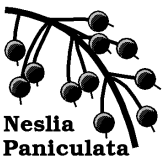
TOWARDS PROBABILISTIC UNIFICATION-BASED PARSING

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Ter Doest, Hugo

Towards probabilistic unification-based parsing /
Hugo ter Doest - Enschede: Neslia Paniculata. -I11.
Thesis Universiteit Twente Enschede. - With ref.
With summary in Dutch.
ISBN 90-75296-04-5
Subject headings: parsing

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).



Neslia Paniculata
Uitgeverij, Enschede

TOWARDS PROBABILISTIC UNIFICATION-BASED PARSING

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 12 februari 1999 te 15.00 uur.

door

Hugo Wilfried Laurenz ter Doest

geboren op 17 november 1969
te Hengelo

Dit proefschrift is goedgekeurd door de promotor,

prof.dr.ir. A. Nijholt,

en door de assistent-promotor,

dr.ir. H.J.A. op den Akker.

Acknowledgements

This is the right time and the right place to look back, and to thank the people that have supported me. I would never have finished this thesis without their help, advise, and attention.

First of all I would like to thank my advisor Rieks op den Akker. He coached me these four years, he believed in me, and he motivated me to start writing for my thesis. I am also grateful to the other members of my *begeleidingscommissie*, Anton Nijholt, Franciska de Jong, and Gerrit van der Hoeven, for their advise, and their interest in my ideas.

I would like to thank Anton Nijholt for giving me the opportunity to work as an *Assistent in Opleiding* at the Parlevink research group, and for providing the freedom for working out my ideas.

I would like to thank Rieks op den Akker, Franciska de Jong, Anton Nijholt, Klaas Sikkel, and prof. Schaafsma for their useful comments on an earlier version of this thesis. It has greatly benefited from their advise and insights.

Thanks to my roommates Jos Buis and Hendri Hondorp for sharing lots of coffee, UNIX, C, and all those other important things in life. I thank Hendri Hondorp for TEXnical support and for enduring my moods and restless behaviour in the past year.

I thank Eric Schol for all tea breaks, and for *CDs plakken* in the Dixo (Stichting CD-Uitleen Drienerlo).

Thanks to everyone at the Dixo. In particular, I would like to thank Bart Lucassen, André Reimerink, and Monique Engelbertink of the *woensdag-avond-1* team.

```
\selectlanguage{dutch}
```

Ik bedank Laura en Viola voor hun enthousiasme en humor. Tenslotte bedank ik Ingrid voor onze interessante gesprekken over kennisrepresentatie en natuurlijke taalverwerking, voor haar verdraagzaamheid en geduld, en voor alles wat ze voor me heeft gedaan in het afgelopen jaar.

Contents

Acknowledgements v

1 Introduction 1
1.1 Focus . . . 1
1.2 Setting of the research . . . 2
1.3 The SCHISMA system . . . 3
1.3.1 An architecture . . . 4
1.3.2 Processing modules . . . 6
1.3.3 Storage modules . . . 7
1.3.4 Interfaces between the modules . . . 9
1.3.5 Keyboard input versus spoken input . . . 9
1.4 Contribution . . . 10
1.5 Organisation of the thesis . . . 11

I Basics 13

2 Probabilistic Grammars, a Survey 15
2.1 Context-free grammars . . . 15
2.2 Probabilistic context-free grammars . . . 17
2.2.1 Supervised training . . . 17
2.2.2 Unsupervised training: the Inside-Outside algorithm . . . 18
2.2.3 Discussion . . . 19
2.3 Extensions of probabilistic context-free grammars . . . 20
2.3.1 Weakly restricted stochastic grammars . . . 20
2.3.2 History-based grammars . . . 21
2.3.3 Salomaa's probabilistic grammars . . . 22
2.4 Statistics of parser actions . . . 22
2.4.1 Probabilistic GLR parsing . . . 22
2.4.2 Probabilistic left-corner parsing . . . 23
2.5 Tree-based formalisms . . . 23
2.5.1 Tree adjoining grammar . . . 24
2.5.2 Data-oriented parsing . . . 25
2.6 Unification grammars . . . 27
2.6.1 Probabilistic ALE . . . 27

2.6.2 Probabilistic LR parsing with unification grammars . . . 28
2.6.3 Probabilistic feature grammars . . . 29
2.6.4 Stochastic attribute-value grammars . . . 30
2.7 Extensions of Constraint Logic . . . 30
2.7.1 Probabilistic CUF . . . 30
2.7.2 Weigthed constraint logic . . . 31
2.7.3 Probabilistic constraint logic . . . 32
2.8 Summary . . . 33

3 Unification-based Parsing **35**
3.1 Feature structures . . . 36
3.2 Subsumption and unification . . . 38
3.3 Multi-rooted feature structures . . . 41
3.4 Unification grammars . . . 43
3.5 Parsing . . . 44
3.5.1 Chart parsing . . . 45
3.5.2 Left-corner parsing . . . 46
3.5.3 Head-corner parsing . . . 49
3.6 Probabilistic extensions . . . 50
3.7 Summary . . . 51

4 Maximum Entropy Modelling **53**
4.1 Some probability theory . . . 54
4.2 Entropy measures . . . 55
4.3 The maximum entropy method . . . 59
4.4 Parameter estimation . . . 63
4.4.1 Generalized Iterative Scaling . . . 63
4.4.2 Improved Iterative Scaling . . . 65
4.5 Monte Carlo Sampling . . . 68
4.6 Predictive maximum entropy models . . . 69
4.7 Summary . . . 69

II Application **71**

5 The Grammar Inference Engine **73**
5.1 A short introduction to SGML . . . 73
5.2 Annotation scheme . . . 76
5.2.1 Tagset . . . 76
5.2.2 Syntactic relations . . . 77
5.2.3 Other attributes . . . 79
5.3 Annotation scheme comparison . . . 82
5.4 Grammar inference . . . 83
5.5 Unification constraints . . . 85
5.5.1 Lexical constraints . . . 86
5.5.2 RHS nonterminal constraints . . . 86

5.5.3 Identifier attribute constraints . . . 87
5.5.4 Referring attribute constraints . . . 87
5.5.5 General constraints . . . 89
5.6 Facts and figures . . . 90
5.7 Summary . . . 91

6 Experimental Results **93**
6.1 The parsing system . . . 93
6.1.1 Unknown words and parsing failure . . . 94
6.1.2 Probabilistic parsing . . . 94
6.2 Parameter estimation . . . 96
6.3 Parser evaluation . . . 96
6.4 Experiment 1 . . . 98
6.5 Experiment 2 . . . 98
6.6 Experiment 3 . . . 99
6.7 Discussion . . . 101
6.8 Summary . . . 101

III Epilogue **103**

7 Conclusions **105**
7.1 Towards integration in a SCHISMA prototype . . . 105
7.2 Recommendations for future research . . . 106

IV Appendices **107**

A SCHISMA Treebank DTD **109**

B Meta-constraints **115**

C Samenvatting **119**

Bibliography **121**

Index **131**

Abbreviations **133**

Chapter 1
Introduction

This thesis is about natural language parsing with corpus-based grammars that are enriched with statistics. Parsing is the process of analysing the syntactic structure of an utterance. The role of statistics is to improve the parsing process. Our research is done in the context of the SCHISMA project, in which a natural language dialogue system for theatre information and booking services is developed. In the next section we further develop the subject of our research. Section 1.2 describes the SCHISMA project which provided the setting of the research, and in section 1.3 we propose an architecture for the SCHISMA system. In section 1.4 we state the contribution of this thesis to language technology, and section 1.5 gives an overview of the thesis.

1.1 Focus

Our research was motivated by the need for an efficient and effective syntactic parser for the SCHISMA system. In a practical environment like a dialogue system, parsing is a difficult task, mostly because of the enormous freedom the user has to express himself. Even dialogue strategies that are carefully chosen to influence the user in its behaviour, cannot prevent the use of syntactic structure that is not covered by the grammar. This calls for a back-up strategy which can process any sentence that is outside the language of the grammar. Ideally, such a strategy takes into account the results of earlier at analysing the sentence, i.e. it should do an educated guess at the right structure of the sentence using partial results.

It is our conviction that automatic corpus-based development of grammars is a good way to guarantee coverage and to keep expertise requirements to a minimum. "Conventional parsers utilising hand-crafted generative grammars and knowledge bases require considerable linguistic expertise and knowledge engineering effort to produce, and suffer from problems of under-generation, brittle behaviour and domain dependence." (Keller 1998). Also, corpus-based grammar engineering turns maintenance of the language model into a matter of keeping the corpus (or corpora) and its annotation in line with the domain and the system requirements. Moreover, corpus-based grammar development can be integrated in an iterative development cycle of a dialogue system very

easily: fully functional prototypes of a dialogue system can be used to collect additional data that can be used to improve the language model further.

Extending a grammar formalism with statistics does not imply that the formalism is inadequate for processing natural language. On the contrary, in my view it confirms the adequacy of the formalism for modelling natural language. The reason for combining it with statistics is that we recognise that we simply cannot model every aspect of natural language. It is impossible to collect and represent all information needed to process language deterministically. Like Abney we believe that "A probabilistic model is only a stopgap in absence of an account of the missing factors: semantics, pragmatics, what topics I've been talking to other people about lately, how tired I am, whether I ate breakfast this morning" (Abney 1996). Such a lack of knowledge often results in *overgeneration*, strictly speaking the generation of more than one parse tree. Associating weights or probabilities with parse trees that represent preference or probability of being correct, helps to choose the best analysis (or n-best analyses).

Research on probabilistic grammars and parsing is very much in line with the classical model of probabilistic context-free grammars (PCFGs). Unfortunately, the classical model is not very flexible for extension to more context-sensitive probabilities, and the incorporation of knowledge other than rule frequencies is cumbersome. We advocate maximum entropy modelling as a framework for probabilistic extensions of grammar formalisms. Maximum entropy modelling does not put any restrictions on the underlying formalisms, and the parameters of the probabilistic model are not attached to the rules of the grammar. The probability of a derivation tree simply does not have to depend on how the tree was created.

The probabilistic extension of a grammar formalism serves to find the most probable parse tree, and can be applied to pre-empt the analyses that will receive low probabilities. The latter application is of computational interest only, it may result in less memory consumption.

1.2 Setting of the research

SCHISMA is a research project of the Parlevink group[1] aimed at the development of a natural language dialogue system for theatre information and booking services. (van der Hoeven et al. 1994) is an early overview publication on the project; it gives a clear account of the motivation for the project and defines the goals of the project: "The aim of the project is to develop a prototype of a natural language dialogue system. The envisaged system is capable of providing a user with information about theatre performances, and it should allow the user to book seats for such performances. In addition to the goal of building a prototype of some quality there is the equally important goal of gaining deeper insight in the problems one encounters in the process of building a natural language

[1] Parlevink is a language theory and technology project of the Department of Computer Science at the University of Twente. The main interests in research are linguistic engineering, neuro-engineering and the use of formal methods in these areas.

dialogue system. Getting experienced is considered one of the prerequisites for a successful follow-up of the project." An environment was developed to enable the collection of a corpus of dialogues in Wizard of Oz experiments. All this with an emphasis on keyboard input: "The emphasis is on keyboard input and if possible we would like to add the possibility to access SCHISMA using speech input." (van der Hoeven et al. 1994).

Since 1994 a continuing stream of publications documents the SCHISMA project. Komen (1995) evaluates Natural Language^{TM} for application in the SCHISMA domain. Natural Language is a tool for building natural language query interfaces for databases. In (op den Akker et al. 1995) parsing with unification grammars in the SCHISMA system is investigated. (ter Doest et al. 1996) investigates, from a language engineering point of view, the integration in the SCHISMA system of the unification-based head-corner parser developed by Moll (1995).

Papers that investigate semantical issues are (Hulstijn et al. 1996), which applies topic-focus ideas to the SCHISMA domain, and (Hulstijn 1997), which defines Update Semantics with Questions (USQ), a combination of update semantics and partition semantics of questions. Andernach and van Steenbergen (1994) investigate the several types of knowledge involved in natural language dialogue systems. They consider representation languages for modelling knowledge and give recommendations for the SCHISMA project. Andernach (1996) presents a machine learning approach to the classification and prediction of speech acts.

In (Lie et al. 1998) a SCHISMA prototype called Theatre Information System (THIS) is described. The natural language understanding in THIS is based on a string rewriting technique. User utterances are rewritten into a normal form through the application of context-sensitive string-to-string transformations. The resulting normal form is then interpreted in the context of the dialogue. Recently, this prototype has been integrated into a virtual reality model of the Muziekcentrum, a theatre for music and play in Enschede, specified in the Virtual Reality Modelling Language (VRML). The virtual Muziekcentrum serves as an environment for the experimentation with multi-modal interaction in task domains (Nijholt et al. 1998). In the near future the environment will be extended with a speech-driven navigation functionality.

1.3 The SCHISMA system

SCHISMA is a natural language interface to a database, which "...is a system that allows the user to access information stored in a database by typing requests expressed in some natural language (e.g. English)" (Androutsopoulos et al. 1995). We consider dialogue systems that allow the user to access the database by engaging in a *dialogue* with the system. This implies that the system should have an awareness of (the structure of) dialogues, and the capability of interpreting the user's input in the context of the dialogue. The dialogue systems we consider have task domains that are considerably restricted by the

information contained in the database; formulated somewhat more precisely, we consider dialogue systems that allow the user to perform queries and updates on the database through natural language dialogue.

Obviously, the system should have *linguistic knowledge*, i.e. knowledge about language in general and about the language in which dialogues take place in particular. In our definition, linguistic knowledge describes the morphology and syntax of a language. It accounts for the form of the constituents an utterance consists of, and the function they have within that utterance.

The database has a strong influence on the domain in which dialogues take place. The system should have knowledge about the concepts related to the contents and the structure of the database, and knowledge about the purpose of such concepts. This type of knowledge is called *domain knowledge*; also it needs some amount of *world knowledge*, i.e. knowledge of concepts and actions that are not domain-specific, but necessary to understand natural language. Not only should the system know things, it should know what it does not know as well. If the user talks about things outside the domain, the system should act intelligently. The boundary between world knowledge and domain knowledge is difficult to define. The same holds for world/domain knowledge and linguistic knowledge.

Information dialogues in restricted domains differ from general dialogue in that they are directed at communicating and acquiring information relevant to the domain. The structure of such dialogues reflects this directedness, and dialogue systems can exploit it by making assumptions on the user's behaviour. The system assumes that the user has an information need which fits the information it can provide. This is a strong assumption, but a relief at the same time. The system's behaviour can be based on assumptions about how users act in information dialogues, and expectations about the user's answers and wishes may simplify the understanding task considerably. The maxim's of Grice are often cited in this respect (Grice 1975). Ultimately, the formulation of restrictions and assumptions gives perspective on a declarative specification of (classes of) dialogue systems. A commercial product like Speech Mania^{TM} in which the structure of dialogue can be defined in a specification language shows the potential of a restricted view of information dialogue.

The purpose of the rest of this section is to place the parsing task in the context of the SCHISMA system, and to make clear how a software module that performs this task communicates with the rest of the system.

1.3.1 An architecture

We advocate a modular architecture for the SCHISMA system as given in figure 1.1. We have represented the processing modules by square boxes and the storage modules by oval boxes. Processing modules are software programs that perform a task, and storage modules are locations for storing something.

