuss here is adapted from Collins (1999) and Aho and Ullman (1972).

The CYK algorithm assumes the following input, output and data structures.

- **Input**
  - A CNF PCFG. Assume that the $|N|$ non-terminals have indices 1, 2, ..., $|N|$, and that the start symbol S has index 1.

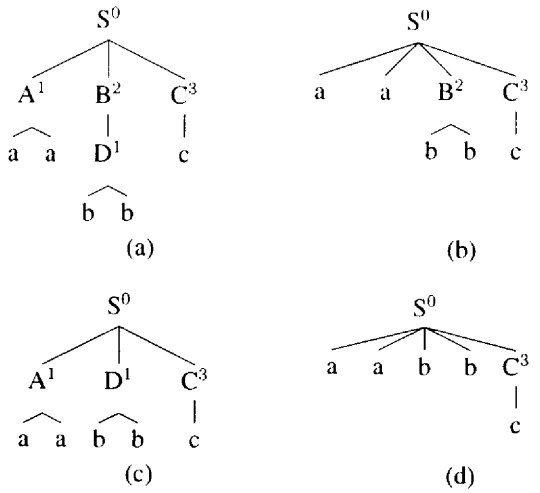


Figure A.2: (a) A tree with its nodes augmented with visibility levels. (b) Level of visibility 1 hidden. (c) Level of visibility 2 hidden. (d) Levels of visibility 1 and 2 hidden.

  - n words $w_1, \ldots, w_n$.

- **Data structures.** A dynamic programming array $\pi[i, j, a]$ holds the maximum probability for a constituent with non-terminal index a spanning words $i \ldots j$.

- **Output.** The maximum probability parse will be $\pi[1, n, 1]$: the parse tree whose root is S and which spans the entire string of words $w_1, \ldots, w_n$.

The CYK algorithm fills out the probability array by induction. Figure A.3 gives the pseudo-code for this probabilistic CYK algorithms as it appears in (Jurafsky and Martin, 2000).

Note that steps 10, 11 and 12 are actually building all possible rules that can be built using the non-terminals of the grammar. In order to minimize the number of iterations, we iterate only on those rules that actually belong to the grammar and that can help building the solution. In order to achieve this, we implement our grammars as dictionaries indexed on bodies of rules. This approach is easy to implement because bodies of rules are of length one or two. Unfortunately, this modification does not reduce the worst case complexity, because in that case the grammar contains all the possible rules that can be built with its set of non-terminals.

In order to parse with a grammar that is not in CNF, our parsing algorithm first transforms the given grammar into an equivalent grammar in CNF. Clearly, since the

function CYK($words$,$grammar$) **returns** The most probable parse and its probability.

Create and clear $\pi[num\_words, num\_words, num\_nonterminals]$ {Base case}
for $i \leftarrow 1$ to num_words **do**
  for $A \leftarrow w_i$ to $num_nonTerminals$ **do**
    if $A \rightarrow w_i$ is in the grammar **then**
      $\pi[i, i,] \leftarrow P(A \rightarrow w_i)$
{recursive case}
for $span \leftarrow 2$ to num_words **do**
  for $begin \leftarrow 1$ to $num\_words - spam + 1$ **do**
    $end \leftarrow begin + span - 1$
    for $m = begin$ to $end - 1$ **do**
      for $A = 1$ to $num\_nonterminals$ **do**
        for $B = 1$ to $num\_nonterminals$ **do**
          for $C = 1$ to $num\_nonterminals$ **do**
            $prob = \pi[begin, m, B] \times \pi[m + 1, end, C] \times P(A \rightarrow BC)$
            if $prob > \pi[begin, end, A]$ **then**
              $\pi[begin, end, A] = prob$
              $back[begin, end, A] = \{m, B, C\}$
return $buildtree(back[1, num\_words, 1], \pi[1, num\_words, 1])$

Figure A.3: The probabilistic CYK algorithm for finding the maximum probability parse of a string of num_words words given a PCFG grammar with num_rules rules in Chomsky Normal Form. *back* is an array of back-pointers used to recover the best parse.

algorithm parses with a CNF, it will return a tree generated by the CNF grammar and not by the original grammar. Since we are interested in the tree generated by the grammar before it was transformed to CNF, we use a new level of visibility j in all the rules that were added during the transformation into CNF process. In order to obtain the tree in the original grammar, we hide the level of visibility j.

Summing up, our parsing algorithm consists of the following items:

1. **A translation module**: An algorithm that transforms any grammar into CNF.
2. **A parsing module**: A CYK parsing algorithm for CNF grammars.
3. **A post processing module**: An algorithm that hides levels of visibility in a tree.

We compute now the computational complexity of our parsing algorithm. For this purpose we only take into consideration items (2) and (3). Since item (1) is done one

time for each grammar, we only consider it indirectly: we take into consideration how the transformation to CNF affects the size of the grammar.

We want to compute the computational complexity of the three items whenever a grammar G is used to parse a sentence s.

By (Ney, 1991), the computational complexity of the CYK algorithm for parsing a sentence s using grammar G is $2nQ + (n^3/6)R$, where n is the length of the sentence, Q the number of preterminal rules in the grammar, and R the number of rules in the grammar. Note that the number of rules and preterminal rules refer to the transformed grammar.

According to (Hopcroft and Ullman, 1979), if a grammar G with R rules is transformed to CNF, the resulting grammar contains $O(R^2)$ rules. The complexity of the post processing time depends on the number of rules in the CNF tree. Since a CNF tree yielding a sentence of length n has $\sum_{i=1}^{n-1} i$ rules, the post processing step takes time $\sum_{i=1}^{n-1} i$. Finally, the complexity for the whole algorithm becomes $2nQ + (n^3/6)R^2$.

A.4 Conclusions

In this appendix we have dealt with two particular aspects of our implementation. First, we showed that it may happen that our implementation does not always return the most probable tree. We also showed that there are some grammars for which our grammar does return the most probable tree. For all grammars used in this thesis, however, the parser does return the most probable tree. We also showed that the problem of determining whether the parser returns the most probable tree or the most probable derivation tree is undecidable.

The second aspect we focused on was related to the actual implementation of the parser. We decomposed our implementation into three different modules, we gave the computational complexity of each of them, and showed that the computational complexity of the whole algorithm is $O(n^3R^2)$, where n is the length of the sentence and R the size of the grammar.

Appendix B

Revising the STOP Symbol for Markov Rules

B.1 The Importance of the STOP Symbol

In this section we discuss Section 2.4.1 of Collins PhD thesis, where he discusses the importance of the STOP symbol for generating Markovian sequences. The idea is that any sequence of strings that is generated by a Markovian process should end in a STOP symbol. He argues that without the STOP symbol the probability distribution generated by a Markovian process over finite strings is not really a probability distribution.