A modular architecture has several advantages over an integrated one. The most obvious ones are maintainability and portability to new domains. Another consideration is that modularity requires the development of a dialogue

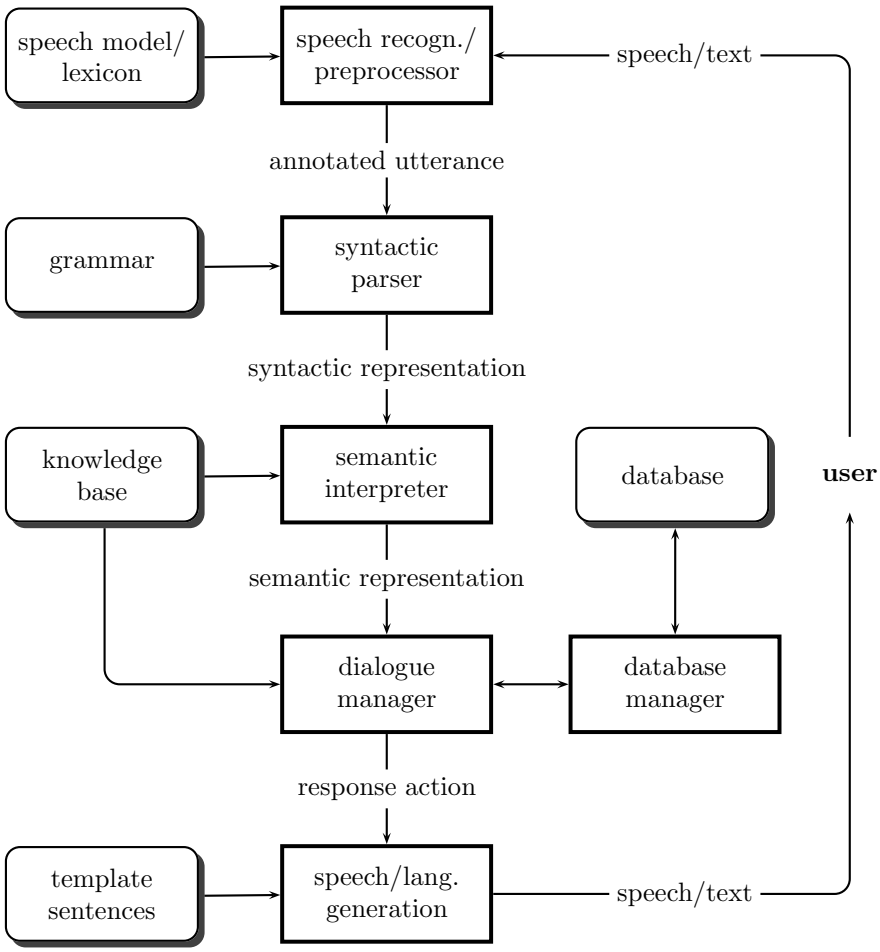


Figure 1.1: An architecture for SCHISMA.

system to be divided in several subproblems, which is, taking into account the complexity of such an enterprise, very welcome.

Our architecture of the SCHISMA system implies communication in one direction only, from pre-processor to response generator. This is a severe architectural restriction. It means that modules cannot dispose of information extracted by 'future' modules from the current utterance. Modules either have to hypothesise alternative interpretations, or force a decision on one interpretation of their input. Disambiguation by dialogue history will be possible in the dialogue manager only. The advantages of this *pipe-line* architecture are mainly of engineering interest. For instance, the interfaces between the mod-

ules can be kept simple. It is straightforward to define the subproblems (and therefore projects) involved in building a dialogue system. The final system can be maintained much better. Different implementations of the same module can be exchanged fairly simple, which is useful for testing and evaluation purposes.

In section 1.3.2 we discuss the processing modules, and in section 1.3.3 the storage modules. Section 1.3.4 considers the communication between the modules of the system. In section 1.3.5 we restrict our research to a keyboard-driven SCHISMA system, and we discuss the relevance of our research for spoken dialogue systems

1.3.2 Processing modules

Below we discuss the functionality of the processing modules of the SCHISMA system.

speech recogniser In the case of spoken input, speech recognition will produce a *word lattice*. A word lattice is a labelled directed acyclic graph. The vertices of the graph represent boundaries in the speech signal. The edges are labelled with words or segments of words that have been recognised.

pre-processor The pre-processor entails at least the functionality of a Part-of-Speech (PoS) tagger and spelling-corrector. PoS tagging, if effective, lightens the task of the parser considerably; it prevents the creation of a lot of alternative interpretations. Spelling correction should repair spelling errors to some extent. In addition, the following tagging functionality is necessary:

- The recognition of proper names that are in the database.
- The recognition of numbers and ordinals.
- The tagging of special phrases like references to time and/or date.

In the case of spoken input spelling correction can be left out. Instead the pre-processor has to prune the word lattice it receives from the speech recogniser. We assume the output of the pre-processor to be an annotated utterance stored in a *chart*. A chart is a labelled directed graph. The vertices of the graph are word boundaries, edges represent words or word groups, and the labels represent syntactic information (categories, or features).

syntactic parser The syntactic parser assigns syntactic structure to the user input. It receives a chart from the pre-processor, and it outputs a syntactic representation which we assume to be a set of trees and/or feature structures ranked w.r.t. preference. The parser needs a specification of a grammar.

semantic interpreter The semantic interpreter applies selectional constraints to rule out parse trees, and completes predicate argument relations where necessary and possible. It maps the output of the parser to a semantic representation, probably a formula in a logic tailored to application in dialogue systems. For instance, the Quasi Logical Form (QLF) (Alshawi 1992) or Conceptual Graphs (CGs) (Sowa 1984) can be used for semantic representation. It outputs as many formulae as there are possible interpretations. The interpreter receives a syntactic representation from the parser, and needs access to the knowledge base.

dialogue manager The dialogue manager performs the *contextual interpretation* of the user input, and maintains the *context* of the current dialogue. The context of a dialogue is a representation of the history of the dialogue. The dialogue manager selects one interpretation of the alternatives it receives from the semantic interpreter by interpreting each of them in the context of the dialogue. An important part of this interpretation phase is the resulution of anaphora and ellipsis. After it has selected an interpretation, it does not discard the other interpretations. If the continuation of the dialogue reveals the erroneous interpretation of an earlier utterance, then an alternative interpretation of that utterance can be retrieved.

The dialogue manager is responsible for the formulation of queries to meet the user's information need, it performs updates (for instance, ticket reservations), and it initiates the response generation. So, the dialogue manager is responsible for the *pragmatic* interpretation of the user input.

database manager The database manager provides access to the database. It supports both querying and updating. It accepts commands from the dialogue manager in some query language, returns information retrieved from the database, and it confirms the update of the database according to requests of the dialogue manager.

speech/language generation The generation module takes care of responding to the user. We assume the use of template sentences for language generation. See the section on the storage modules for an explanation of template sentences. The dialogue manager commands the generation module by giving it a response type, the information elements the generated response should contain, or the type of information it should ask for. Information elements are phrases that originate from the database, or references to date and/or time.

1.3.3 Storage modules

Now we discuss the storage modules.

knowledge base The knowledge base is a database for world and domain knowledge. Separating world and domain knowledge from the other modules keeps

the dialogue system more portable to other application domains and improves maintainability. An important part of such a knowledge base is an ontology of the concepts that are important to the domain. Many knowledge representation languages exist to date; we mention the KL-ONE family of description logics (Woods and Schmolze 1992) of which LOOM (Brill 1993) and BACK (Hoppe et al. 1993) are members. The semantic interpreter and the dialogue manager will depend on the knowledge base for the semantics of domain-specific concepts and world knowledge.

database We require the database to be a separate module to make sure that the other modules cannot exploit database internal peculiarities, and to obtain database independence to some extent. The dialogue manager (see below) may update the database only through the database manager.

speech model In the case of a spoken system we need a model for the recognition of spoken language. Often speech recognition is based on hidden markov models.

lexicon For domain-specific applications of natural language dialogue, the lexicon can best be divided in two sub-lexicons. One lexicon is domain-specific, and provides detailed syntactic and domain-specific information for each word it contains; for the domain-specific information references to domain and world knowledge are made. The other lexicon aims at broad coverage and provides superficial syntactic information (syntactic category only, for instance); it makes references to world knowledge only.

grammar The grammar specification is an adequate description of the syntactic structure of the language used in the SCHISMA domain. Certain constructions (noun phrases, for instance) should be modelled more specific than others.

template sentences Template sentences are sentences with slots. These slots can be filled with phrases about date, time, or (other) information from the domain or the database. Slots are annotated with agreement information. Given a response type, the information that should be asked for, and the information it should provide or mention, the generation module can select the template sentence that is applicable.

The semantic interpreter, dialogue manager, and response generator apply *world knowledge* and *domain knowledge*, i.e. world knowledge specific to the domain of the dialogue system. World knowledge is often thought of as divided in *declarative* and *procedural* knowledge. Declarative knowledge is best described as factual knowledge about objects and agents in the real world. Procedural knowledge is knowledge about how to do something: in the context of a dialogue system, it is knowledge about what situation should trigger what actions. We believe, the most elegant solution to the problem of modelling world and domain knowledge would be, in theory, to maintain a database that can

be queried as necessary. In practice, world and domain knowledge are often spread over several modules and specifications of the system. For instance, the grammar specification for the parser is sometimes a semantic grammar, or the implementation of the dialogue manager contains 'hard-coded' domain-specific actions.

1.3.4 Interfaces between the modules

We believe the communication between the modules should be standardised in some way. Several languages or *interlinguae* for this purpose can be found in the literature. We mention the Knowledge Query and Manipulation Language (KQML) (Labrou and Finin 1997), which can be used as a language for two or more intelligent systems (agents) to share knowledge in support of cooperative problem solving, and we mention the Knowledge Interchange Format (KIF) (Genesereth 1998), which is a computer-oriented language for the interchange of knowledge among disparate programs.

Another possibility is the definition of a communication language using Standard Generalized Markup Language (SGML) (Goldfarb 1990) or Extensible Markup Language (XML) (Bradley 1998). A disadvantage is that the protocol of the communication cannot be defined in SGML/XML. The parsing system we present in chapter 6 has an SGML output interface for syntax trees and feature structures. The choice for an SGML interface was motivated by the evaluation of the parser on SGML-annotated data. See also section 5.1 for a short introduction to SGML.

1.3.5 Keyboard input versus spoken input

In principle, we restrict ourselves to the parsing task in a keyboard-driven SCHISMA system. However, the results of our research are relevant to spoken dialogue systems as well, if we keep the following in mind:

- The output of a speech recogniser is word lattice possibly annotated with measures that indicate the quality of the recognition. It may be very large, and therefore it may be necessary to prune it before sending it to the parser.
- The problems that occur in the processing of spoken language are different, and spoken language use is different from 'typed' language. For instance, restarts and corrections are a problem for speech recognition, whereas, in the case of keyboard input, typing errors are a source of problems.
- In general, spoken language cannot be described by the same language model (i.e. lexicon and grammar) as we would use for 'typed' language. As a consequence, if we induce a grammar from data that was collected in experiments with a keyboard interface, this grammar is not guaranteed to be a good model for spoken language.

1.4 Contribution

The contribution of this thesis to language technology is the experimental evaluation in a task-oriented domain of probabilistic extensions of unification grammars. Other contributions are the definition of an annotation scheme, and the development of a flexible method for inferring grammars from annotated data. Below we give an overview of the contribution of our research in the disciplines of specification, experimentation, and implementation.

Specification

We specify an annotation scheme for syntax and syntactic relations. We apply Standard Generalized Markup Language (SGML), and the specification of the annotation scheme is a so-called Document Type Definition (DTD). We annotated 873 client utterances from the SCHISMA corpus according to this scheme. The resulting collection is called the SCHISMA Treebank, and serves as testing material for our experiments.

Experimentation

Experiments will show how context-free and unification grammars can be derived from the SCHISMA Treebank in a flexible way. We divide the tree-bank in a train and test set, and compare the performance of context-free grammars, PCFGs, and probabilistic unification grammars.

Implementation

To make possible the experiments with the SCHISMA Treebank, we had to write and adapt some software. We mention:

syntactic parser A robust syntactic parsing system for the SCHISMA domain.

Statistics::MaxEnt A module for Maximum Entropy modelling (ter Doest 1998b).

grammar inference A highly configurable engine for the induction of grammars from SGML-annotated data.

parser evaluation A module for the evaluation of parsing systems with bracket precision, bracket recall, and crossing brackets measures. A module for cross-evaluation of several test runs.

environment An environment for the training and testing of grammars derived from the SCHISMA Treebank. The software components described above are integrated into this environment.

1.5 Organisation of the thesis

This introductory chapter is followed by a chapter that surveys the literature on probabilistic extensions of grammar formalisms and probabilistic parsing. Then two somewhat theoretical chapters follow, which explain the theory of unification grammars and maximum entropy modelling. Chapters 5 and 6 are more practical in nature. They present the SCHISMA Treebank, and experiments with probabilistic extensions of context-free grammars and unification grammars. Chapter 7 gives the conclusions of the thesis. Below we give a short description of each of the chapters.

Chapter 2 Probabilistic Grammar, a Survey

This chapter gives an overview of research on the application of statistics in natural language parsing. We start with a short introduction to context-free grammar, and explain the 'classical' model of probabilistic context-free grammar. Then we give a survey of the literature on probabilistic grammar formalisms and probabilistic parsing.

Chapter 3 Unification-based Parsing

In chapter 3 we explain the theory of feature structures and unification grammars. We define a parsing schema for left-corner chart parsing with unification grammars, and discuss possible approaches to the probabilistic extension of unification grammars.

Chapter 4 Maximum Entropy Modelling

In chapter 4 we explain the maximum entropy formalism which goes back to (Jaynes 1957). Maximum entropy modelling is a way of inducing probability distributions from real world data. Its main feature is its ability to select probability distributions without making any assumption except for the expectation values of properties that we believe are characteristic for the data. It will be explained how the principle of maximum entropy is applied to achieve this. Then we discuss two so-called scaling algorithms that estimate the parameters of maximum entropy distributions, and an algorithm for the induction of important properties.

Chapter 5 The Grammar Inference Engine

In chapter 5 we present the annotation scheme we developed for annotating the SCHISMA Treebank. We discuss the syntactic tags chosen, and how syntactic relations are represented. Examples will help clarify the annotation scheme. It is explained how unification grammars can be derived from the tree-bank. In particular meta-constraints, tag patterns that trigger the generation of certain unification constraints, are treated in detail.

Chapter 6 Experiments

We discuss the parser that we use for our experiments, and we explain the evaluation procedure that we apply to measure the performance of the parser. We present experiments to investigate the performance of the parser with probabilistic extensions of context-free grammars and unification grammars derived from the SCHISMA Treebank.

Chapter 7 Conclusions

Chapter 7 presents the conclusions of this thesis. We recommend directions for future research. Both practical issues like the application of our results in the SCHISMA project, and theoretical subjects like the induction of good probabilistic models are covered.

Appendix A Schisma Treebank DTD

Appendix A contains the SGML Document Type Definition that specifies the annotation scheme of the SCHISMA Treebank.

Appendix B Meta-constraints

In appendix B we give the specification of the meta-constraints that we used to generate a unification grammar from the SCHISMA Treebank.

Appendix C Samenvatting (Summary, in Dutch)

Samenvatting van dit proefschrift in het Nederlands.

Part I

Basics

Chapter 2
Probabilistic Grammars, a Survey

In this chapter we give an overview of the literature on probabilistic extensions of grammar formalisms and probabilistic parsing.

The first section gives a short explanation of the theory of context-free grammars. Then, in section 2.2, we treat the "classical" model of probabilistic context-free grammars (PCFGs). We discuss supervised and unsupervised training of PCFGs, and identify the most important defects of the formalism. In section 2.3 we discuss some approaches directly based on context-free grammar. Section 2.4 discusses approaches that attach probabilities to parser actions. In section 2.5 we study probabilistic extensions of two tree-based grammar formalisms: tree substitution grammar and tree adjoining grammar. Section 2.6 gives an overview of probabilistic extensions of unification grammar formalisms that are based on the PCFG approach. In section 2.7 we discuss grammar weighting schemes based on constraint logic programming.

2.1 Context-free grammars

Before we turn to probabilistic grammar formalisms, we give a short introduction to the theory of context-free grammars. "THE CONCEPT OF A CONTEXT-FREE LANGUAGE was first introduced by Chomsky in 1959 [Ch 3] in an attempt to find a reasonable mathematical model of natural languages such as English, French, etc."[1] (Ginsburg 1966).

Formally, a context-free grammar (CFG) is a quadruple (V_N, V_T, P, S) where V_N and V_T are the symbols of the grammar, $S \in V_N$ the *start symbol*, and P a set of *production rules* (or rules). The symbols V_N are called *nonterminals*, and can be rewritten by applying grammar rules. The symbols in V_T are called *terminals*, and cannot be rewritten. The rules in P are of the form $A \rightarrow \alpha$, where A is a nonterminal and $\alpha \in (V_N \cup V_T)^*$ is a string of grammar symbols. A is called the left-hand side (LHS) of the rule, and α the right-hand side (RHS).

A string $\alpha A \gamma$ can be rewritten into $\alpha\beta\gamma$, denoted by $\alpha A \gamma \Rightarrow \alpha\beta\gamma$, if a rule $A \rightarrow \beta$ exists. A *derivation* is a sequence of such rewritings. $\Rightarrow^*$ denotes the reflexive, transitive closure of the derivation relation. A *leftmost derivation* is a derivation in which each derivation step rewrites the leftmost nonterminal.

[1] [Ch 3] refers to (Chomsky 1959). The smallcaps are by Ginsburg.

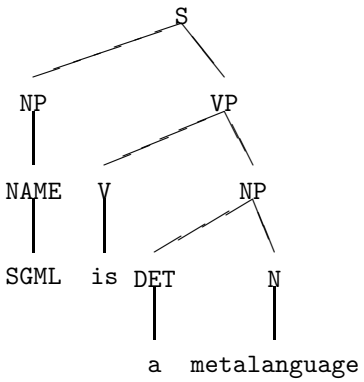


Figure 2.1: Example of a derivation tree.

The *language of a grammar* $L(G)$ is the set of terminal strings that can be derived from the starting symbol S:

$$L(G) = \{x \in V_T^* | S \Rightarrow^* x\}$$

Below is an example of a context-free grammar.

Example 2.1 (context-free grammar)
`S` is the start symbol of the following grammar:

```
S -> NP VP
NP -> NAME
VP -> V NP
NP -> DET N
NAME -> SGML
V -> is
DET -> a
N -> metalanguage
```

The nonterminals of the grammar are `S`, `NP`, `VP`, `DET`, `NAME`, `N`, `V`. The terminals are `SGML`, `is`, `a`, and `metalanguage`. A leftmost derivation w.r.t. this grammar is

```
S => NP VP => NAME VP => SGML VP => SGML V NP =>
  SGML is NP => SGML is DET N => SGML is a N =>
    SGML is a metalanguage
```

The intermediate derivation results above are called *sentential forms*. The corresponding derivation tree is given in figure 2.1. Another sentence that can be generated with this grammar is `a metalanguage is SGML`.

2.2 Probabilistic context-free grammars

Probabilistic context-free grammars (PCFGs) are the most common and well-understood probabilistic grammars. In general, PCFGs are defined as context-free grammars extended with a function that assigns a probability to each rule. Formally, a *probabilistic context-free grammar* is a quintuple (V_N, V_T, S, P, π) where (V_N, V_T, S, P) is a context-free grammar, and $\pi : P \to [0, 1]$ is a function that assigns a probability to each rule such that for each $A \in V_N$:

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

The probability of a derivation tree of the grammar is defined as the product of the probabilities of the subtrees directly below the top node and the probability of the production rule apply to the nonterminal of the top node. Formally, given a derivation tree $\sigma = \sigma(\sigma_1 \dots \sigma_n)$, $\sigma \in P$, and σ_i derivation trees, the probability of the derivation is given by:

$$p(\sigma()) = \pi(\sigma)$$

$$p(\sigma(\sigma_1 \dots \sigma_n)) = \pi(\sigma) \prod_{i=1}^{n} p(\sigma_i)$$

If more distinct derivation trees derive the same sentence, the sum of the probabilities of all possible derivations is taken as the probability of the sentence (so-called *sum-interpretation*):

$$p(w) = \sum_{\sigma \in Trees(G): yield(\sigma) = w} p(\sigma)$$

where *Trees*(G) is the set of trees grammar G generates. Similarly to the sum interpretation, the *max-interpretation* can be used. In applications of natural language parsing often the parse tree with maximum probability is considered the right one and selected for further interpretation.

A nice property of PCFGs is that the probability of a sentence decreases with its length. This corresponds to our intuition that long sentences have a smaller probability to occur. We refer to (Suppes 1972) for a lengthier discussion of the naturalness of PCFGs. For investigations into the more theoretical aspects of PCFGs we refer to (Grenander 1967; Booth and Thompson 1973; Wetherell 1980; ter Doest 1994).

2.2.1 Supervised training

PCFGs can easily be inferred from a so-called tree-bank. Given a set $E = \{e_1, e_2, \dots, e_N\}$ of trees (a tree-bank), the probability of a rule $A_i \to \alpha_j$, where $A_i \in V_N$, and $\alpha_j \in (V_N \cup V_T)^*$, is defined by

$$\pi(A_i \to \alpha_j) = \frac{n(A_i \to \alpha_j)}{n(A_i)} \tag{2.1}$$

where $n(A_i)$ denotes the number of times a (sub)tree occurs in the tree-bank with nonterminal A_i as its root, and where $n(A_i \rightarrow \alpha_j)$ is the number of times a (sub)tree occurs that is generated by $A_i \rightarrow \alpha_j$.

A PCFG with rule probabilities that are estimated according to (2.1) induces a probability distribution on its language that has maximum likelihood of generating the data used to estimate the probabilities from; see (Fu and Booth 1975) for a proof. Abney (1997) uses the term expected rule frequency (ERF) for this estimation.

2.2.2 Unsupervised training: the Inside-Outside algorithm

An unsupervised way to infer a PCFG is using the Inside-Outside algorithm (Lari and Young 1990). The Inside-Outside algorithm is a generalisation of the Baum-Welch algorithm used to train Hidden Markov Models (Baum 1972). Charniak (1993) explains how the Inside-Outside algorithm can be derived from the Baum-Welch algorithm.

The basic idea of the Inside-Outside algorithm is to use the current rule probabilities to estimate from the sample the expected frequencies of certain derivation steps, and then compute new rule probability estimates as appropriate frequency rates. Each iteration of the algorithm starts by calculating the *inside* and *outside* probabilities for all sentences in the sample. These probabilities are in fact probability functions that have as arguments a sentence w from the sample, indices that indicate what part of sentence w should be considered, and a nonterminal, say A_k. With these arguments, the inside probability $I_w(i, j, A_k)$ is the probability that A_k derives $w_{i+1} \ldots w_j$. The outside probability $O_w(i, j, A_k)$ is the probability that a sentential form $w_1 \ldots w_{i-1} A_k w_{j+1} \ldots w_{|w|}$ can be derived.

We give a formulation along the lines of (Pereira and Schabes 1992) where an adapted version of the Inside-Outside algorithm is given for re-estimation from partially bracketed corpora. The grammar is assumed to be in Chomsky Normal Form (CNF). This means that a rule is either of the form $A \rightarrow B\ C$, with $A, B, C \in V_N$, or of the form $A \rightarrow a$, with $A \in V_N$ and $a \in V_T$. In (Kupiec 1992) the Inside-Outside algorithm is defined for general context-free grammar.

Given the inside and outside probabilities, and a sample S of (not annotated) sentences, the probability of rule $A_p \rightarrow A_q A_r$, here denoted by P_{pqr}, is re-estimated as follows:

$$\hat{P}_{pqr} = P_{pqr} \frac{\sum_{w \in S} 1/P(w) \sum_{0 \leq i < j < k \leq |w|} I_w(i, j, A_q) I_w(j, k, A_r) O_w(i, k, A_p)}{\sum_{w \in S} P_w(p)/P(w)}$$

where $P_w(p)$ is the probability that nonterminal A_p is involved in a derivation of w

$$P_w(p) = \sum_{0 \leq i < j \leq |w|} I_w(i, j, A_p) O_w(i, j, A_p)$$

and $P(w)$ the probability of w according the current model, or $I_w(0, |w|, S)$ in terms of the inside probability. Intuitively, P_{pqr} is re-estimated by the ratio of the expected number of times that rule $A_p \rightarrow A_q A_r$ is used in the sample, and the expected number of times nonterminal A_p is used in the sample. Similarly, the probability of rule $A_p \rightarrow a_q$, denoted by P_{pq}, is re-estimated according to

$$\hat{P}_{pq} = P_{pq} \frac{\sum_{w \in S} 1/P(w) \sum_{1 \leq i \leq |w| : w_i = a_q} O_w(i-1, i, A_p)}{\sum_{w \in S} P_w(p)/P(w)}$$

The rule probabilities are known to converge such that the PCFG has maximum likelihood of generating the sample.

The re-estimation algorithm can be used both to refine the current estimated probabilities of a PCFG and to infer a PCFG from scratch. The former application can be said to be incremental. In the latter case, the initial grammar consists of all possible CNF rules over given sets of nonterminals and terminals. The inference process should then be initialised with suitable (a priori or random) nonzero probabilities.

2.2.3 Discussion

Presently it is generally agreed upon that PCFGs lack representational power in the probabilistic sense, and that they are not flexible enough for the incorporation of probabilistic information other than rule frequencies.

context-sensitivity For the disambiguation of phenomena like PP attachment or long distance dependencies the scope of rule probabilities is too small. In PCFG the probability of a rule expresses the probability that it is used for rewriting its LHS nonterminal; this is called the *context-freeness assumption*. But if we want to model a phenomenon like PP attachment statistically, and some researchers try to do so, we need distributions that model characteristics of syntax trees that go beyond that of context-free grammar rules. A lot of research on statistical language models concentrates on richer context-sensitive statistics. One obvious improvement is to condition rule probabilities on characteristics of syntax trees or derivations beyond grammar rules. In section 2.3 we discuss some of these approaches. Another more fundamental solution is to abandon context-free grammar, and use a different grammar formalism. Sections 2.5 through 2.7 give an overview of such attempts.

lexical information PCFGs are not very flexible regarding the incorporation of statistical information other than rule frequencies. For instance, co-occurrence statistics (n-grams) of words or syntactic categories hold important information about lexical dependencies. Incorporation into the PCFG of such information obscures the notion of what events are modelled by the resulting statistical model of language: “A difficulty with the hybrid approaches, however, is that

they leave unclear what the statistical model of the language *is*. In probabilistic context-free grammar and in surface-string analysis of lexical co-occurrence, there is a precise definition of the *event space* – what events go into making up a sentence. (...) The absence of such a characterization in the hybrid approaches makes it more difficult to identify what assumptions are being made, and gives such work a decidedly empirical flavor." (Resnik 1992) (p. 421). The tree grammar formalisms that we treat in section 2.5 provide a framework for the incorporation of lexical statistics. The maximum entropy method, treated in chapter 4, allows the incorporation of arbitrary information sources.

generative nature Implicit in the definition of PCFGs is the assumption that the application of a rule changes the probability of the final derivation tree. The probability distribution over the language of a grammar is defined in a generative way. I think PCFGs narrow the mind in thinking about probabilistic languages too much. Why depend on the grammar for defining the probability of a tree whereas we can define a probabilistic model that does not depend on the grammar at all for the computation of the probability of a derivation tree? In chapter 4 we will see how this can be done.

2.3 Extensions of probabilistic context-free grammars

As we saw in the previous section, the PCFG formalism is simply not rich enough to capture phenomena that are outside the scope of context-free grammar rules. In this section we discuss some approaches that try to overcome this problem by conditioning rule probabilities on information outside the scope of context-free grammar rules.

2.3.1 Weakly restricted stochastic grammars

In (op den Akker and ter Doest 1994) the authors propose a slight enrichment of the PCFG formalism in order to obtain some context-sensitivity for the rule probabilities. The idea is to assign rule probabilities depending on where in the grammar the nonterminal to be rewritten was introduced. It is argued that the knowledge about the rule that introduced the nonterminal will improve the context-sensitivity of the probabilistic model.

Formally, a weakly restricted stochastic grammar (WRSG) is a quintuple (V_N, V_T, S, P, π) where (V_N, V_T, S, P) is a context-free grammar and π assigns to each RHS nonterminal occurrence A_{ij} (which denotes the j-th RHS occurrence of A_i) a function π_{ij} that assigns to each rule for A_i a probability. So, the probability of a rule to rewrite some nonterminal depends on where in the grammar it was introduced.

It is shown that a weakly restricted grammar can be transformed into a PCFG that is stochastically strong equivalent. This implies that the increase in probabilistic descriptive power is drawn from the size of the grammar, and not from defining a more powerful formalism. Also an adapted version of the

Inside-Outside algorithm is given to enable training and inference of WRSGs directly from a tree-bank or from raw language data.

2.3.2 History-based grammars

In (Black et al. 1992) a sophisticated model of probabilistic grammar, called *history-based grammar* (HBG), is introduced. The idea is that probabilities are conditioned on the history of applied grammar rules (leftmost derivation is assumed). HBG is a framework for rich probabilistic grammar models. The rule probabilities are conditioned on the equivalence class of the sequence of previously applied rules. Formally, the probability of a derivation $\sigma = \sigma_1 \dots \sigma_n$, $\sigma_i \in P$, is defined by

$$p(\sigma) = \prod_{i=1}^{m} p(\sigma_i | [\sigma_1 \dots \sigma_{i-1}])$$

where m is the number of derivation steps, and $[\sigma_1 \dots \sigma_{i-1}]$ the equivalence class of the previously applied grammar rules.

The grammar used for experiment with the HBG model is a hand-crafted feature grammar that uses unification. Terminals and nonterminals are feature-value pairs, similar to the model Goodman (1997) applies (see section 2.6.3 for a discussion). The difference is that some of the features are compiled into the context-free backbone. The PCFG thus constructed is trained on the Lancaster Treebank using the Inside-Outside algorithm. The grammar thus obtained is used for later evaluation of the HBG model, and for 'bootstrapping' the induction of the HBG.

To obtain an HBG, the PCFG is run on the Lancaster Treebank. Each most likely parse is considered an event if it matches the tree-bank analysis. Each node of these trees is annotated with syntactic (*Syn*) and (*Sem*) features, its primary and secondary head, and the rule that is applied to rewrite it. The proposed model associates for each node of each parse tree the following conditional probability:

$$p(Syn, Sem, R, H_1, H_2 | Syn_p, Sem_p, R_p, I_{pc}, H_{1p}, H_{2p})$$

Subscript p indicates features of the parent node, and I_{pc} is the index of the node in the RHS of the rule that is applied to its parent node. This rule is denoted by R_p. Then a statistical decision tree (see (Magerman 1994) for an explanation) is designed to approximate this probability with the probability that a rule will be used for rewriting a node in the parse tree:

$$p(R | Syn, Sem, Syn_p, Sem_p, R_p, I_{pc}, H_{1p}, H_{2p})$$

The performance of the HBG shows a considerable improvement over the performance of the PCFG that was constructed to create the event space of parse trees.

2.3.3 Salomaa's probabilistic grammars

A probabilistic grammar according to (Salomaa 1969) is a grammar extended with probabilities such that each rule application receives a probability dependent on the last rule applied. Formally, a probabilistic grammar of type i is a triple (G, δ, ϕ) where $G = (V_N, V_T, S, P)$ is a type i grammar, $\delta : P \to [0, 1]$, and $\phi : P \times P \to [0, 1]$. The probability of a derivation $\sigma = \sigma_0, \ldots, \sigma_n, \sigma_i \in P$ is defined by

$$p(\sigma) = \delta(\sigma_0) \prod_{i=1}^{n} \phi(\sigma_{i-1}, \sigma_i)$$

Clearly, the derivation order (left-most, or right-most, for instance) results in different probabilities. Under the maximum interpretation the probability of a string is the maximum probability of all possible derivations. The sum interpretation instructs us to take the sum of the probabilities of all the derivations of the string.