We argue that the even though the STOP symbol is indeed important, the justification Collins provides is not fully correct. We show that the mere existence of the STOP symbol is not enough to guarantee consistency. We argue that probability distributions generated by Markovian process over finite sequences of symbols should be thought of as probability distributions defined over infinite sequences of symbols instead of probability distributions defined over finite sequences.

In Section B.2 we present Collins's explanation on STOP symbols; in Section B.3 we provide background definitions on Markov chains; in Section B.4 we use Markov chains for rethinking the importance of STOP symbols, and in Section B.5 we conclude the appendix.

B.2 Collins's Explanation

Suppose we want to assign a probability p to sequences of symbols $w_1, w_2, w_3, \ldots, w_n$, where each symbol w_i belongs to a finite alphabet Σ.

We first rewrite the probability of the given sequence using the chain rule of prob-

abilities as:

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1}). \tag{B.1}$$

Second, we use m-order Markovian independence assumptions and the equation above becomes:

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-m}). \tag{B.2}$$

Collins (1999, page 46) argues that such a definition of probabilities is not correct: the problems arises because n, the sentence length, is variable. In his argumentation, he states that Equation B.1 would be correct if the event space under consideration would have been the space of n-dimensional vectors Σ^n instead of the set of all strings in the language Σ^*. Writing the probability under consideration as $P(w_1, w_2, w_3, \ldots, w_n)$ implies that Σ^n is the event space. To avoid this confusion he writes the probability of a sequence $\langle w_1, w_2, \ldots, w_n \rangle$ as $P(\langle w_1, w_2, \ldots, w_n \rangle)$: the angled braces imply that $\langle w_1, w_2, \ldots, w_n \rangle$ is a sequence of variable length rather than an n-dimensional vector.

He states that in the case of speech recognition, the length n of strings is often large and that STOP probabilities in that case may not be too significant. In Collins's use of Markov processes, the sequences under consideration are typically of length 0, 1 or 2, and for this case the STOP probabilities certainly become important.

He presents the following example to support his point on the failing of Equation B.1

B.2.1. EXAMPLE. Consider the following.

- Assume $\Sigma = \{a, b\}$, and therefore that Σ^* is $\{\epsilon, a, b, aa, bb, ab, bb, \ldots\}$.
- Assume that we will model the probability over Σ^* with a 0'th order Markov process, with parameters $P(a) = P(b) = 0.5$

We can now calculate the probability of several strings using the formula in Equation B.2: $P(\langle a \rangle) = 0.5$, $P(\langle b \rangle) = 0.5$, $P(\langle aa \rangle) = 0.5^2$, $P(\langle bb \rangle) = 0.25$ and so on. We already see from these 4 probabilities that the sum, over the event space will be greater than 1: $P(\langle a \rangle) + P(\langle b \rangle) + P(\langle aa \rangle) + P(\langle bb \rangle) = 1.5!$. An additional problem is that the probability of the empty string, $P(\langle \rangle)$, where $n = 0$, is undefined.

Collins argues that adding STOP symbols fixes this inconsistency. He adds STOP symbols with the parameters of the Markov process modified to include. For example, let $P(a) = P(b) = 0.25$, $P(\texttt{STOP}) = 0.5$). In this case we have $P(\langle \texttt{STOP} \rangle) = 0.5$, $P(\langle a\texttt{STOP} \rangle) = 0.25 * 0.5 = 0.125$, $P(\langle b\texttt{STOP} \rangle) = 0.25 * 0.5 = 0.125$, $P(\langle aa\texttt{STOP} \rangle) = 0.25^2 * 0.5 = 0.03125$, $P(\langle bb\texttt{STOP} \rangle) = 0.03125$ and so on. Thus far the sum of

probabilities does not exceed 1, and the distribution looks much better behaved. We can prove that the sum over all sequences is 1 by noting that the probability of any sequence of length n is $0.25^n * 0.5$, and that there are 2^n sequences of length n, giving:

$$\begin{aligned}
\sum_{W \in \Sigma^*} &= \sum_{n=0}^{\infty} 2^n * 0.25^n * 0.5 \\
&= \sum_{n=0}^{\infty} 0.5^n * 0.5 \\
&= \sum_{n=0}^{\infty} 0.5^{n+1} \\
&= \sum_{n=0}^{\infty} 0.5^n \\
&= 1.
\end{aligned}$$

In a 0'th order Markov process the distribution over length of strings is related directly to $P(\texttt{STOP})$ — the probability of a string having length n is the probability of generating n non-STOP symbols followed by the STOP symbol:

$$P(length = n) = (1 - P(\texttt{STOP}))^n * p(\texttt{STOP}).$$

With higher order Markov processes, where the probability is conditioned on previously generated symbols, the conditional probability $P(\texttt{STOP}|w_{i-m}, \ldots, w_{i-1})$ encodes the preference for certain symbols or sequences of symbols to end or nor to end a sentence. For example, if we were building a bigram (1st order Markov) model of English we would expect the word *the* to end a sentence very rarely, and the corresponding parameter $P(\texttt{STOP}|the)$ to be very low. The STOP symbol not only ensures the probability distributions to be well defined, but also to have useful interpretation.

B.3 Background on Markov Chains

Markovian processes like the one described in Example B.2.1 are better described through Markov chains (Taylor and Karlin, 1998).

B.3.1. DEFINITION. A *Markov chain* is 3-tuple $M = (W, P, I)$ where W is a set of states, P is a real $|W| \times |W|$ matrix with entries in $\mathbb{R}$, such that $\sum_{j=1}^{|W|} p_{ij} = 1$; p_{ij} is the probability of jumping from state i to state j, and I is $|W|$-dimensional vector defining the initial probability distribution.

B.3.2. EXAMPLE. Let $M = (W, P, I)$ be a Markov chain where $W = \{a, b\}$,

$$P = \begin{array}{c|cc} & a & b \\ \hline a & 0.5 & 0.5 \\ b & 0.5 & 0.5 \end{array}$$

and $I = (0.5, 0.5)^t$. The graphical representation for this Markov chain is in Figure B.1

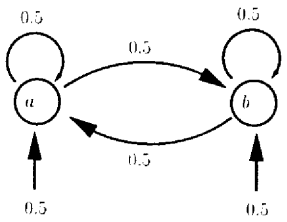


Figure B.1: A graphical representation of the model in Example B.2.1

Markov chains may be viewed as discrete stochastic processes. A discrete stochastic process is a distribution over an infinite sequence of the random variables, each taking a value out of a finite set. We say that the process is *Markovian* if the outcome of a particular random variable in the sequence depends only on its two neighbors (the one before it and the one after it in the sequence).