Salomaa does not mention the problem of inference or training. His paper concentrates mainly on theoretical issues like the hierarchy of stochastic languages, and the relation of weighted grammars to programmed grammars and time-variant grammars.

Although potentially much richer than PCFG, HBG is restricted to modelling statistical dependencies that are in the history of the lefmost derivation. The description of the experiments by (Black et al. 1992) shows that implementation of an HBG model is a complex undertaking. The claim that WRSG is stochastically strong equivalent to PCFG makes it an interesting approach. Salomaa's definition of probabilistic grammar seems of theoretical interest only, as the matter of parameter estimation is not solved.

2.4 Statistics of parser actions

Another approach to the enrichment of probabilistic grammar is by extending the parser. Each parsing algorithm has a limited set of actions it can perform in analysing a sentence. Typically, LR parsers do shift and reduce actions, left-corner parsers shift, hypothesise, or complete. If each of these actions receives a probability, then the probability of a derivation can be computed as the product of the probabilities of the actions performed for building the derivation. Action probabilities can be computed from an initial PCFG, or induced from a tree-bank. In this section we discuss two probabilistic parser models: a probabilistic LR parser that is compiled from a PCFG, and a probabilistic left-corner parser that estimates its action probabilities from a tree-bank.

2.4.1 Probabilistic GLR parsing

Wright and Wrigley (1991) extend Generalised LR parsing by compiling probabilities into the parsing tables. They describe how PCFG probabilities are

transformed into probabilities of LR parser actions. Algorithms are given for SLR, LALR, and canonic LR parsers. The transformation is based on the idea that LR state transitions correspond to shift and reduce actions.

Furthermore, Wright and Wrigley give a Bayesian interpretation of the LR action probabilities, propose a solution to the unknown words problem, and show that probabilistic LR performs far better than Markov models.

See-Kiong and Tomita (1991) report on experiments with a number of corpus-based grammars based on this model of probabilistic GLR parsing. (Inui et al. 1997) gives another formalisation of probabilistic GLR. In section 2.6 we look at work by Briscoe and Carroll on a model of probabilistic LR parsing combined with a unification grammar.

2.4.2 Probabilistic left-corner parsing

Manning and Carpenter (1997) present a probabilistic parser based on a left-corner stack parser with probabilities assigned to its actions. A left-corner stack parser employs a stack to keep track of the constituents it has found so far, and the constituents it is looking for.

Typically, the parser employs three types of actions: *shift*, *attach*, and *lc-project*. A *shift* action moves the dot over a lexical category, an *attach* action shifts over a recognised consitituent, and an *lc-project* action predicts new constituents using the left-corner relation. For an explanation of left-corner stack parsing we refer to (op den Akker 1988; Nederhof 1994).

The probability of a left-corner derivation is defined as the product of the probabilities of the parser actions applied during parsing. If the probability of each action is conditioned on the history of parser actions, we have

$$P(t) = \prod_i P(C_i|C_1, \ldots, C_{i-1})$$

The authors recognise that in practice such conditional probabilities cannot easily be estimated. They propose two models that classify histories of parser actions. Note that the same problem arises with the definition of History Based Grammar (see section 2.3.2), where a classification is defined on the history of applied rules. In the simplest one, the history of parser actions is classified according to the current left-corner category and goal category. It is argued that this results in a probabilistic model slightly richer than the PCFG formalism. Experiments on the Penn Treebank II show a worthwhile performance improvement over standard PCFG.

Furthermore a classification of parser histories is explored that is based on the left-corner and goal categories (as before), and the depth of the stack. No experiments are reported for such extended left-corner models.

2.5 Tree-based formalisms

In this section we treat two tree-based formalisms, stochastic lexicalised tree adjoining grammars (SLTAG) and stochastic tree substitution grammar (STSG).

Both formalisms are more powerful than the PCFG formalism, and therefore interesting alternatives.

2.5.1 Tree adjoining grammar

Stochastic lexicalised tree adjoining grammars (SLTAGs) (Schabes 1992) are stochastic extensions of lexicalised tree-adjoining grammars (LTAGs) (Joshi and Schabes 1992). We refer to (Joshi 1987) for an introduction to tree adjoining grammars (TAGs).

Schabes (1992) gives a definition of SLTAGs by defining stochastic linear indexed grammars (SLIGs), and showing how their stochastic languages are related. We follow the more intuitive definition of SLTAGs by Resnik (1992). A lexicalised TAG is a pair (I, A) where I is the set of initial trees, and A the set of auxiliary trees. Each auxiliary tree has a frontier node that matches the root node. It is called the *foot* node, and the path from root to foot *spine*.

Adjunction and substitution are the two operations that allow building more complex trees from I and A. Each tree should contain a terminal symbol on its frontier to be lexicalised.

substitution each tree in $I \cup A$ has a (possibly empty) subset of its frontier nodes marked as nodes at which an initial tree may be substituted; given a tree α this set is denoted $s(\alpha)$;

$S(\alpha, \alpha', \eta)$ denotes the event of substituting α' into tree α at node η; let S denote the set of all substitution events;

adjunction the adjunction operation replaces a tree node γ by an auxiliary tree; the root and foot nonterminal should of course match the nonterminal of the node that is being replaced; the node is not allowed to be a substitutable node; give a tree α, the set of adjunction nodes is denoted $a(\alpha)$;

$A(\alpha, \beta, \eta)$ denotes the event of adjoining auxiliary tree β into tree α at node η; $A(\alpha, \mathit{none}, \eta)$ is the event that no adjunction is performed at η; let A denote the set of all adjunction events.

LTAGs are extended with statistics by introducing the following probability functions:

- $P_I : I \rightarrow [0, 1]$ assigns a probability to each initial tree such that

$$\sum_{\alpha \in I} P_I(\alpha) = 1$$

- $P_S : S \rightarrow [0, 1]$ assigns a probability to each substitution event such that

$$\forall \alpha \in I \cup A : \forall \eta \in s(\alpha) : \sum_{\alpha' \in I} P_S(\alpha, \alpha', \eta) = 1$$

- $P_A : A \rightarrow [0, 1]$ assigns a probability to each adjunction event such that

$$\forall \alpha \in I \cup A : \forall \eta \in a(\alpha) : \sum_{\beta \in A \cup \{none\}} P_A(\alpha, \beta, \eta) = 1$$

The probability of a derivation is the product of the probabilities of the substitution and adjunction events, and the probability of the initial tree. If the derivation is given by $\tau = \sigma_1, \ldots \sigma_n$, $\sigma_i \in S \cup A$, and the initial tree is given by α_0, then we have

$$p(\tau) = P_I(\alpha_0) \prod_i P(\sigma_i)$$

Schabes (1992) gives an algorithm for computing the probability of a sentence (in $O(n^6)$ time and $O(n^4)$ space), the Inside-Outside algorithm is given for SLTAGs, and experiments are described in which SLTAGs are automatically constructed from a corpus using the Inside-Outside algorithm. One particular advantage of SLTAGs over PCFGs is the faster convergence of the former. The author claim this to be a consequence of PCFGs' deficit in the representation of lexical influences on the distribution, while SLTAGs integrate both lexical distributional and hierarchical statistics.

An interesting restriction of the SLTAG formalism is given by stochastic lexicalised tree insertion grammars (SLTIGs) (Schabes and Waters 1993). SLTIGs are based on LTIGs (Schabes and Waters 1994), and LTIGs are restricted from LTAGs by forbidding *wrapping* auxiliary trees. This restriction is accomplished if

- there are no wrapping elementary trees, i.e. auxiliary trees that have non-empty frontier nodes on both left and right side of the spine;
- left (right) auxiliary trees cannot be adjoined to a node that is on the spine of an elementary right (left) auxiliary tree; and
- there is no adjunction allowed to the right (left) of the spine of an elementary left (right) auxiliary tree.

LTIGs lexicalise CFGs without changing the trees that are derived. Furthermore a constructive procedure exists to transform any CFG into an equivalent LTIG (Schabes and Waters 1994).

SLTIGs can be parsed in $O(n^3)$ time, and Inside-Outside training of an SLTIG can be accomplished in $O(n^4)$ time. It is claimed that an Inside-Outside algorithm can be constructed that has time complexity $O(n^3)$ just like the parsing algorithm.

2.5.2 Data-oriented parsing

In data-oriented parsing (DOP) (Bod and Scha 1997; Bod 1998) stochastic tree substitution grammars (STSGs) are defined, a generalisation of PCFGs. An STSG is a PCFG with the set of rules replaced by a set of trees. Formally, an STSG is a quintuple (V_N, V_T, S, T, π) where

- V_N is a finite set of nonterminals;
- V_T is a finite set of terminals;
- $S \in V_N$ is the start symbol;
- T a finite set of elementary trees; an elementary tree is a tree with nonterminals at the root and the interior nodes; the final nodes may be either terminal of nonterminal; $root(t)$ denotes the nonterminal at the root of tree t;
- π is a function that assigns a probability to each elementary tree, such that for each nonterminal A

$$\sum_{t:root(t)=A} \pi(t) = 1$$

The notion of derivation is based on the binary substitution operator $\cdot$. If t_1 and t_2 are trees, and the leftmost frontier nonterminal of t_1 equals the root of t_2, then $t_1 \cdot t_2$ is the tree that results from substituting t_2 for the leftmost frontier nonterminal. A leftmost derivation is defined as a sequence of elementary trees $(t_1, \ldots, t_n)$, such that $t_1, \ldots, t_n \in R$, the root of t_1 is labelled by S, and the yield of $t_1 \cdot \ldots, \cdot t_n$ is a string of terminals. The probability of a derivation is the product of the elementary trees it consists of. Clearly, a parse tree may have multiple (leftmost) derivations; the probability of a parse tree is defined as the sum of the probabilities of its distinct derivations. The string language of an STSG consists of those strings for which a parse tree exists; the probability of a string is the sum of the probabilities of its distinct parses, and this equals the sum of the probabilities of its distinct derivations.

STSGs are weakly equivalent to PCFGs, e.g. "The set of stochastic string languages generated by STSGs is equal to the set of stochastic string language generated by SCFGs" (Bod 1998) (p. 29).[2] Bod shows that STSGs "are not only stronger than SCFGs because there are STSGs for which there are no strongly equivalent SCFGs, but that STSGs are really *stochastically* stronger, also with respect to SCFGs that are strongly equivalent to STSGs" (Bod 1998) (p. 33).

An STSG can easily be inferred from a corpus by collecting all and every subtree it contains, count them and normalise by the total number of subtrees that have the same nonterminal at the root. Clearly, STSGs will become very large, even larger than a PCFG would be for the same tree-bank. A possible way to 'tame' STSG is restricting the elementary trees. Bod applies restrictions on the depth of the elementary trees. Still, the grammar may be very large, and parsing an expensive operation. Often the size of the grammar is not taken into account in time complexity analyses, because it is a constant factor. However, if a grammar is large, this constant factor will dominate the sentence length factor.

[2]SCFG stands for stochastic context-free grammar, another term for PCFG.

Bod defines parsing as a sampling procedure. If during parsing, either bottom up or top down, several subtrees can be substituted on the same node, then the parser selects one subtree by sampling, substitutes it, and continues.

Goodman (1996) showed that for each STSG a PCFG can be constructed that is strongly equivalent, i.e. for each STSG derivation tree there exists a PCFG derivation tree that has equal probability, and vice versa. Then he presents a parsing algorithm that runs in $O(n^3)$ time. It does does not produce the most probable derivation, but the *Maximum Constituents Parse*, the parse that is likely to have the largest number of correct constituents. Bod (1996) replies, correctly I think, that Goodman's PCFG model is not a DOP model as it does not produce the same tree language and the parsing algorithm does not produce the same (best) tree as the DOP parsing algorithm.

If we compare the STSG formalism to the SLTAG formalism, we see that the SLTAG formalism is more powerful than the STSG formalism because of the adjunction operation, and SLTAGs can, in theory, "capture at least the stochastic dependencies that can be captured by STSG" (Bod 1998) (pp. 36-37). In both formalisms co-occurrence (or other lexical) statistics can be represented very easily. The main difference is in the philosophy that has motivated the formalisms. In the DOP research the STSG formalism is applied to use a corpus of annotated data as a performance model of human language use(rs), whereas the TAG formalism is a linguistically motivated grammar formalism for the development of hand-crafted grammars.

2.6 Unification grammars

Extending unification grammar with statistics is not a trivial matter. In principle, it is wrong to attach probabilities to the rules of the unification grammar like is done with PCFGs. The possibility of unification failure causes such probabilities and therefore the distribution they induce to be invalid. However, successful experiments that apply such rule probabilities can be found in the literature.

2.6.1 Probabilistic ALE

Brew (1995) develops stochastic HPSG as a pendant of PCFG. His model is based on a probabilistic extension of ALE signatures. ALE stands for Attribute Logic Engine, and "is an integrated phrase structure parsing and definite clause logic programming system in which the terms are typed feature structures" (Carpenter and Penn 1994). An ALE signature is a mixture of type hierarchy and appropriateness specification. Appropriateness is a way of constraining the set of possible feature structures by specifying required features with types and the types these features may introduce. For a more theoretical discussion of well-typedness and appropriateness we refer to (Carpenter 1992).

Brew formalises *probabilistic signature* as a quintuple (T_N, T_T, τ_s, I, P) where

- T_T is a set of maximal types

- T_N a set of non-maximal types;
- τ_s the starting symbol;
- I is a set of introduction relationships of the form $(t_i \Rightarrow t_j) \rightarrow \xi, t_i, t_j \in T_N$ and ξ a multi-set of maximal and non-maximal types; and
- $P : I \rightarrow [0, 1]$ a function that assigns probabilities to introduction relationships such that the sum of the relationships that apply to a given type sums to one.

The probability of a (well-typed) feature structure is defined recurrently by

- $P(\tau_s) = 1$;
- if f and f' are feature structures, and f' differs from f in that a non-maximal node of f having type $\sigma \in T_N$ has been refined to type $\tau \in T_N$, and subsequently expanded to ξ_k, then $P(f') = P(f) \times P((\sigma \Rightarrow \tau) \rightarrow \xi_k)$.

(Brew 1995) is rather cryptic on the issue of training. "Training is a matter of counting the transitions which are found in the observed results, then using counts to refine initial estimates of the probabilities of particular transitions." He explains here that training comprises parsing a corpus, and counting the number of times each introduction relationship is applied. Afterwards counts are normalised to obtain estimations of the probabilities of the introduction relationships.

The matter of re-entrancy is handled by associating a probability p_τ with each non-maximal type $\tau \in T_N$. Given a feature tree, and a possible re-entrancy regarding type τ, then the probability of the part of the structure not involved is multiplied by p_τ, in case the re-entrancy is added; otherwise the structure not involved is multiplied by $1 - p_\tau$. Brew describes this procedure from a generative point of view, whereas we would like to know how to re-estimate the probabilities already found for the introduction relationships.

The practical value of the model is unclear as long as no experiments have been performed. Abney (1997) argues that the estimation procedure defined by (Brew 1995) is based on expected rule frequencies, and that it converges to the wrong distribution.

2.6.2 Probabilistic LR parsing with unification grammars

In (Carroll and Briscoe 1992) and (Briscoe and Carroll 1993) an approach to probabilistic parsing with a unification-based grammar is described. Briscoe and Carroll used the ANLT (Alvey Natural Language Tools) wide-coverage grammar of approximately 800 grammar rules, and a lexicon of about 64,000 entries built from the *Longman Dictionary of Contemporary English* (LDOCE).

The most important features of the grammar are compiled into the context-free 'backbone' from which a non-deterministic LR parser is constructed. "Probabilities are assigned to transitions in the LALR(1) action table via a process

of supervised training based on computing the frequency with which transitions are traversed in a corpus of parse histories constructed using a user-driven, interactive version of the parser" (Carroll and Briscoe 1992).

An LR parser with probabilities on the state transitions can discriminate between distinct derivation trees that are created using the same grammar rules. Clearly, this is impossible for a PCFG: the probability of a derivation tree of a PCFG is independent of the order in which the rules were applied.

2.6.3 Probabilistic feature grammars

In probabilistic feature grammars (PFGs), introduced by Goodman (1997), terminals and nonterminals are vectors of features. Typically a binary PFG rule looks like

$$(a_1 \dots a_g) \rightarrow (b_1 \dots b_g)(c_1 \dots c_g) \tag{2.2}$$

Features are instantiated one by one, and each instantiation is considered an event. Two kinds of events are distinguished: *binary events* and *start events*. A binary event is the application of a rule like (2.2). A start event is the introduction of the root of a tree. Both are modelled probabilistically by so-called *EventProb*'s. An EventProb is a triple $e = (K, N, F)$, where

- K is a set of conditioning features (the known features),
- $N = N_1, N_2, \dots, N_n$ is an ordered list of conditioned features (the new features), and
- $F = f_1, f_2, \dots, f_n$ is an ordered list of functions;
  $f_i(n_i, k_1, \dots, k_k, n_1, \dots, n_{i-1})$ returns the conditional probability $P(N_i = n_i | K_1 = k_1, \dots, K_k = k_k, N_1 = n_1, \dots, N_{i-1} = n_{i-1})$, i.e. the probability that feature N_i receives value n_i given the known features $K_1, \dots K_k$ and the lower indexed new features $N_1, \dots, N_{i-1}$.

Given the PFG rule in (2.2), a binary event is represented by

$$e_b = (\{a_1, \dots, a_g\}, (b_1, \dots, b_g, c_1, \dots, c_g), F_b)$$

which means that the child features $b_1, \dots, b_g$ and $c_1, \dots, c_g$ are conditioned on earlier child features and all parent features $a_1, \dots, a_g$. A start event is modelled by

$$e_s = (\{\}, (a_1, \dots, a_g), F_s)$$

meaning that parent features are conditioned on each other.

Because the number of conditional probabilities that need to be estimated easily become very large, data sparsity is a serious problem. Goodman applies *smoothing* to overcome this problem. Smoothing makes sure that events having a zero frequency in an event space are given a non-zero probability (*are smoothed*).

The paper gives the Inside-Outside algorithm for PFGs, and describes experiments using the Penn Treebank II. Performance comparable to state of the art is reported.

2.6.4 Stochastic attribute-value grammars

A very advanced approach is inspired on *random field theory* as applied in (Mark et al. 1991) to context-free grammars and in (Della Pietra et al. 1997) to orthography. Random field theory is well-known in Bayesian image analysis (Winkler 1995). (Abney 1997) explains how the general idea of random field induction can be applied for estimating parameters of stochastic attribute-value grammar (SAVG).

An attribute-value grammar (AVG) according to Abney is a context-free grammar with attribute labels and path equations. The attribute labels uniquely identify the symbols in the RHS of the rules (or: the children of the LHS node). The path equations specify what paths in the derivation tree lead to the same child node. AVGs can conveniently be represented by feature structures over a set of syntactic category types and integer feature labels that represent the ordering of the children nodes.

Abney explains that maximum likelihood estimation for PCFGs cannot be applied to attribute-value grammars, unification grammars, HPSGs, and other unification-based grammars, because the application of rules is not independent. He proposes the application of MaxEnt models with lattice-based properties. Properties detect substructures of feature structures and new properties can be built from existing ones by combining the structures they detect.

2.7 Extensions of Constraint Logic

Context-free grammars can very conveniently be specified in a logical programming language like Prolog. Moreover parsing with respect to a grammar is equivalent with querying the goal that corresponds to the top node of the derivation tree. Definite clause grammars (DCGs) are based on some Prolog definitions that facilitate the definition of grammars even further (Pereira and Warren 1980). Also the specification of attribute-value grammars and unification grammars is straightforward. In this section we discuss probabilistic extensions of grammar formalisms that are based on the underlying constraint language.

We start with the definition of P-CUF, an extension still very similar to the PCFG formalism, both from the definition and the training point of view. Then we go on to the better fundamented weighted constraint logic and probabilistic constraint logic by Riezler.

2.7.1 Probabilistic CUF

Eisele (1994) defines a probabilistic extension of the Comprehensive Unification Formalism (CUF) (Dörre and Dorna 1993; Dörre et al. 1994). CUF is, like PATR II and ALE, a declarative language for describing natural language.

Eisele argues that CUF definitions can be expressed in a relational way as clauses of the form

$$r \leftarrow q_1 \wedge \ldots \wedge q_n \wedge \phi \tag{2.3}$$

where $r, q_1, \ldots, q_n$ are predicates with distinct variables as arguments, and ϕ a constraint in the underlying constraint language (Prolog, for instance) that relates these variables. He proposes to annotate these clauses with probabilities such that the probabilities of all clauses for the same predicate sum to 1. The probability of a proof, the intuitive counterpart of a derivation, then is the product of the probabilities of the clauses applied in it. Due to the possibility of failure, the probabilities of the possible proofs of a particular goal will, in general, not sum to 1.

In order to train a P-CUF specification on a corpus of queries, the Baum-Welch algorithm is applied. It expects the clauses of the specification to have some initial probabilities attached. Then, in each iteration the clause probabilities are re-estimated by multiplying them by their expected (normalised) frequency of being applied in a proof. Note the similarity to the Inside-Outside algorithm.

Riezler (1996) observes that this approach is correct for the context-free case only. In cases where the independence of clauses of the CUF specification is harmed (the ϕ in 2.3 is non-empty), the estimation of expected clause frequencies no longer results in a valid probabilistic model of CUF.

2.7.2 Weigthed constraint logic

In (Riezler 1996) the syntax and semantics of a weighted extension of constraint logic are given. The paper defines constraint logic, and definite clause specification over such logics formally. Then the quantitative extension of definite clause specifications is defined formally as an annotation of definite clauses:

$$r \leftarrow_f q_1 \wedge \ldots \wedge q_n \wedge \phi$$

where $f \in (0, 1]$. The weight of a proof is defined as the minimum of the weights assigned to the clauses used in the proof. If more proofs exists the maximum weight applies. So, the usual **and**/**or** trees for the representation of all possible proofs, have become **min**/**max** trees.

The application Riezler has in mind is the specification of weighted definite clause grammars. Seen from this perspective, the **min**/**max** interpretation of proofs is strongly related to the **min**/**max** interpretation of derivation trees in fuzzy grammar (Lee and Zadeh 1969).

Two possible interpretations of the weighted extension are discussed. One possible interpretation is that the weight of a proof tree gives the "degree of grammaticality"; clearly this interpretation corresponds to the notion "degree of membership" in fuzzy set algebra (Zadeh 1965). And the other, more common, interpretation is to consider the weights as preference values. However, the paper lacks an account of the parameter estimation problem. The author is aware of this problem: "Regarding the interest in computational linguistics problems such as ambiguity resolution, however, a necessary prerequisite for a more sophisticated semantics for probabilistically interpreted quantitative CLP is the development of a probabilistic model for CLP which allows for correct

parameter estimation from empirical data." (p. 364). The work by Riezler treated in the next section repairs this defect by defining a probabilistic model based on maximum entropy models.

2.7.3 Probabilistic constraint logic

In (Riezler 1998c) and (Riezler 1998b) a probabilistic extension of constraint logic programming based on the MaxEnt method is presented. The idea is to induce a probability distribution that gives the probability of a proof conditional on some query. Riezler advocates unsupervised learning of probabilistic constraint logic programs (CLPs), and introduces a revised version of Improved Iterative Scaling for incomplete data; it is called Iterative Maximization (IM) . The data is incomplete because, if more than one proof for a query is possible, their frequency is not known. The IM algorithm accounts for this incompleteness.

The event space for IM is defined as follows. Y denotes the set of observed queries, and X a set of proofs. Let $c(y)$ denote the number of times y was observed, and *proofs*(y) the set of proofs for y w.r.t. program P. Each $x \in X$ is a proof of some query in $y \in Y$. It is unknown how often a proof x of y occurs.

Now two probability distributions are defined, one on X and one on Y, denoted by p_X and p_Y resp. p_X is a MaxEnt distribution, and p_Y depends on p_X as follows:

$$p_Y(y) = \sum_{x \in proofs(y)} p_X(x)$$

Given a program P, the IM algorithm determines the parameters of p_X such that the likelihood of observing Y is maximised:

$$L(\alpha) = \alpha \in \mathbf{R}^k \prod_{y \in Y} p_Y(y)^{c(y)}$$

The IM algorithm is given by the iterative step

$$\alpha^{(n+1)} = M(\alpha^{(n)})$$

where

$$\begin{aligned} M(\alpha) &= \hat{\gamma} + \alpha \\ \hat{\gamma} &= \arg\max_{\gamma \in \mathbf{R}^k} A(\gamma, \alpha) \\ A(\gamma, \alpha) &= \sum_{y \in Y} (1 + p(x|y)[\gamma.f] - p_X[\sum_i \overline{f}_i e^{\gamma_i f_\#}]) \\ p(x|y) &= \frac{p_\lambda(x)}{\sum_{x \in proofs(y)} p_X(x)} \end{aligned}$$

The auxiliary function A is a lower bound on the gain in maximum likelihood if the parameters α are adapted by γ, i.e. on $L(\gamma+\alpha)-L(\alpha)$. This re-estimation is indeed very similar to that of the IIS algorithm. In both algorithms an auxiliary function that serves as lower bound on the difference in likelihood is maximised.

2.8 Summary

In this chapter we have given a survey of the literature on probabilistic extensions of grammar formalisms. In table 2.1 we have summarised the formalisms we considered together with their location in this chapter and the relevant literature reference. Also we have indicated what methods exists for supervised and unsupervised learning of the parameters of the probability distributions.

We distinguished 5 types of probabilistic extensions. Two of them, *CFG/rule-based* and *CFG/action-based*, are strongly related to context-free grammar and PCFG. Probabilistic extensions of *Tree-based* formalisms are more powerful than the PCFG formalism. In general, training and parsing are more difficult and computationally more expensive. In practice, restrictions and approximations make these formalisms computationally feasible.

We saw extensions of *Unification-based* grammar formalisms based on rule frequencies, and a theoretically better motivated approach based on the maximum entropy method. The probabilistic extension of *Constraint Logic* offers a very general approach to the extension of unification grammars.

formalism	**sec.**	**reference**	**unsup.**	**sup.**
		CFG/rule-based		
PCFG	2.2	(Grenander 1967)	I/O	ERF
WRSG	2.3.1	(op den Akker and ter Doest 1994)	I/O	-
Salomaa's PCFG	2.3.3	(Salomaa 1969)	-	-
HBG	2.3.2	(Black et al. 1992)	-	decision tree
		CFG/action-based		
GLR	2.4.1	(Wright and Wrigley 1991)	-	ERF
LR-parsing UG	2.6.2	(Briscoe and Carroll 1993)	-	action/manual
left-corner	2.4.2	(Manning and Carpenter 1997)	-	action
		Tree-based		
SLTAG	2.5.1	(Schabes 1992)	I/O	-
SLTIG	2.5.1	(Schabes and Waters 1993)	I/O	-
STSG	2.5.2	(Bod 1998)	-	ERF
		Unification-based		
PFG	2.6.3	(Goodman 1997)	I/O	-
SAVG	2.6.4	(Abney 1997)	-	MaxEnt
LR-parsing UG	2.6.2	(Briscoe and Carroll 1993)	-	action/manual
P-ALE	2.6.1	(Brew 1995)	-	ERF
		Constraint Logic		
P-CUF	2.7.1	(Eisele 1994)	-	ERF
Quantitative -	2.7.2	(Riezler 1996)	-	-
Probabilistic -	2.7.3	(Riezler 1998a)	MaxEnt	-

Table 2.1: Overview of probabilistic extensions of grammar formalisms. I/O is an abbreviation of Inside-Outside algorithm, and ERF of expected rule frequency.

Chapter 3
Unification-based Parsing

Unification grammars, often referred to as constraint-based grammars, provide a popular way of modelling syntax in computational linguistics. They are more powerful than context-free grammars, flexible, and can be specified declaratively. Moreover, description logics can be defined easily to show nice properties like monotonicity, soundness, and completeness (Kasper and Rounds 1986; Carpenter 1992).

In this chapter we give formal definitions of unification grammar and left-corner chart parsing with unification grammars. The parser we used for our experiments (see chapter 6) closely conforms to the definition in this chapter.

First we will define type hierarchies and typed feature structures which are rooted labelled directed acyclic graphs. Then we discuss how the domain of feature structures is structured by the subsumption relation, a relation similar in nature to the subset relation in set theory. Furthermore we define unification of feature structures. Then we define multi-rooted feature structures to enable a convenient definition of unification grammars. Multi-rooted feature structures are feature structures that may have multiple roots (Sikkel 1997; Wintner 1997). We express derivations in terms of multi-rooted feature structures, and define the language generated by a unification grammar. We discuss the chart parsing algorithm, and present a *parsing schema* (Sikkel 1997) for left-corner parsing of unification grammar based on multi-rooted feature structures.

In the first section we define feature structures. Section 3.2 discusses subsumption and unification, and algebraic properties of the domain of feature structures. In section 3.3 we define multi-rooted feature structures, the subsumption relation, concatenation, and some other convenient operations. In section 3.4 we define unification grammar in terms of multi-rooted feature structures. Section 3.5 presents a parsing schema for left-corner chart parsing of unification grammar. In section 3.6 we discuss possibilities for the probabilistic extension of unification grammars.

For the definition of ordinary feature structure we used (Shieber 1992; Shieber 1986; Carpenter 1992). For the theory of orders and lattices we consulted (Davey and Priestley 1990) and (Wechler 1992). We obtained the idea for the definition of unification grammar as a set of multi-rooted feature structures from (Sikkel 1997) and (Wintner 1997). The definition of chart parsing and the left-corner parsing schema is based on (Sikkel 1997).

3.1 Feature structures

Feature structures are directed a-cyclic graphs. Their edges are labelled with features, and their nodes with types. Types are partially ordered in a so-called type hierarchy. Unlike Carpenter (1992), who defines types as sets of concepts, we adopt a rather simple definition of types: our types are atomic. The set of all types is denoted by Type. A type hierarchy is a bounded complete partial order on Type.

Definition 3.1 (type hierarchy)
A type hierarchy is a partially ordered set of types $(\mathsf{Type}, \sqsubseteq)$ such that

- it is bounded complete: for every subset $T \subseteq \mathsf{Type}$, a unique least upper bound, $\sqcup T$, exists;
- types $\top, \bot \in \mathsf{Type}$ exist such that $\sqcup\mathsf{Type} = \top, \sqcap\mathsf{Type} = \bot$, resp.

If types $\sigma, \tau \in \mathsf{Type}$ $\sigma \sqsubseteq \tau$, we say that σ *subsumes* τ, i.e. σ is more general than τ. It can be shown that $(\mathsf{Type}, \sqcup, \sqcap)$ is a bounded complete lattice.

Feature structures are directed acyclic graphs (DAGs). Their nodes are labelled are labelled with types from Type. The edges are labelled with feature labels. The set of all feature labels is denoted by Feat. Feature labels (or features) are atomic as well, and we assume the intersection of Feat and Type to be empty.

Definition 3.2 (feature structure)
A *feature structure* (FS) over a set of features Feat and a set of types Type is a rooted, connected, labelled, DAG (N, n_0, δ, θ) where

- N is a finite set of nodes,
- $n_0 \in N$ is the root of the graph,
- $\delta : N \times \mathsf{Feat} \rightarrow N$ a partial function,
- $\theta : N \rightarrow \mathsf{Type}$ a function that assigns types to nodes.

Note that the type assignment function θ is total: every node has a type from the type hierarchy assigned. The set of all feature structures is denoted by FS.

A *path* is a string of feature labels. The *value* of a path is a type if the path points to a *final node* (which is a node that has no edges leaving it). The value of a path is a feature structure if it points to a node that is not a final node. Apparently, multiple paths may point to the same node: this is called *coreference*, two paths *share* the same value, either a type or a feature structure.