The following definition formalizes this idea.

B.3.3. DEFINITION. A stochastic process $\{W_0, W_1, \ldots, W_n, \ldots\}$ at consecutive points of observations $0, 1, \ldots, n, \ldots$ is a *discrete Markov process* if, for all $n \in \mathbb{N}$, $w_n \in W$

$$P(W_{n+1} = w_{n+1} | W_n = w_n, W_{n-1} = w_{n-1}, \ldots, W_0 = w_0) = \tag{B.3}$$
$$P(W_{n+1} = w_{n+1} | W_n = w_n) \tag{B.4}$$

Let $W = \{w_1, \ldots, w_n\}$. The quantities

$$p_{ij} = P(W_{n+1} = w_j | W_n) = P(W_1 = w_j | W_0 = w_i)$$

are known as the *transition probabilities* of the Markov process.

B.3.4. DEFINITION. A state W in a Markov chain is called *absorbing* if it does not have any outgoing arc, or, equivalently, every outgoing arc points to the state itself.

Markov chains have been widely used in software modeling for describing probability distributions over *infinite* sequences of states (Infante-Lopez et al., 2001). Under these approach, every string accepted by a Markov chain is in fact an infinite string, and it describes an infinite path in the Markov chain. To define the probability distribution, sets of infinite strings are used as a building block. Each block is characterized by a finite string α, and it contains all infinite strings starting with prefix α that are accepted by the Markov chain. The probability assigned to an α block, by a Markov chain M is defined as the probability of traversing the path described by α in M. The probability associated to a path $X_1, \ldots, X_k$ is defined as $I(X_1) \times \prod_{i=1}^{i=k-1} P(X_i, X_{i+1})$.

The probability of a *finite* sequence α of length n is defined as the probability of the infinite sequence $\alpha\beta$ where β is an infinite sequence consisting of the STOP symbol, i.e., $\beta = \texttt{STOP}^\omega$. The intuition underlying this definition is that the Markov chains reach a state from which it can not leave, and consequently, the β cycles for ever in the same state.

Under this perspective, the Markov chain in Example B.2.1 does *not* generate any distribution over finite sequences. But it does generate a distribution over infinite sequences. It is interesting to note that every single infinite path receives probability zero, but sets of infinite paths do receive probability values greater than zero. This situation is comparable to probability distributions over real numbers, where each single number receives a probability zero but where subsets of real numbers receive non-zero probability values.

B.4 The STOP Symbol Revisited

Collins Example B.2.1 has a direct translation into a Markov chains. We can define a Markov chain with states $W = \{a, b\}$, and transition matrix

$$P(W_{n+1} = w_{n+1} | W_n = w_n, W_{n-1} = w_{n-1}, \ldots, W_0 = w_0) = \tag{B.5}$$

$$P(W_{n+1} = w_{n+1} | W_n = w_n) \tag{B.6}$$

and initial distributions $I = (0.5, 0.5)$. Figure B.1 presents this Markov chain in a graphical way. This Markov chain generates a well defined probability distribution over infinite sequences of states, and since there is no absorbing state, it can not be used for deriving probability distributions over finite sequences.

The solution suggested by Collins adds a new state named STOP. Again, the solution can be described using a zero order Markov chain; the corresponding chain is pictured in Figure B.2.

As Collins suggests, the probability of the STOP state is 0.5, since it is a zero order Markov model it has two fundamental properties: first, all incoming edges have the

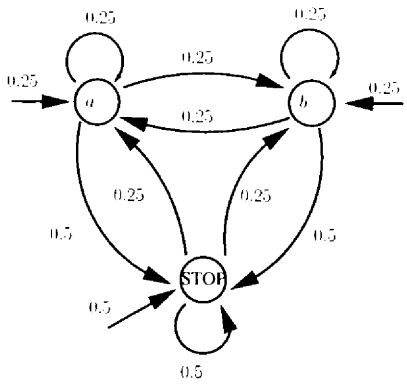


Figure B.2: A zero order Markov chain using the STOP symbol

same probability and second there is a directed arc connecting every pair of states. The resulting Markov chain has a serious problem. It assigns probability mass to strings like "aaSTOPabbSTOP", i.e., strings containing the STOP symbol more than once.

The situation we have so far is that, even though Collins's solution ruled out probability distributions over "correct strings" to sum up above one, the solution proposed produces distributions that sum up below one. We would like to disallow this kind of distributions, a way to solve this particular problem is to modify Figure B.2 into Figure B.3. This Markov chain is a first order Markov model, even more it can *not* be define using 0 order Markov chains.

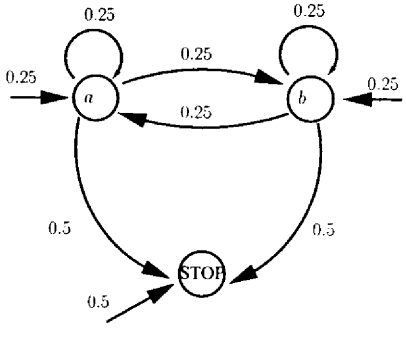


Figure B.3: A Markov chain with the STOP state absorbing.

Actually, it seems that Collins assumes that the STOP state is an absorbing state. His model is a zero order Markov chain plus the condition that generation stops once the STOP symbol has been generated. His solution can be restated as "there has to be a

STOP symbol and it has to be absorbing."

But again, this is not a proper solution, the mere presence an absorbing STOP symbol is not enough to rule out inconsistent models. There can be Markov chains with a STOP symbol, that still produce wrong distributions; an example is shown in Figure B.4.

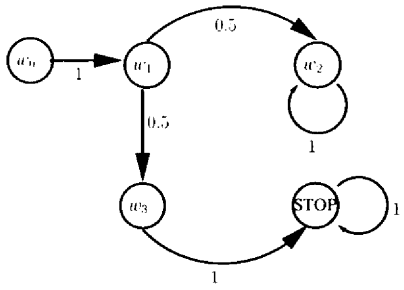


Figure B.4: A Markov chain with an absorbing STOP symbol that defines an inconsistent probability distribution over finite strings.

The model assigns probability 0.5 to the infinity sequence $w_0w_1w_2w_2w_2w_2w_2w_2\ldots$, and 0.5 to the finite string $w_0w_1w_3$. The problem is that the model will enter into the infinite cycle producing w_4 with probability 0.5.