We consider an example of an FS now.

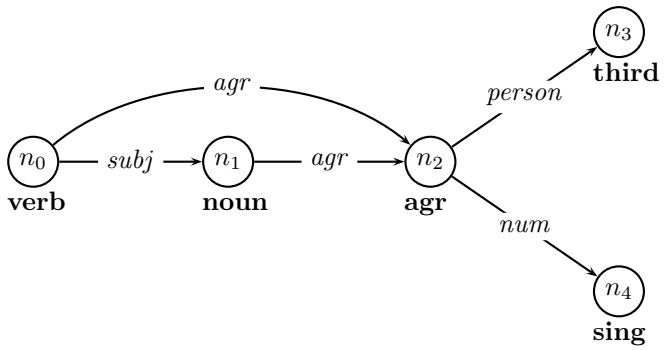


Figure 3.1: Directed labelled graph for the typed feature structure of example 3.3.

Example 3.3 (feature structure)
The feature $f = (N, n_0, \delta, \theta)$ structure depicted in figure 3.1 is written formally as

$$\begin{aligned} N =& \{n_0, \ldots, n_4\}, \\ \delta =& \{((n_0, \mathit{subj}), n_1), ((n_1, \mathit{agr}), n_2), ((n_0, \mathit{agr}), n_2), \\ & ((n_2, \mathit{person}), n_3), ((n_2, \mathit{num}), n_4)\}, \\ \theta =& \{(n_0, \mathbf{verb}), (n_1, \mathbf{noun}), (n_2, \mathbf{agr}), (n_3, \mathbf{third}), (n_4, \mathbf{sing})\}. \end{aligned}$$

For convenience, feature structures are written in the attribute value matrix (AVM) notation:

$$f = \begin{bmatrix} \mathbf{verb} & \\ \mathit{agr}\colon \boxed{1} & \begin{bmatrix} \mathbf{agr} \\ \mathit{num}\colon \mathbf{sing} \\ \mathit{person}\colon \mathbf{third} \end{bmatrix} \\ \mathit{subj}\colon & \begin{bmatrix} \mathbf{noun} \\ \mathit{agr}\colon \boxed{1} \end{bmatrix} \end{bmatrix}$$

We adopt the convention of writing the types at the top of the left square bracket. Alternatively, we can write the feature structure by means of path equations. A *path equation* is a partial specification of a feature structure; a path equation is of the form $p = q$ where p is a path and q is either a type or a path. A set of path equation may define a feature structure, but not every set of equations defines a feature structure (think of cyclicity). Below we have given a set of path equations which specifies f.

$$\begin{aligned} <f> &= \textbf{verb} \\ <f\ subj> &= \textbf{noun} \\ <f\ subj\ agr> &= \textbf{agr} \\ <f\ subj\ agr\ person> &= \textbf{third} \\ <f\ subj\ agr\ number> &= \textbf{sing} \\ <f\ agr> &= <f\ subj\ agr> \end{aligned}$$

Note that this is not the only set of equations which specifies f.

3.2 Subsumption and unification

Subsumption is a relation on FSs that defines a partial ordering in terms of information content. A feature structure is subsumed by another one if it contains more information. Subsumption is defined syntactically in terms of feature graph morphisms. A feature graph morphism is a mapping of the nodes of one feature structure to the nodes of another.

Definition 3.4 (feature graph morphism)
A *feature graph morphism* g is a *total* function that, given two feature graphs (N, n_0, δ, θ) and $(N', n'_0, \delta', \theta')$, maps each node in N to a node in N'.

A feature structure may contain another feature structure in the following way:

Definition 3.5 (substructure)
An FS $f_1 = (N, n_0, \delta, \theta)$ is contained in another FS $f_2 = (N', n'_0, \delta', \theta')$ as a *substructure*, written as $f_1 \leq f_2$, if and only if there exists a feature graph morphism $g : N \rightarrow N'$ such that:

1. for all $n \in N$ and for all $l \in \mathsf{Feat}$, if $\delta(n, l)$ is defined then $\delta'(g(n), l)$ should also be defined, and $g(\delta(n, l)) = \delta'(g(n), l)$;
2. $\forall n \in N : \theta(n) \sqsubseteq \theta'(g(n))$.

The substructure relation induces the lub and glb operators $\vee$, $\wedge$ on FS, and $(\mathsf{FS}, \vee, \wedge)$ is a lattice. We define subsumption as the substructure relation extended with the requirement that root nodes should be related by the morphism.

Definition 3.6 (subsumption)
An FS $f_1 = (N, n_0, \delta, \theta)$ subsumes another FS $f_2 = (N', n'_0, \delta', \theta')$, written as $f_1 \sqsubseteq f_2$, if and only if there exists a morphism $g : N \rightarrow N'$ such that $f_1 \leq f_2$, and $g(n_0) = n'_0$.

Feature structures that subsume each other mutually, only differ in their node labels. In (Carpenter 1992) such feature structures are called alphabetic variants. We will ignore node labels, and consider feature structures that subsume each other as equal.

Example 3.7 (subsumption)
Consider the feature structure g:

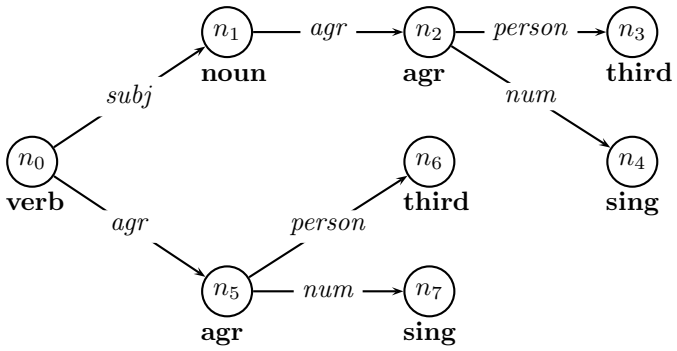


Figure 3.2: Graph representation of the feature structure of example 3.7

$$g = \begin{bmatrix} \textbf{verb} & \\ \textit{agr}: & \begin{bmatrix} \textbf{agr} & \\ \textit{num}: & \textbf{sing} \\ \textit{person}: & \textbf{third} \end{bmatrix} \\ \textit{subj}: & \begin{bmatrix} \textbf{noun} & \\ \textit{agr}: & \begin{bmatrix} \textit{num}: & \textbf{sing} \\ \textit{person}: & \textbf{third} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

In formal notation this is feature structure $g = (N, n_0, \delta, \theta)$ (see also figure 3.2):

$$\begin{aligned} N =& \{n_0, \ldots, n_7\}, \\ \delta =& \{((n_0, \textit{subj}), n_1), ((n_1, \textit{agr}), n_2), ((n_2, \textit{person}), n_3), ((n_2, \textit{num}), n_4), \\ & ((n_0, \textit{agr}), n_5), ((n_5, \textit{person}), n_6), ((n_5, \textit{num}), n_7)\}, \\ \theta =& \{(n_0, \textbf{verb}), (n_1, \textbf{noun}), (n_2, \textbf{agr}), (n_3, \textbf{third}), (n_4, \textbf{sing}), (n_5, \textbf{agr}), \\ & (n_6, \textbf{third}), (n_7, \textbf{sing})\}. \end{aligned}$$

The feature structure f of example 3.3 and g seem to state the same facts about agreement features of the verb and that of the subject. The difference is that f uses a coreference for the agreement information and that g carries a copy of the agreement subgraph (or substructure). We can find that $g \sqsubseteq f$ with the following morphism:

$$\begin{aligned} & \{(n_0, n_0), (n_1, n_1), (n_2, n_2), (n_3, n_3), (n_4, n_4), \\ & (n_5, n_2), (n_6, n_3), (n_7, n_4)\}. \end{aligned}$$

But $f \not\sqsubseteq g$, as mapping node n_3 is a problem here: should it map to n_1 or n_6? Intuitively, this is exactly what we want, since f not only says that the agreement features of the verb and that of the subject are equal, but that they are equal because they are shared.

The least upper bound operator $\sqcup$ and greatest lower bound operator $\sqcap$ are defined as usual. Unification is defined as the least upper bound.

Definition 3.8 (lub, join)
The *least upper bound* (or *lub*) of two feature structures f and g with respect to subsumption is a feature structure $f \sqcup g$ such that

(i) $f \sqsubseteq f \sqcup g$ and $g \sqsubseteq f \sqcup g$, and

(ii) $\forall h \in \mathsf{FS} : f \sqsubseteq h \wedge g \sqsubseteq h \Rightarrow f \sqcup g \sqsubseteq h$

The lub is also called *join*.

Now we will see what the algebraic structure of feature structures under the lub operator is like.

Theorem 3.9 ($(\mathsf{FS}, \sqcup)$ is a semilattice)
$(\mathsf{FS}, \sqcup)$ is a semilattice (i.e. idempotent and commutative semigroup).

We need to show that $(\mathsf{FS}, \sqcup)$ is an idempotent and commutative semigroup. That $(\mathsf{FS}, \sqcup)$ is a semigroup follows from associativity of $\sqcup$. A proof of associativity, commutativity, and idempotency is straightforward. For completeness, we give the definition of the greatest lower bound of two feature structures.

Definition 3.10 (glb, meet)
The *greatest lower bound* (or *glb*) of two feature structures f and g with respect to subsumption is a feature structure $f \sqcap g$ such that

(i) $f \sqcap g \sqsubseteq f$ and $f \sqcap g \sqsubseteq g$

(ii) $\forall h \in \mathsf{FS} : h \sqsubseteq f \wedge h \sqsubseteq g \Rightarrow h \sqsubseteq f \sqcap g$.

The glb is also called *meet*.

Furthermore it can be shown that $(\mathsf{FS}, \sqcap)$ is a *bounded complete semilattice*, and that $(\mathsf{FS}, \sqcup, \sqcap)$ is a lattice.

For the definition of unification we need to define the union of two next-node functions with disjoint domains. Given next-node functions δ and δ' with disjoint domains, the union denoted by $\delta \cup \delta'$ is defined as

$$\delta \cup \delta'(n,l) = \begin{cases} \delta(n,l), \text{ if } \delta(n,l) \text{ is defined,} \\ \delta'(n,l), \text{ if } \delta'(n,l) \text{ is defined,} \\ \text{undefined, otherwise} \end{cases}$$

The union of two type functions with disjoint domains is defined similarly. We now define the unification of two feature structures.

Definition 3.11 (unification)
The unification of two FSs $f = (N_f, n_{0_f}, \delta_f, \theta_f)$ and $g = (N_g, n_{0_g}, \delta_g, \theta_g)$ having $N_f \cap N_g = \emptyset$, is defined by the the feature structure $h = (N_h, n_{0_h}, \delta_h, \theta_h)$ where

- $N_h = (N_f \cup N_g)/_\sim$ where $\sim$ is the least equivalence relation on nodes $N_f \cup N_g$ such that
  - $n_{0_f} \sim n_{0_g}$; and
  - $\delta_f(n, f) \sim \delta(n', f)$ if both are defined and $n \sim n'$.
- $n_{0_h} = [n_{0_f}]_\sim = [n_{0_g}]_\sim$;
- $\delta_h(n, [f]_\sim) = \begin{cases} [\delta_f \cup \delta_g(n, f)]_\sim, & \text{if } (\delta_f \cup \delta_g)(n, f) \text{ is defined} \\ \text{undefined}, & \text{otherwise} \end{cases}$;
- $\theta_h([n]_\sim) = \bigsqcup\{\theta_f \cup \theta_g(n') | n' \in [n]_\sim\}$.

Unification fails if θ_h results in $\top$ for some $[n]_\sim$.

It can be shown that, given two FSs f and g, unification as defined above results in the least upper bound $f \sqcup g$. We give an outline of the proof. First show that it results in an FS. This can easily be established with the definition. Then show that unification results in an upper bound by constructing morphisms from the argument FSs into the unified FS. Finally show that it is the least upper bound by proving that the unification subsumes arbitrary upper bounds of the argument FSs. For a formal proof we refer to (Carpenter 1992).

3.3 Multi-rooted feature structures

A multi-rooted feature structure (MRFS) is a feature structure that may have multiple root nodes. An index function regulates the ordering of these nodes.

Definition 3.12 (MRFS)
A multi-rooted feature structure (MRFS) is a five tuple $(N, N_0, \delta, \theta, I)$ where

- N is a finite set of nodes;
- $N_0 \subseteq N$ a non-empty set of root nodes;
- $\delta : N \times \mathsf{Feat} \to N$ a partial next-node function;
- $\theta : N \to \mathsf{Type}$ a type function;
- $I : [0 \ldots n] \to N_0$ an index function for $n \in \mathbf{N}$.

In the next section we will use MRFSs to define the rules of unification grammars. In this context the index function is best understood as an assignment of feature structures to constituent positions in a grammar rule (counting from left to right starting with the left-hand side of the rule). So, the index function I assigns to each constituent (position) of the rule a root node, i.e. a feature structure that can be reached from this root. The definition allows positions to share a root node. The length of an MRFS f, denoted by $len(f)$, is defined as the largest $n \in \mathbf{N}$ such that $I(n)$ is defined. Note that different positions

in the rule may point to the same root node. The set of all MRFSs is denoted by MRFS. Similar to subsumption for ordinary feature structures we define subsumption for MRFSs.

Definition 3.13 (subsumption of MRFSs)
An MRFS $f = (N, N_0, \delta, \theta, I)$ subsumes another MRFS $g = (N', N_0', \delta', \theta', I')$, denoted by $f \sqsubseteq g$, if $len(f) = len(g)$, and there exists a morphism $h : N \to N'$ such that

(i) for every root $n \in N_0$, a root $n' \in N_0'$ exists such that $h(n) = n'$, and an $i \in \mathbf{N}$ exists such that $I(i) = n$ and $I'(i) = n'$;

(ii) for every node $n \in N$, $\theta(n) \sqsubseteq \theta'(h(n))$;

(iii) for every node $n \in N$ and feature $f \in \mathsf{Feat}$, if $\delta(n, f)$ is defined then $h(\delta(n, f)) = \delta'(h(n), f)$.

Note that in the first requirement we say that each root node should be mapped to a root node in the other MRFS such that they have the same indices.

We define some operations on MRFSs to facilitate the definition of unification grammars in the next section. The first new operation is taking the substructure of an MRFS. A *substructure* of a MRFS w.r.t. a contiguous range of indices is that part of the MRFS that can be reached from the root nodes having these indices. For instance, the substructure $f_{i\ldots j}$ consists of those nodes and edges of f that can be reached from the nodes $I(i)$ through $I(j)$. In addition, we require that the index function is adapted such that for each index $k \in \{i, \ldots, j\}$, $I_{f_{i\ldots j}}(k - i) = I_f(k)$; in other words, the substructure is reindexed such that root nodes again start at index 0. A substructure w.r.t. an index i, denoted by f_i, is in fact an FS, and may, depending on the context, be addressed as such. The following substructure operations are of special interest for the representation of grammar rules as MRFSs:

$$lhs(f) \equiv f_0$$
$$rhs(f) \equiv f_{1\ldots\, len(f)}$$

The second is the unification of an MRFS and an FS. Given an MRFS f and an FS g, we write the unification of f and g w.r.t. to root node i of f as $f \sqcup_i g$. The result of such a unification is f with its i-th node unified with g; note that substructures other that f_i may change as well through coreferences present in f.

The third operation is the concatenation of two MRFSs. The concatenation of MRFSs f and g, written as $f \cdot g$ is a new MRFS such that substructure $(f \cdot g)_{0\ldots\, len(f)} = f$ and $(f \cdot g)_{len(f)+1\ldots\, len(f)+len(g)+1} = g$. Note that this requires the index function of $f \cdot g$ to shift the indices defined in g. Concatenation will never add new coreferences. It is an associative operation, but not commutative.