Summing up: In principle the formalism presented by Collins has a potential probability inconsistency. But Collins's model is still on the safe side because his Markov models are learned from data and the following lemma applies:

B.4.1. LEMMA. *A Markov model learned from strings augmented with the* STOP *symbol generate consistent probability distributions over finite strings.*

The lemma has already been proven in Section 4.2.2.

B.5 Conclusions

We presented a different perspective for the presence of the STOP symbols in the generation of finite sequences of strings. We show that formally a Markov chain of order zero can not have an absorbing STOP state. We show that Collins explanation of STOP symbols is not fully correct and we show that independently of his explanation, the Markov chains he induces are consistent.

Bibliography

Abney, S., 1996. Statistical methods and linguistics. In J. Klavans and P. Resnik (eds.), *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*. Cambridge, MA: The MIT Press.

Abney, S., 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23 (4):597–618.

Abney, S., D. McAllester, and F. Pereira, 1999. Relating probabilistic grammars and automata. In *Proceedings of the 37th Annual Meeting of the ACL*. Maryland.

Aho, A. and J. Ullman, 1972. *The Theory of Parsing, Translation and Compiling*, volume I. Prentice-Hall Series in Automatic Computation.

Alshawi, H., 1996. Head automata and bilingual tiling: Translation with minimal representations. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz.

Andor, J., 2004. The master and his performance: An interview with Noam Chomsky. *Intercultural Pragmatics*, 1(1).

Atsumi, K. and S. Masuyama, 1998. On the ambiguity reduction ability of a probabilistic context-free grammar. *IEICE Transanctions on Fundamentals of Electronics, Communications and Computer Sciences*, E81-A(5):825–831.

Berger, A., S. Della Pietra, and V. Della Pietra, 1996. A maximum entropy approach to natural language processing. *Journal of Computational Linguistics*, 22(1):39–71.

Bikel, D., 2004. Intricacies of Collins' parsing model. *Computational Linguistics*. To appear.

Black, E., F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos, 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of 31st Annual Meeting of the ACL*. Ohio.

Bod, R., 1995. *Enriching Linguistics with Statistics: Performance models of Natural Language*. Ph.D. thesis, University of Amsterdam, The Netherlands.

Bod, R., 1998. *Beyond Grammar: An Experience-Based Theory of Language*. Stanford: CSLI Publications.

Bod, R., 2003. An efficient implementation of a new DOP model. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*. Budapest.

Booth, T. and R. Thompson, 1973. Applying probability measures to abstract languages. *IEEE Transaction on Computers*, C-33(5):442–450.

Brew, C., 1995. Stochastic HPSG. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*. Dublin.

Brill, E., 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*. Ohio.

Brown, P., V. Della Pietra, P. de Souza, J. C. Lai, and R. L. Mercer, 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Buchholz, S., 2002. *Memory-Based Grammatical Relation Finding*. Ph.D. thesis, Universiteit van Tilburg.

Caraballo, S. and E. Charniak, 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24-2:275–298.

Cardie, C., 1993a. A case-based approach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of the 11th National Conference on Artificial Intelligence. AAAI Press / MIT Press*.

Cardie, C., 1993b. Using decision trees to improve case-based learning. In *Proceedings of the 10th International Conference on Machine Learning*.

Carrasco, R. and J. Oncina, 1994. Learning sthocastic regular grammars by means of state merging method. In *Grammatical Inference and Applications*, Springer Lecture Notes in Artificial Intelligence. Berlin: Springer-Verlag.

Carrasco, R. and J. Oncina, 1999. Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, 33(1):1–20.

Carreras, X., L. Màrquez, V. Punyakanok, and D. Roth, 2002. Learning and inference for clause identification. In *Proceedings of the 14th European Conference on Machine Learning*.

Carrol, J., 1993. *Practical Unification-based Parsing of Natural Language*. Ph.D. thesis, Computer Lab., University of Cambridge.

Carroll, J., T. Briscoe, and A. Sanfilippo, 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*. Granada, Spain.

Carroll, J. and E. Charniak, 1992. Two experiments on learning probabilistic dependency grammars from corpora. In C. Weir, S. Abney, R. Grishman, and R. Weischedel (eds.), *Working Notes of the Workshop Statistically-Based NLP Techniques*. Menlo Park.

Carroll, J. and A. Fang, 2004. The automatic acquisition of verb subcategorisations and their impact on the performance of an HPSG parser. In *Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP)*.

Charniak, E., 1995. Parsing with context-free grammars and word statistics. Technical Report CS-95-28, Department of Computer Science, Brown University, Providence.

Charniak, E., 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence*. Menlo Park: AAAI Press/MIT Press.

Charniak, E., 1999. A maximum-entropy-inspired parser. In *Technical Report CS-99-12*. Providence, Rhode Island.

Charniak, E., 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings ANLP-NAACL'2000, Seattle, Washington*.

Chastellier, G. and A. Colmerauer, 1969. W-Grammars. In *Proceedings of the 24th National Conference*.

Chaudhuri, R. and A. N. V. Rao, 1986. Approximating grammar probabilities: Solution of a conjecture. *Journal of the ACM.*, 33(4):702–705.

Chen, S., 1995. Bayesian grammar induction for language modeling. In *Proceedings of the 33rd Annual Meeting of the ACL*. Morristown, USA.

Chi, Z. and S. Geman, 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Coello Coello, C., 1999. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 3(1):269–308.

Collins, M., 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*.

Collins, M., 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the ACL and the 8th Conference of the European Chapter of the ACL*. Madrid, Spain.

Collins, M., 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, PA.

Collins, M., 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 7th International Conference on Machine Learning (ICML)*. Stanford.

Collins, M., 2001. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *Proceedings of the 7th International Workshop on Parsing Technologies*. Beijing.

Collins, M. and N. Duffy, 2001. Convolution kernels for natural language. In *Proceedings of Neural Information Processing Systems (NIPS 14)*.

Collins, M. and N. Duffy, 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the ACL*.

Cortes, C. and M. Mohri, 2000. Context-free recognition with weighted automata. *Grammars*, 2-3(3).

Cover, T. and J. Thomas, 1991. *Elements of Information Theory*. New York: John Wiley and Sons.

Daelemans, W., S. Buchholz, and J. Veenstra, 1999. Memory-based shallow parsing. In *Proceedings of the Computational Natural Language Learning Workshop (CoNLL99)*. Bergen.

Daelemans, W., A. van den Bosch, and A. Weijters, 1997. Igtree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

Decadt, B. and W. Daelemans, 2004. Verb classification — machine learning experiments in classifying verbs into semantic classes. In *Proceedings of the LREC 2004 Workshop Beyond Named Entity Recognition - Semantic Labelling for NLP Tasks.*

Denis, F., 2001. Learning regular languages from simple positive examples. *Machine Learning*, 44(1/2):37–66.

Dupont, P. and L. Chase, 1998. Using symbol clustering to improve probabilistic automaton inference. In V. Honavar and G. Slutzki (eds.), *Proceedings of the Fourth International Colloquium on Grammatical Inference*, Lecture Notes in Computer Science. Springer-Verlag.

Dupont, P., F. Denis, and Y. Esposito, 2004. Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition: Special Issue on Grammatical Inference Techniques & Applications*. To appear.

Dupont, P., L. Miclet, and E. Vidal, 1994. What is the search space of the regular inference? In R. Carrasco and J. Oncina (eds.), *Grammatical Inference and Applications; 2nd International Colloquium, ICGI-94*, Lecture Notes in Computer Science. Springer-Verlag.

Eisner, J., 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of 16th International Conference on Computational Linguistics (COLING)*. Copenhagen, Denmark.

Eisner, J., 2000. Bilexical grammars and their cubic-time parsing algorithms. In H. Bunt and A. Nijholt (eds.), *Advances in Probabilistic and Other Parsing Technologies*. Kluwer Academic Publishers, pages 29–62.

Eisner, J. and G. Satta, 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the ACL*. Maryland.

Gaifman, G., 1965. Dependency systems and phrase-structure systems. *Information and Control*, 8(3):304–337.

Galen, A., T. Grenager, and C. Manning, 2004. Verb sense and subcategorization: Using joint inference to inprove performance on complementary tasks. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Gen, M. and R. Cheng, 1997. *Genetic Algorithms and Engineering Design*. New York: John Wiley and Sons.

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

Goodman, J., 1997. Probabilistic feature grammars. In *Proceedings of the 5th International Workshop on Parsing Technologies*. MIT, Cambridge, MA.

Goodman, J., 1998. *Parsing Inside-Out*. PhD thesis, Departement of Computer Science, Harvard University, Cambridge, Massachusetts.

Hara, T., Y. Miyao, and J. Tsujii, 2002. Clustering for obtaining syntactic classes of words from automatically extracted ltag grammars. In *Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*.

Hemphill, C., J. Godfrey, and G. Doddington, 1990. The ATIS Spoken Language Systems pilot corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*. Hidden Valley, Pa.

Henderson, J. and E. Brill, 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing (EMNLP) and Very Large Corpora (VLC)*. Maryland.

Hermjakob, U. and R. J. Mooney, 1997. Learning parse and translation decisions from examples with rich context. In *Proceedings of the 35th Annual Meeting of the ACL and 8th Conference of the European Chapter of the ACL*. Madrid.

Hopcroft, J. and J. Ullman, 1979. *Introduction to Automata Theory, Lanaguges, and Computation*. Reading, MA: Addison Wesley.

Horning, J. J., 1969. A study of grammatical inference. Unpublished doctoral dissertation, Standford University.

Infante-Lopez, G. and M. de Rijke, 2003. Natural language parsing with W-grammars. Paper presented at *CLIN'03*.

Infante-Lopez, G. and M. de Rijke, 2004a. Alternative approaches for generating bodies of grammar rules. In *Proceedings of the 42nd Annual Meeting of the ACL*. Barcelona.

Infante-Lopez, G. and M. de Rijke, 2004b. Comparing the ambiguity reduction abilities of probabilistic context-free grammars. In *Proceedings of LREC'04*.

Infante-Lopez, G. and M. de Rijke, 2004c. Expressive power and consistency properties of state-of-the-art natural language parsers. In J. Vicedo, P. Martinez-Barco, and R. M. et al. (eds.), *Proceedings Advances in Natural Language Processing: 4th International Conference, EsTAL 2004*.

Infante-Lopez, G., H. Hermanns, and J.-P. Katoen, 2001. Beyond memoryless distributions: Model checking semi-markov chains. In *PAPM-ProbMiV 2001, Springer LNCS 2165.*.

Infante-Lopez, G., K. Sim'aan, and M. de Rijke, 2002. A general model for dependency parsing. In *Proceedings of BNAIC'02*.

Inui, K., V. Sornlertlamvanich, H. Tanaka, and T. Tokunaga, 1998. Probabilistic GLR parsing: a new formatisation and its impact on parsing performance. *Journal of Natural Language Processing*, 5(3).

Jelinek, F., J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos, 1994. Decision tree parsing using a hidden derivation model. In *Proceedings of the 1994 Human Language Technology Workshop*. DARPA.

Joan-Andreu, S. and J.-M. Benedí, 1997. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055.

Johnson, M., 2002. The DOP estimation method is biased and inconsistent. *Computational Linguistics*, 28(1):71–76.

Johnson, M., S. Geman, S. Canon, Z. Chi, and S. Riezler, 1999. Estimators for stochastic unification-based grammars. In *Proceedings of the 37th Annual Meeting of the ACL*. Maryland.

Joshi, A., 1985. Tree Adjoining Grammars: How much context sensitivity is required to provide a reasonable structural description. In I. K. D. Dowty and A. Zwicky (eds.), *Natural Language Parsing*. Cambridge, U.K.: Cambridge University Press.

Joshi, A., 1987. An introduction to tree adjoining grammars. In A. Manaster Ramer (ed.), *Mathematics and Language*. Amsterdam: John Benjamins Publishing Co, pages 87–115.

Joshi, A. and B. Srinivas, 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING)*. Kyoto, Japan.

Jurafsky, D. and J. Martin, 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR.

Klein, D. and C. Manning, 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the ACL*.

Krotov, A., M. Hepple, R. J. Gaizauskas, and Y. Wilks, 1998. Compacting the Penn treebank grammar. In *Proceedings of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics COLING*.

Kruijff, G., 2003. 3-Phase grammar learning. In *Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Development*.

Kucera, H. and W. N. Francis, 1967. *Computational analysis of present-day American English*. RI: Brown University Press.

Lafferty, J., D. Sleator, and D. Temperley, 1992. Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Conference on Probabilistic Approaches to Natural Language Processing*.

Levin, B., 1993. *English Verb Classes and Alternations: A preliminary Investigation*. Chicago: The University of Chicago Press.

Lin, D., 1995. A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of IJCAI-95*.

Magerman, D., 1995a. *Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University.

Magerman, D., 1995b. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the ACL*. Cambridge, MA.