3.4 Unification grammars

A unification grammar is a finite set of grammar rules. These grammar rules can conveniently be represented by multi-rooted feature structures, i.e. feature structures that may have more than one root node. The description of unification grammar that we give in this section is inspired by the description of unification grammars based on *abstract multi-rooted feature structures* in (Wintner 1997). Also, *composite feature structures* by Sikkel (1997) are similar to our multi-rooted feature structures.

A grammar rule is simply defined as an MRFS. The left-hand side of a rule f is the substructure that can be reached from root node $I(0)$, and the right-hand side is the substructure that can be reached from the nodes $I(1)$ through $I(len(f))$. The definition licenses ϵ-rules. The grammar does not have an explicit context-free backbone. It can be defined though by distinguishing a path of feature labels that points at the type or structure we want to base the backbone on.

Example 3.14 (rule)
In the example grammar rule below, we have represented the index of the root nodes by a number in front of each root node.

$$r = \left[\begin{array}{l} \textbf{phrase\_np} \\ 0 \rightarrow \left[\begin{array}{l} \textbf{np} \\ \textit{head}\colon \boxed{1} \end{array}\right] \\ 1 \rightarrow \left[\begin{array}{l} \textbf{det} \\ \textit{head}\colon \boxed{2} \left[\begin{array}{l} \textbf{syn} \\ \textit{det}\colon \textbf{yes} \end{array}\right] \end{array}\right] \\ 2 \rightarrow \left[\begin{array}{l} \textbf{noun} \\ \textit{head}\colon \boxed{1} \left[\begin{array}{l} \textbf{syn} \\ \textit{modify}\colon \boxed{2} \end{array}\right] \end{array}\right] \end{array}\right]$$

In a formalism like PATR II a similar rule (untyped) would be specified by

```
NP -> DET NOUN
 <NP head> == <NOUN head>
 <DET head det> == yes
 <NOUN head modify> == <DET head>
```

Basically, a unification grammar is a set of MRFSs.

Definition 3.15 (unification grammar)
A *unification grammar* G is a triple (P, S, L) where

- $P \subset \mathsf{MRFS}$ is a finite set of grammar rules;
- $S \subseteq P$ is the set of start productions; each rule in S may be used to start a derivation;
- $L : \mathsf{Words} \rightarrow 2^{\mathsf{FS}}$ the lexicon function that assigns a set of feature structures to each word.

The lexicon function L corresponds to the set of terminals in CFGs, and the set of start productions S corresponds to the start symbol.

A derivation w.r.t. a grammar starts with a start production and ends if no grammar rules can be applied anymore. One MRFS can be derived from another if a substructure can be rewritten using a grammar rule such that it equals the other MRFS.

Definition 3.16 (derivation relation)
Given a grammar $G = (P, S, L)$,

- an MRFS g can be derived from an MRFS f, denoted by $f \rightarrow g$, if indices i, j, k exists, and a rule $r \in P$, such that
  (i) $lhs(r) \sqsubseteq f_i$;
  (ii) $rhs(r) \sqsubseteq g_{j\ldots k}$;
  (iii) if $r' = r \sqcup_0 f_i$ and $f' = f \sqcup_i lhs(r)$, then $g = f'_{0\ldots i-1} \cdot rhs(r') \cdot f'_{i+1\ldots\, len(f)}$.
- a word $w \in \mathsf{Words}$ can be derived from an FS f, denoted by $f \rightarrow w$, if a feature structure $g \in L(w)$ exists such that $g \sqsubseteq f$. An MRFS f derives a sequence of words $w_1 \ldots w_{len(f)}$ if for each $i \in \{0 \ldots len(f) - 1\}$ f_i derives w_i.

Requirements (i) and (ii) in the first clause of the definition makes sure that rule r may have been the rule that was used to rewrite f into g. Requirement (iii) is there to prevent g of being more specific than the rule allows. Also it expresses how coreferences originally in f are preserved and propagated. If f_i is unified with the LHS of rule r, then information may be propagated to the rest of f, and to the RHS or r. Replacing f_i by the RHS of r while preserving shared nodes, results in a new f in which the RHS of r may share information with f. The second clause defines the derivation of words from the substructures of an MRFS. It says that a word can derived from an FS if it is at least as specific as one of the FSs that L assigns to the word. As usual $\rightarrow^+$ denotes the transitive closure of the derivation relation, and $\rightarrow^*$ the reflexive, transitive closure.

Now we can define the language generated by a unification grammar. It is simply defined as those sentences that can be derived from the grammar starting with a start production.

Definition 3.17 (language)
The language generated by a unification grammar $G = (P, S, L)$ is defined by the set

$$L(G) = \{x \in \mathsf{Words}^* | \exists f \in S : f \rightarrow^+ x\}$$

3.5 Parsing

A parsing algorithm serves to find the structure of a sentence with respect to a grammar. In the case of unification grammars this means to find the grammar

```
proc chart-parser(Tschema s)
  begin
    create initial chart and agenda
    while agenda is not empty do
          delete current item from agenda
          foreach item that can be deduced according to s
                from current with other items in chart do
            if item is neither in chart nor in agenda
              then add item to agenda
            fi
          od
    od
  end
.
```

Figure 3.3: Chart parser in pseudo pascal.

rules that were applied in the derivation(s) of the sentence. In this section we first define chart parsing abstracted from the form and meaning of the items, then we present a parsing schema for LC parsing of unification grammar. Our notion of a parsing schema is from (Sikkel 1997); similar ideas can be found in (Shieber et al. 1995).

We assume the parsing algorithm to operate on *items*. An item is a partial specification of a parse tree, or better, an equivalence class of parse trees that share some relevant properties. For instance, given a sentence $w_1 \dots w_4$, an item $[NP, 2, 4]$ represents all parse trees that have an NP at the root and that generate $w_{3\dots4}$. The syntax of items is free and can be chosen such that particular sets of trees can be conveniently represented. A set of items resulting from parsing is a distributed representation of all possible parses. This means that each possible parse can be re-constructed from the set. We refer the reader to chapter 4 of (Sikkel 1997) for a more detailed treatment of the theory of items and item-based parsing schemata. A *parsing schema* is a set of deduction rules which specifies what new items may be derived from old items.

3.5.1 Chart parsing

Given a parsing schema that describes how new items should be created from old ones, we can easily describe a chart parser. A chart parser is a parser that maintains two stores for items defined by a parsing schema: an *agenda* and a *chart*. Initially, the agenda contains axiomatic items, i.e. items that can be deduced without antecedent, in the chart are those items that represent the words of the sentence. After initialisation of chart and agenda the chart parser

picks an arbitrary item from the agenda, and adds all items not already in chart or agenda that can be deduced by the deduction rules of the parsing schema. Then it picks the next item from the agenda, and so forth. In pseudo pascal the chart parser is given in figure 3.3.

3.5.2 Left-corner parsing

Left-corner parsing was introduced by (Rosenkrantz and Lewis-II 1970) as a deterministic left-to-right pushdown stack machine. In left-corner parsing the prediction of local trees is based on the left-corner relation.

Usually the left-corner relation is defined on the nonterminals of the grammar. We define the left-corner relation directly on the MRFSs that make up the rules of the grammar. The left-corner (LC) of a rule r is the substructure r_1, denoted by $lc(r)$. Since we will define the left-corner on the context-free backbone of the grammar, we introduce the function $cat : \mathsf{FS} \rightarrow \mathsf{FS}$ that results in the value (a type or an FS) that should be used in the left-corner relation.

The left-corner relation on grammar rules is defined as follows.

Definition 3.18 (left-corner relation)
Given a unification grammar $G = (P, S, L)$, two rules $f, g \in P$ stand in left-corner relation, denoted by $f >_l g$, if $cat(lc(f)) = cat(lhs(g))$.

The reflexive, transitive closure is denoted by $>_l^*$. Note that we defined the left-corner on the category features of the LHS and the LC. Clearly, we have some freedom here: the category feature is configurable. Even multiple features can be defined, or we could relax the LC relation such that the LHS of the second rule should be more specific a feature structure than the LC of the first rule. But in general, the left-corner relation is restricted to the context-free backbone of the grammar.

Before we define the types of items we need for describing the parsing, we define so-called dotted MRFSs.

Definition 3.19 (DMRFS)
A *dotted MRFS* (DMRFS) is a six-tuple $(N, N_0, \delta, \theta, I, i)$ where $(N, N_0, \delta, \theta, I)$ is a MRFS (see definition 3.12), and $i \in \{0, \ldots, len(f)\}$, called the *dot* of the DMRFS.

The set of all DMRFSs is denoted by DMRFS. For convenience we define the following functions on DMRFSs:

- a function $dot : \mathsf{DMRFS} \rightarrow \mathbf{N}$ that returns the position of the dot of a DMRFS;
- a function $shift : \mathsf{DMRFS} \rightarrow \mathsf{DMRFS}$ that takes a DMRFS, and returns it with the dot shifted one symbol to the right;
- a function $compl : \mathsf{DMRFS} \rightarrow \{\mathbf{true}, \mathbf{false}\}$ that returns **true** if its argument DMRFS is completely recognised; given DMRFS f, $compl(f)$ returns **true** if $len(f) = dot(f)$, and **false** otherwise.

We will employ DMRFSs for the representation of grammar rules of which some part has been recognised in parsing. Given some rule $f \in \mathsf{DMRFS}$, if $dot(f)$ is 0, then this means that nothing of the rule has been recognised, if $dot(f)$ equals $len(f)$ the complete RHS of the rule has been recognised. In general if $dot(f) = i$, then substructure $f_{1\ldots i}$ has been recognised. Given a set of MRFSs P, the set of possible DMRFSs is given by $\mathsf{DMRFS}(P)$.

A parsing schema consists of a definition of the domain of items and a set of deduction rules specifying what new items may be introduced from old ones. In the case of unification we need to specify in addition what should happen to the feature structures of the items. A parsing schema is not a parsing algorithm. It only defines how items are constructed. In general, a parsing schema is not deterministic. A parsing algorithm like chart parsing should decide on the order in which it applies the deductions.

In left-corner parsing three types of items are needed: goal items, left-corner items, and initial items. Given a grammar $G = (P, S, L)$ and a sentence $w = w_1 \ldots w_n$ these are defined as follows:

initial items are of the form $[\alpha, i, i+1]$, $0 \leq i < n, \alpha \in L(w_i)$;

left-corner items are of the form $[f;\ g, i, j]$ where $f \in P, g \in \mathsf{DMRFS}(P), 0 \leq i \leq j \leq n$; $[f;\ g, i, j]_\alpha$ implies that

- $[i, f]$ is set as a goal;
- $f >_l^* g$;
- $g_{1\ldots\, dot(g)} \rightarrow^* w_i \ldots w_j$;

goal items are of the form $[i, f]_\alpha$, where $f \in P$, and $\alpha \in \mathsf{MRFS}$; $[i, f]_\alpha$ means that at position i rule f is expected to derive (part of) the sentence.

Summarising, the items involved in left-corner parsing are given by:

$$I_{MRFS\text{-}LC} = I_{Init} \cup I_{Pred} \cup I_{LC(f)} \cup I_{LC(\epsilon)}$$

where

$$\begin{aligned}
I_{Init} =& \{[\alpha, i, i+1] | \exists u \in \mathsf{Words} : \alpha \in L(u)\},\\
I_{Pred} =& \{[i, f] | f \in \mathsf{DMRFS}(P), i \geq 0\},\\
I_{LC(f)} =& \{[f;\ g, i, j] | f, g \in \mathsf{DMRFS}(P) : f >_l^* g, len(g) > 0, 0 \leq i \leq j\},\\
I_{LC(\epsilon)} =& \{[f;\ g, i, i] | f, g \in \mathsf{DMRFS}(P) : f >_l^* g, len(g) = 0, i \geq 0\}
\end{aligned}$$

To identify uniquely the MRFSs and DMRFSs contained in the items we have attached subscripts to the items where necessary. For instance, to address g in the left-corner item $[f;\ g, i, j]_\alpha$ we say g_α. If we want to address the MRFS of g, we use $\phi(g_\alpha)$. It is important not to confuse $\phi(g_\alpha)$ with g itself. MRFS g is a rule of the grammar. MRFS $\phi(g_\alpha)$ however, is the feature structure associated with g for this particular item. It is the result of previous deductions, and can be propagated to new items to be created. If this item were the result of a

left-corner deduction (see below), then $\phi(g_\alpha)$ would equal g with its left-corner unified with the LHS of f.

We have spread the definition of each set of deductions over three lines; on each first line we have written the actual deduction rule, each second line holds the conditions w.r.t. the dot of the left-corner items involved (if applicable), and conditions on the categories of the MRFSs involved; the third line defines the MRFSs attached to the rules of the items.

Definition 3.20 (left-corner deduction)
Given a unification grammar $G = (P, S, L)$, the left-corner parsing schema is given by the following deduction rules:

$$
\begin{aligned}
D_{Init} = \quad & \{\vdash [0, f] | \\
& \quad f \in S, \\
& \quad \phi(f) = f\}, \\
D_{LC(u)} = \quad & \{[i, f]_\alpha, [\beta, i, i+1] \vdash [f;\ g, i, i+1]_\gamma | \\
& \quad dot(g) = 1, cat(\beta) = cat(lc(g)), \\
& \quad \phi(f_\gamma) = \phi(f_\alpha), \phi(g) = g \sqcup_0 lc(\phi(f_\alpha)) \sqcup_1 \beta\}, \\
D_{LC(g)} = \quad & \{[f;\ g, i, j]_\alpha \vdash [f;\ h, i, j]_\beta | \\
& \quad compl(g), cat(lhs(g)) = cat(lc(h)), dot(h) = 1, \\
& \quad \phi(f_\beta) = \phi(f_\alpha), \phi(h) = h \sqcup_1 lhs(\phi(g))\}, \\
D_{LC(\epsilon)} = \quad & \{[i, f]_\alpha, \vdash [f;\ g, i, i]_\beta | \\
& \quad len(g) = 0, \\
& \quad \phi(f_\beta) = \phi(f_\alpha), \phi(g_\beta) = g \sqcup_0 lc(\phi(f_\alpha))\}, \\
D_{Pred} = \quad & \{[f;\ g, i, j] \vdash [j, h] | \\
& \quad \neg\, compl(g), cat(g_{dot(g)+1}) = cat(lhs(h)), \\
& \quad \phi(h) = h \sqcup_0 \phi(g)_{dot(g)+1}\}, \\
D_{Shift(u)} = \quad & \{[f;\ g, i, j]_\alpha, [\beta, j, j+1] \vdash [f;\ shift(g), i, j+1]_\gamma | \\
& \quad cat(\beta) = cat(g_{dot(g)+1}), \\
& \quad \phi(f_\gamma) = \phi(f_\alpha), \phi(shift(g)) = \phi(g) \sqcup_{dot(g)+1} \beta\}, \\
D_{Shift(g)} = \quad & \{[f;\ g, i, j]_\alpha, [h;\ h', j, k]_\beta \vdash [f;\ shift(g), i, k]_\gamma | \\
& \quad compl(h'), cat(\beta) = cat(g_{dot(g)+1}), \\
& \quad \phi(f_\gamma) = \phi(f_\alpha), \phi(shift(g)) = \phi(g) \sqcup_{dot(g)+1} lhs(\phi(h'))\}, \\
D_{MRFS\text{-}LC} = \quad & D_{Init} \cup D_{LC(u)} \cup D_{LC(g)} \cup D_{LC(\epsilon)} \cup \\
& D_{Pred} \cup D_{Shift(u)} \cup D_{Shift(g)}.
\end{aligned}
$$

Below we give a short explanation of each of these deduction rule sets.

Init These deductions are axiomatic. For each production in S a goal item may be introduced.

LC We distinguish three left-corner deduction sets, for pairs of goal and initial items, for completed left-corner items, and for pairs of goal items and completed left-corner items for ϵ rules:

LC(u) Given a goal item $[i, f]$, and an initial item $[u, i, i+1]$, a new left-corner $[f;\ g, i, i+1]$ may be introduced; the goal of the left-corner is f, and the left-corner g should have a category equal to that of the initial item. The MRFS of the new f equals that of the old f. The MRFS of g is g with its LHS unified with the left-corner of f and its left-corner unified with the FS of the initial item. The unification of g with the left-corner of f allows constraints on f's left-corner to be applied top-down.