Magerman, D. and M. Marcus, 1991. Pearl: A Probabilistic Chart Parser. In *Proceedings the European Chapter of the ACL*. Berlin.

Magerman, D. and C. Weir, 1992. Efficiency, robustness and accuracy in picky Chart Parsing. In *Proceedings of the 30th Annual Meeting of the ACL*. Newark, Delaware.

Manning, C., 1993. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the ACL*.

Manning, C. and H. Schütze, 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA.

Marcus, M., G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, 1994. The Penn treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*.

Marcus, M., B. Santorini, and M. Marcinkiewicz, 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

Mateescu, A., 1989a. Van Wijngaarden grammars and systems. *Annals University of Bucharest*, 2:75–81.

Mateescu, A., 1989b. van Wijngaarden grammars and the generative complexity of recursively enumerable languages. *Annals University of Bucharest*, 2:49–54.

Mateescu, A. and A. Salomaa, 1997. Aspects of classical language theory. In Rozemberg and Salomaa (1997), pages 175–251.

Merlo, P. and S. Stevenson, 2001. Automatic verb classification based on statistical distributions of argument structure. *Computational Linguistics*, 27(3):373–408.

Miikkulainen, R., 1996. Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, 1(20).

Mitchell, T., 1997. *Machine Learning*. McGraw-Hill Series in Computer Science.

Musillo, G. and K. Sima'an, 2002. Towards comparing parsers from different linguistic frameworks: An information theoretic approach. In *Proceedings of Beyond PARSEVAL: Towards Improved Evaluation Measures for Parsing Systems, LREC'02*. Las Palmas, Gran Canaria, Spain, 2002.

Nederhof, M.-J., A. Sarkar, and G. Satta, 1998. Prefix probabilities from probabilistic tree adjoining grammars. In *Proceedings of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING)*. Montreal.

Nederhof, M.-J. and G. Satta, 2002. Probabilistic parsing strategies. In J. Dassow, M. Hoeberechts, H. Jürgensen, and D. Wotschke (eds.), *Descriptional Complexity of Formal Systems (DCFS), Pre-Proceedings of a Workshop*. London.

Ney, H., 1991. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2):336–340.

Ney, H. and R. Kneser, 1993. Inproved clustering techniques for class-based statistical language modelling. In *European Conference on Speech Communication and Technology*.

Ng, S.-K. and M. Tomita, 1991. Probabilistic LR parsing for general contextfree grammars. In *Proceedings of the 2nd International Workshop on Parsing Technologies*.

NIST, 2004. *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST. URL: `http://www.itl.nist.gov/div898/handbook/`.

Oncina, J. and P. Garcia, 1992. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis, volume 1 of Series in Machine Perception and Artificial Intelligence*. pages 49–61.

Parikh, R. J., 1966. On context-free languages. *Journal of the ACM*, 13:570–581.

Perrault, R., 1984. On the mathematical properties of linguistic theories. *Computational Linguistics - Special issue on mathematical properties of grammatical formalisms*, 10(3-4):165–176.

Punyakanok, V. and D. Roth, 2000. The use of classifiers in sequential inference. In *Proceedings of NIPS-13, The 2000 Conference on Advances in Neural Information Processing Systems*.

Qumsieh, A., 2003. Ai::genetic - a pure perl genetic algorithm implementation. Perl Package, http://search.cpan.org/ aqumsieh/AI-Genetic-0.02/Genetic.pm.

Ratnaparkhi, A., 1997. A linear observed time statistical parser based on maximum entropy models. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Ratnaparkhi, A., 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Ratnaparkhi, A., 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

Rayner, M. and D. Cater, 1996. Fast parsing using pruning and grammar specialization. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics. Santa Cruz*.

Resnik, P., 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*. Nantes.

Rosenfeld, R., 1994. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. Ph.D. thesis, University of Carnegie Mellon.

Rozemberg, G. and A. Salomaa (eds.), 1997. *Handbook of Formal Languages*.

Samuelsson, C., 1994. Grammar specialization through entropy thresholds. In *Proceedings of the 32nd Annual Meeting of the ACL.*

Sarkar, A., 2001. Applying co-training methods to statistical parsing. In *Proceedings of the 2nd Meeting of the North American Chapter of the ACL (NAACL 2001).* Pittsburgh.

Satta, G., 1998. Recognition and parsing for tree adjoining grammars. In *Tutorial presented at the 4th International Workshop on Tree Adjoining Grammars (TAG+4).* Pennsylvania.

Satta, G., 2000. Parsing techniques for lexicalized context-free grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT).* Trento, Italy.

Schabes, Y., 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING).* Nantes.

Schapire, R. E. and Y. Singer, 1999. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3).

Sima'an, K., 2000. Tree-gram parsing: Lexical dependencies and structual relations. In *Proceedings of the 38th Annual Meeting of the ACL.* Hong Kong, China.

Sima'an, K. and L. Buratto, 2003. Backoff parameter estimation for the dop model. In *Proceedings of ECML.*

Simmons, R. and Y. Yu, 1992. The acquisition and use of context-dependent grammars for English. *Computational Linguistics*, 4(18).

Sleator, D. and D. Temperley, 1991. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, School of Computer Science.

Sleator, D. and D. Temperley, 1993. Parsing English with a link grammar. In *Proceedings of the 3rd International Workshop on Parsing Technologies (IWPT).* Bergen, Norway.

Srinivas, B., 1997. *Complexity of Lexical Descriptions and its Relevance to Partial Parsing.* PhD thesis, Computer and Information Science, University of Pennsylvania.

Stevenson, S. and P. Merlo, 2000. Automatic lexical acquisition based on statistical distributions. In *17th conference on Computational linguistics.*

Taylor, H. and S. Karlin, 1998. *An Introduction to Stochastic Modeling*. Academic Press.

Thollard, F., P. Dupont, and C. de la Higuera, 2000. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proceedings of the 7th International Conference on Machine Learning (ICML)*. Stanford.

Tjong Kim Sang, E. and H. Déjean, 2001. Introduction to the CoNLL-2001 shared task: clause identification. In *Proceedings of the Computational Natural Language Learning Workshop (CoNLL-2001)*. Toulouse.

Tomita, M., 1996. *Efficient Parsing for Natural Language*. The Netherlands: Kluwer Academic Publishers.

Van Wijngaarden, A., 1965. Orthogonal design and description of a formal language. Technical Report MR76, Mathematisch Centrum. Amsterdam.

Van Wijngaarden, A., 1969. Report on the algorithmic language ALGOL 68. *Numerische Mathematik*, 14:79–218.

Veenstra, J. and W. Daelemans, 2000. A memory-based alternative for connectionist shift-reduce parsing. Technical Report ILK Report 00-12.

Venable, P., 2001. Lynx: Building a statistical parser from a rule-based parser. In *Proceedings of the Student Research Workshop of the NAACL. Pittsburgh.*

Watkinson, S. and S. Manandhar, 2001. Translating treebank annotation for evaluation. In *Workshop on Evaluation for Language and Dialogue Systems, ACL/EACL.*

Wetherell, C. S., 1980. Probabilistic languages: A review and some questions. *ACM Computer Surveys.*, 4(12):361–379.

Wich, K., 2000. Exponential ambiguity of context-free grammars. In *Proceedings of the 4th International Conference on Developments in Language Theory.*

Wich, K., 2001. Characterization of context-free languages with polynomially bounded ambiguity. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS).*

Wood, M., 1993. *Categorial Grammars*. London: Routledge.

Wright, D., 1997. *Understanding Statistics. An Introduction for the Social Sciences*. London: Thousand Oaks, New Delhi: Sage.

Wright, J., 1990. LR parsing of probabilistic grammars with input uncertainty for speech recognition. *Computer Speech and Language*, 4:297–323.

Wright, J. and E. Wrigley, 1989. Probabilistic LR parsing for speech recognition. In *Proceedings of the 1st International Workshop on Parsing Technologies*. Pittsburgh.

Wright, J., E. Wrigley, and R. Sharman, 1991. Adaptive probabilistic generalized LR parsing. In *Proceedings of the 2nd International Workshop on Parsing Technologies*. Cancun, Mexico.

Xia, F., C. Han, M. Palmer, and A. Joshi, 2001. Automatically extracting and comparing lexicalized grammars for different languages. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*. Seattle.

Yoshinaga, N., 2004. Improving the accuracy of subcategorizations acquired from corpora. In *Student Session ACL04*.

Zelle, J. and R. Mooney, 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the 13th National Conference on Artificial Intelligence*. Portland, OR.

Abstract

Natural language is a very complex phenomenon. Undoubtedly, the sentences we utter are organized according to a set of rules or constraints. In order to communicate with others, we have to stick to these rules up to a certain degree. This set of rules, which is language dependent, is well-known to all speakers of a given language, and it is this common knowledge that makes communication possible. Every sentence has a clear organization: words in an utterance glue together to describe complex objects and actions. This hidden structure, called syntactic structure, is to be recovered by a parser. A *parser* is a program that takes a sentence as input and tries to find its syntactic organization. A parser searches for the right structure among a set of possible analyses, which are defined by a *grammar*. The language model decides what the syntactic components of the sentence are and how they are related to each other, depending on the required level of detail.

Designing and building language models is not a trivial task; the design cycle usually comprises designing a model of syntax, understanding its underlying mathematical theory, defining its probability distribution, and finally, implementing the parsing algorithm. The building of a new language model has to complete at least these steps, and each of them is very complex and a line of research in itself. To help handling the intrinsic complexity of this cycle a good level of *abstraction* is required.

Our view is that state-of-the-art natural language models lack abstraction; their design is often *ad hoc*, and they mix many features that, at least conceptually, should be kept separated. In this thesis, we explore new levels of abstraction for natural language models. We survey state-of-the-art probabilistic language models looking for characteristic features, and we abstract away from these features to produce a general language model. We investigate three state-of-the-art language models and discover that they have one very noticeable feature: the set of rules they use for building trees is built on the fly, meaning that the set of rules is not defined *a priori*. The formalisms we

review have two different levels of derivations even though this is not explicitly stated. One level is for generating the set of rules to be used in the second step, and the second step is for building the set of trees that characterize a given sentence. Our formalism, based on *Van Wijngaarden grammars* (W-grammars), makes these two levels explicit. Our approach to parsing comes from a formal language perspective: we identify features that are used by state-of-the-art language models and take a formalism off the shelf and modify it to incorporate the necessary features.

From a theoretical point of view, general models help us to clarify the set of parameters a particular instance has fixed, and to make explicit assumptions that underlie a particular instance. When analyzing the necessary features from the formal language perspective, the need for probabilities and their role in parsing are the first issue to address. We answer many questions regarding the role of probabilities in probabilistic context free grammars. We focus on these grammars because they are central to the formalism we present.

From a computational point of view, general models for which a clear parsing algorithm and a relatively fast implementation can be defined, produce fast and clear implementations for all particular instances.

General models do not add anything *per se*. Their importance is rather in the set of instances they can capture and the new directions they are able to suggest. We show that bilexical grammars, Markovian context free grammars and stochastic tree substitution grammars are instances of our general model. Our model has well-established consistency properties which we use to derive consistency properties of these three formalisms. The new research directions suggested by a general formalism are a consequence of instantiating the model's parameters in different ways or by re-thinking the set of assumptions the particular instances have made. A brief description of the directions explored in this thesis follows.

Markov models are heavily used in parsing models and they can be replaced by probabilistic regular languages. Since our formalism is not bound to Markov models, we can use any algorithm for inducing probabilistic automata. We explore this idea. We define a type of grammar that uses probabilistic automata for building the set of rules. We compare two classes of grammars that differ in the type of algorithm they use for learning the probabilistic automata. One of them is based on n-grams, and the other one is based on the minimum divergence algorithm (MDI). We show that the MDI algorithm produces both smaller and better performing grammars.

The fact that probabilistic automata replace Markov chains in the definition of our model allows us to think of a regular language as the union of smaller, more specific sublanguages. Our intuition is that the sublanguages are easier to induce and that the combination of them fully determines the whole language. We explore this idea by splitting the training material before inducing the probabilistic automata, then inducing

one automaton for each component, and, finally, combining them into one grammar. We show that in this way, a measure that correlates well with parsing performance can be defined over grammars.

Our formalism allows us to isolate particular aspects of parsing. For example, the linear order in which arguments appear in a parse tree is a fundamental feature used by language models. We investigate which sequences of information better predict sequences of dependents. We compare sequences of part-of-speech tags to sequences of non-terminal labels. We show that part-of-speech tags are better predictors of dependents.

Samenvatting

Natuurlijke taal is een erg complex fenomeen. Het lijdt geen twijfel dat de zinnen die we uiten georganiseerd zijn volgens een verzameling regels of randvoorwaarden. Om met anderen te kunnen communiceren dienen we ons tot op zekere hoogte aan deze regels te houden. Deze verzameling regels, die taalafhankelijk is, is bekend bij alle sprekers van een gegeven taal, en het is deze gedeelde kennis die communicatie mogelijk maakt. Iedere zin heeft een duidelijke organisatie: woorden in een uiting kunnen samen gebruikt worden om complexe objecten en acties te beschrijven. Het is deze verborgen structuur, de syntactische structuur, die door een parser ontdekt moet worden. Een *parser* is een programma dat een zin als invoer neemt en probeert de syntactische organisatie van die zin te vinden. Een parser zoekt naar de juiste structuur temidden van een verzameling van mogelijke analyses, die gedefinieerd wordt door een *grammatica*. Het taalmodel besluit wat de syntactische componenten van de zin zijn, en hoe ze met elkaar verbonden zijn, afhankelijk van de gewenste mate van detail.

Het ontwerpen en bouwen van taalmodellen is verre van triviaal; de ontwerpcyclus bestaat doorgaans uit het ontwerpen van een model van de syntax, het begrijpen van de onderliggende wiskundige theorie, het definiëren van een waarschijnlijkheidsverdeling, en tenslotte, het implementeren van een parseeralgoritme. Het bouwen van een nieuw taalmodel dient elk van deze stappen te doorlopen, en elk van deze stappen is complex, en een onderzoeksveld op zich. Om de intrinsieke complexiteit van deze cyclus te beheersen is een zeker niveau van *abstractie* vereist.

Ons standpunt is dat het moderne natuurlijke taalmodellen aan abstractie ontbreekt. Hun ontwerp is vaak *ad hoc*, en ze vermengen aspecten die uit elkaar gehouden zouden moeten worden, in ieder geval op een conceptueel niveau. In dit proefschrift onderzoeken we nieuwe niveaus van abstractie voor natuurlijke taalmodellen. Op zoek naar karakteristieke aspecten geven we een overzicht van probabilistische taalmodellen, en we abstraheren van deze karakteristieke aspecten weg om tot een algemeen taalmodel

te komen. We onderzoeken drie moderne taalmodellen en ontdekken dat ze één opmerkelijk aspect delen: de regels die zij gebruiken voor het bouwen van bomen worden *on the fly* gecreeërd. Met andere woorden, deze regels staan niet *a priori* vast. De formalismen die wij bestuderen hebben twee niveaus van afleidingen, hoewel dit niet expliciet vermeld wordt. Eén niveau betreft het genereren van de verzameling van regels die in de tweede stap gebruikt zullen worden, en de tweede stap betreft het bouwen van de bomen die een gegeven zin karakteriseren. Ons formalisme, dat gebaseerd is op *Van Wijngaardengrammatica's* (W-grammatica's), maakt deze twee niveaus expliciet. Ons perspectief op parseren vindt zijn oorsprong in de formele talen: we identificeren aspecten die door moderne taalmodellen gebruikt worden en nemen een bestaand formalisme dat we zó aanpassen, dat het de vereiste aspecten in zich opneemt.

Vanuit een theoretische gezichtspunt helpen algemene taalmodellen ons om duidelijk te maken welke parameters een specifieke instantie heeft ingevuld en vastgelegd, en om expliciet te maken welke aannames een specifieke instantie heeft gemaakt. Wanneer we de voor parseren vereiste aspecten vanuit een formele talenperspectief willen bestuderen, dan is duidelijk dat we voor alles waarschijnlijkheden en hun rol in het parseren moeten bestuderen. In het proefschrift beantwoorden we een groot aantal vragen betreffende de rol van waarschijnlijkheden in probabilistische context-vrije grammatica's, die een centrale rol spelen in de formalismen die we bestuderen.

Vanuit een computationeel gezichtspunt kunnen algemene modellen waarvoor een duidelijk parseeralgoritme en redelijke snelle implementaties gedefinieerd kunnen worden, leiden tot snelle en doorzichtige implementaties voor alle specifieke instanties.

Op zich voegen algemene modellen niets toe. Hun belang ligt veeleer in de verzameling instanties die ze kunnen beschrijven en de nieuwe onderzoeksrichtingen die ze kunnen suggereren. We laten zien dat bilexicale grammatica's, Markoviaanse context-vrije grammatica's, en stochastische boomstubstitutiegrammatica's instanties zijn van ons algemene model. Ons model heeft duidelijke en goed begrepen consistentie-eigenschappen die we gebruiken om consistentie-eigenschappen af te leiden voor de genoemde drie formalismen. De nieuwe onderzoeksrichtingen die ons algemene formalisme suggereert zijn een gevolg van het instantiëren van de parameters van het model of van herbezinning op de aannames die een specifieke instantie doet. We geven nu een korte beschrijving van de richtingen die we in het proefschrift onderzoeken.

Markov-modellen worden veelvuldig gebruikt in parseermodellen, en ze kunnen vervangen worden door, meer algemene, probabilistische reguliere talen. Omdat ons formalisme niet beperkt is tot Markov-modellen, kunnen we om het even welk algoritme gebruiken voor het induceren van probabilistische automaten. We onderzoeken dit idee. We definiëren een soort van grammatica's die probabilistische automaten gebruiken voor het genereren van regels, en vergelijken twee klassen van grammatica's, die verschillen in het soort van algoritme dat ze gebruiken voor het leren van proba-

bilistische automaten. De ene klasse is gebaseerd op n-grammen, en de ander op het minimale divergentie algoritme (MDI). We laten zien dat het MDI-algoritme leidt tot kleinere grammatica's die zich beter gedragen.

Het feit dat probabilistische automaten Markov-ketens vervangen in de definitie van ons model stelt ons in staat een reguliere taal te zien als de vereniging van kleinere, meer specifieke deel-talen. Een belangrijke intuïtie is dat de deel-talen eenvoudiger geïnduceerd kunnen worden dan de gehele taal (in één keer), en dat hun combinatie de gehele taal volledig vastlegt. We onderzoeken dit idee door het trainingsmateriaal te splitsen voordat we de probabilistische automaten induceren. Vervolgens induceren we één automaat per componenent, en tot slot combineren we deze tot één grammatica. We laten zien dat we op deze manier een maat op grammatica's kunnen definiëren die goed correleert met het uiteindelijke parseergedrag.

Ons algemene formalisme stelt ons in staat om specifieke aspecten van het parseren te isoleren en bestuderen. Om een voorbeeld te noemen, de lineaire orde waarin de argumenten in een parseerboom verschijnen is een wezenlijk aspect dat door taalmodellen gebruikt wordt. We onderzoeken welke reeksen van informatie bettere voorspellingen geven over reeksen van afhankelijken in een parseerboom. We vergelijken reeksen van labels van woordklassen met reeksen van niet-terminale labels, en we laten zien dat wooordklasse-labels betere voorspellers zijn.