LC(g) Given a completed item $[f;\ g, i, j]$ a new left-corner item $[f;\ h, i, j]$ can be introduced. The category of the LHS of g should equal the left-corner of h. Furthermore the MRFS of the new f equals that of the old. The MRFS of h is h unified with the LHS of the MRFS of g. We do not have to unify with the left-corner of f, since we already establish this connection between the goal and the rule in $D_{LC(u)}$ and $D_{LC(\epsilon)}$; by unifying with the LHS of g, this connection is propagated.

LC(ϵ) Given a goal item $[i, f]$, we can derive a new left-corner item $[f;\ g, i, i]$ where g should have $len(g) = 0$. The MRFS of the new f equals the MRFS of the old f. The MRFS of the new g is equal to g unified with the left-corner of the MRFS of f.

Pred Predictive deductions license the introduction of goal items. The premisse is a left-corner item $[f;\ g, i, j]$ that is not complete, the consequent a goal item $[j, h]$ such that the category of the LHS of h equals the category of the next to recognise substructure of g. The MRFS of h is h unified with the MRFS of the next to recognise substructure of g.

Shift We distinguish two shift deduction steps: one shifts over an initial item, and the other over a completed left-corner item:

Shift(u) Shifting a left-corner item $[f;\ g, i, j]$ over an initial item $[\beta, j, j+1]$ results in a new left-corner item $[f;\ g, i, j+1]$ which has its dot one position further to the right. The MRFS of f stays the same. The MRFS of g is unified with the FS of the recognised initial item.

Shift(g) If a left-corner item $[f;\ g, i, j]$ is shifted over a completed left-corner item $[h;\ h, j, k]$, the new left-corner item has its dot shifted, and the MRFS of the item is unified with MRFS of the LHS of h.

The left-corner chart parser for unification grammar can now easily be defined. The chart should be initialised with the set initial items, and the agenda with the items that result from D_{Init}.

3.5.3 Head-corner parsing

An interesting generalisation of LC parsing is head-corner parsing. Head-corner parsing can be described as left-corner parsing with the left-corner relation generalised to the head-corner relation. Whereas the left-corner relation is based

on the left-corner of grammar rules, the head-corner relation is based on head-corners. These are distinguished constituents in the RHS of grammar rules that function as head and are specified by the grammar writer. The idea for head-corner parsing is generally attributed to Kay (1989). Sikkel and op den Akker (1993) developed an item-based head-corner chart-parser.

Several implementations of the head-corner parsing algorithm exist to date. An early implementation was by (Verlinden 1993b; Verlinden 1993a); other implementations are described in (Veldhuijzen van Zanten and op den Akker 1994) and (Moll 1995). In (van Noord 1997) an efficient implementation in Prolog of a head-corner parser for unification grammars is described.

3.6 Probabilistic extensions

Often grammars suffer from over-generation. Extending the formalism with statistics may help to prune parses that are not plausible, and to disambiguate, i.e. select from alternative parses. A probabilistic extension of unification grammar should result in a probability distribution over derivations of the unification grammar. We discuss the possibilities we see for such extensions.

rule-based The extension of unification grammar with statistics is not as simple as it may seem. Of course, attaching a probability to each grammar rule and saying that the probability of a derivation is the product of the probabilities of the rules applied in that derivation, is simple enough. But that is only half the story. The other half should answer the question where these probabilities come from: the problem of statistical inference. For probabilistic context-free grammar this is quite trivial, but for unification grammar it is not. Failure of unification is the source of the problem: the grammar loses probability mass that cannot be accounted for by rule probabilities based on expected rule frequencies.

parser actions Another possibility is to attach probabilities to the actions of the parser. To obtain a more fine-grained probabilistic model, actions can be distinguished by the rules and grammar symbols involved, or some notion of the state the parser is in. For instance, in (Manning and Carpenter 1997) the current left-corner and goal symbols are taken into account. A drawback inherent to these approaches is that the parameters of the probabilistic model are often not portable to other parsing algorithms.

non-generative By non-generative we mean independent of the generation mechanism of the grammar formalism. The probability of a tree or feature structure is not affected by the rules applied in their generation. This means that the probability of a derivation tree or a feature structure can be computed from properties of the feature structures only. If we restrict ourselves to modelling integer-valued (or discrete) properties of feature structures, we can easily define a probability distribution on feature structures. In the next chapter we will show how this can be achieved with maximum entropy modelling.

3.7 Summary

In this chapter we have given a formal definition of feature structures, and discussed some algebraic properties of the domain of feature structures. We defined multi-rooted feature structures, and based a definition of unification grammars on these. Then we defined left-corner parsing for such unification grammars by specifying a parsing scheme. Finally we made some remarks on the possibilities for probabilistic extensions of unification grammars. The point with probabilistic extensions of formalisms, though it is easy to attach weights to operations or objects of the formalism, we have to account for the origin of these weights as well. And concerning parameters of such extensions, we need to explain how these should be determined from observations.

Chapter 4
Maximum Entropy Modelling

In this chapter we discuss the theory of maximum entropy (MaxEnt) modelling. We explain how the principles of MaxEnt modelling and maximum likelihood estimation are related and how their relation can be exploited for the determination of MaxEnt probability distributions.

In general, finding a probabilistic model of a real-world phenomenon involves the definition of a sufficiently precise description language (the sample space). Also a mapping from observations to this language is needed. Furthermore we need to capture the phenomenon in a finite sample of events that we believe to be representative. In computational linguistics such a sample is called a corpus.

We assume that our representation language is discrete, and that events are described by so-called property functions (or properties).[1] In MaxEnt modelling each property has a parameter associated with it. A method for parameter estimation should find values for these parameters such that certain (later to be made precise) constraints are met.

In section 4.1 we present some prerequisites from probability theory. In 4.2 we discuss several entropy measures and explain what they measure, and how they are related; also, we define the likelihood of one distribution w.r.t. another. In section 4.3 it is assumed that we have estimated or know the expectations of functions that model interesting properties of the observed events. We define two sets of probability distributions. In the first set we require the expectations of these functions to be equal to the observed expectations. It is shown that, if the distribution we want should have maximum entropy, it necessarily has an exponential form with one parameter for each function. The second set contains distributions that have this exponential form. We will see that the distribution in this set with maximum likelihood coincides with the maximum entropy distribution in the first set. In section 4.4 we discuss procedures for finding the parameters of maximum likelihood models contained in the second set.

[1] Like Abney (1997) we will not use the term *feature* to prevent confusion with the features used in unification grammar.

4.1 Some probability theory

Probability theory is a theory of uncertainty that seeks to provide mathematical models of situations where the outcome is not deterministic, but depends on uncertain circumstances. The set of all possible outcomes is called the sample space; it is denoted by Ω. The sample space may be countable (i.e. finite), countably infinite, or continuous. In our discussion we assume that the sample space is countable, and that the Kolmogorov axioms hold:

A1 If A and B are events, then so is the *intersection* $A \cap B$, the *union* $A \cup B$, and the *difference* $A - B$.

A2 The *sample space* Ω is an event. We call Ω the *certain* event. The *empty set* $\emptyset$ is an event. We call $\emptyset$ the *impossible* event.

A3 To each event E is assigned a nonnegative real number $P(E)$ which we call the *probability* of event E.

A4 $P(\Omega) = 1$.

A5 If A and B are disjoint, then $P(A \cup B) = P(A) + P(B)$.

A *stochastic variable* maps an element from the sample space to an aspect we are interested in. Formally, a stochastic variable X is a function that maps elements from Ω to a domain $\mathcal{X} = X(\Omega)$. For a stochastic variable X with $X(\Omega) = \{x_1, x_2, \ldots, x_n\}$ we denote the probability that X takes value x_i by

$$P(\{X = x_i\}) = p(x_i) = p_i$$

where $\{X = x_i\}$ is defined as the inverse function $X^{-1}(x_i) = \{\omega \in \Omega | X(\omega) = x_i\}$. We use $\Delta_{\mathcal{X}}$, or shorter Δ if the sample space is clear from the context, to denote the set of all distributions over Ω:

$$\Delta = \{p : \mathcal{X} \to \mathbf{R} | \sum_{x \in \Omega} p(x) = 1, \forall x \in \Omega : p(x) \geq 0\}$$

If we define stochastic variables X and Y by $(X, Y) : \Omega \to \mathcal{X} \times \mathcal{Y}$, and $\{X = x, Y = y\}$ as the inverse function that gives us $\{\omega \in \Omega | X(\omega) = x, Y(\omega) = y\}$, then we can define the *joint probability* of x and y as $p(x, y) = P(\{X = x, Y = y\})$. The *conditional probability* of event x to occur given that event y was observed, denoted by $p(x|y)$, is defined as the probability that both events occur divided by the probability that y occurs:

$$p(x|y) = \frac{P(\{X = x, Y = y\})}{P(\{Y = y\})} = \frac{p(x, y)}{p(y)} = \frac{p(x, y)}{\sum_{x' \in \mathcal{X}} p(x', y)}$$

Bayes' Rule relates the probability of an event x conditional on an event y, i.e. $p(x|y)$, with the probability of event y conditional on event x, i.e. $p(y|x)$, as follows:

$$p(x|y) = \frac{p(x)p(y|x)}{p(y)} \tag{4.1}$$

Bayes' Rule is useful if $p(x|y)$ cannot easily be determined (or estimated), and $p(y|x)$ can.

The *expectation* of a function $f : \mathcal{X} \to \mathbf{R}$ is defined as

$$\begin{aligned} E(f(X)) &= \sum_{\omega \in \Omega} f(X(\omega))P(\{\omega\}) \\ &= \sum_{x \in \mathcal{X}} \sum_{\omega \in \Omega : X(\omega) = x} f(X(\omega))P(\{\omega\}) \\ &= \sum_{x \in \mathcal{X}} f(x)p(x) \end{aligned}$$

If it is clear from the context what stochastic variable is meant we will write $p[f]$ instead of $E(f(X))$.

In the rest of the chapter we assume that we have a sample S of $|S|$ points x_i with $i = 1, \ldots |S|$. The estimation $\tilde{p}$ of p is defined as

$$\tilde{p}(x) = \frac{|\{i|x_i = x\}|}{|S|}$$

We will call $\tilde{p}$ the *reference distribution*.

4.2 Entropy measures

In physics, the word entropy has important physical implications as the amount of "disorder" of a system. In probability theory it gives the/an "amount of uncertainty" of a probability distribution. The Shannon entropy (Shannon 1948) of a probability distribution is defined as

Definition 4.1 (Shannon's entropy)
Shannon's entropy of a probability distribution $p_i = P(\{X = x_i\})$ is

$$H(p) = -\sum_i p_i \log p_i = -p[\log p]$$

To complete our understanding of information entropy, we give some well-known properties of entropy.

- H is strictly concave down[2] on Δ;
- H is continuous;
- H reaches its maximum, $\log(n)$, in the uniform distribution $p(x) = 1/n$; H reaches its minimum, 0, if one event has probability 1, and the other 0.

[2] A function f is *concave up* if $f(\alpha x_0 + (1-\alpha)x_1) \leq \alpha f(x_0) + (1-\alpha)f(x_1)$, for all $0 \leq \alpha \leq 1$. A function f is *concave down* if $-f$ is concave up. A necessary condition for f to be concave up on an interval (a, b): the second derivative f'' has $f''(x) \geq 0$ on the interval. f is concave down if $f''(x) \leq 0$ on the interval. Strict concavity turns $\leq$ in $<$, and $\geq$ in $>$. We prefer the terms concave up and concave down to *convex* and *concave*, resp.

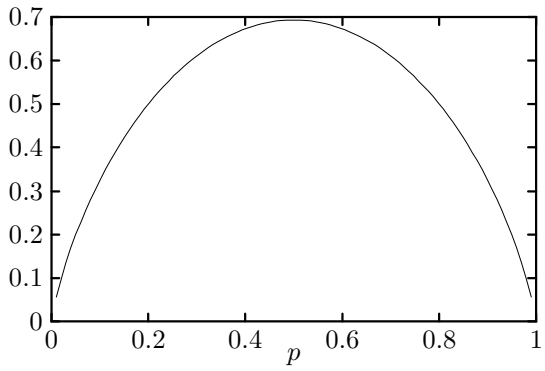


Figure 4.1: Shannon entropy of distribution $(p, 1-p)$.

In figure 4.1 we have depicted the Shannon entropy of the distribution $(p, 1-p)$ as a function of p. It can be seen that the distribution $(0.5, 0.5)$ has maximum entropy.

Suppose we want to find a distribution of the stochastic variable X. Often we think we know something about the distribution of X from observations. For instance, we have estimated the expectation of some function g on X from a random sample, and we require the distribution of X to have the expectation of g equal to the observed estimations. However, this information is not enough to find one unique distribution.

The *Principle of Maximum Entropy* (PME) now says that the distribution to choose is that which maximises entropy, subject to whatever constraints are imposed on the distributions. This does not mean that the maximum entropy distribution is the only distribution that meets the constraints; the principle advises us to take the distribution that has maximum uncertainty subject to the constraints imposed.

The cross entropy is defined as the expectation of a distribution q with respect to another distribution p.

Definition 4.2 (cross-entropy)
The *cross-entropy* $H_c(p, q)$ of a distribution q w.r.t. another distribution p defined over the same space $\mathcal{X}$ is given by

$$H_c(p, q) = p[-\log q] = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

Given a sample S, the *Principle of Maximum Likelihood* says that we should take the distribution that maximises the probability of observing S

$$\prod_{x \in S} p(x)$$

If we compute this product in $\mathcal{X}$ we arrive at

$$\prod_{x \in \mathcal{X}} p(x)^{c(x)}$$

where $c(x)$ is the number of times x occurs in S. Instead of maximising this product of probabilities, we may maximise the sum of log-probabilities

$$\sum_{x \in \mathcal{X}} \log(p(x)^{c(x)}) = \sum_{x \in \mathcal{X}} c(x) \log(p(x)) = |S| \sum_{x \in \mathcal{X}} \tilde{p} \log(p(x))$$

which is propertional to $-H_c(\tilde{p}, p)$.

The *Kullback-Leibler (KL) divergence*, or *relative entropy* measures the distance between a *reference* distribution p and another distribution q.

Definition 4.3 (Kullback-Leibler divergence)
The *Kullback-Leibler divergence* of a distribution q with respect to a *reference* distribution p is defined as

$$\begin{aligned} D(p\|q) = p[\log \frac{p}{q}] &= \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \\ &= p[-\log q] - p[-\log p] \end{aligned}$$

The KL divergence between p and q at point x is the log of the ratio of p to q. We state some important properties without proof:

- $D(p\|q) = H_c(p, q) - H(p) = -L(p, q) - H(p)$;
- for all $p, q \in \Omega$, $D(p\|q) \geq 0$, and $D(p\|q) = 0$ if and only if $p = q$;
- $D(p\|q)$ is strictly concave up in p and q separately; this property ensures that a distribution $\hat{q}$ exists such that $D(p\|\hat{q})$ is minimal.

Since the KL divergence does not satisfy the triangle inequality and is not symmetric, it is not a metric[3].

Example 4.4 (Tree-banks and PCFGs)
In table 4.1 we have given an sample space Ω_T of syntax trees (tree-bank) with their observed frequencies and reference distribution $\tilde{p}$. It is common practice

[3]Remember that for a measure $m : X \times X \to \mathbf{R}$ to be a metric it should have the following properties:

- $m(x, y) \geq 0$, and $m(x, y) = 0 \Leftrightarrow x = y$, for all $x, y \in X$;
- $m(x, y) = m(y, x)$, for all $x, y \in X$;
- $m(x, y) \leq m(x, z) + m(z, y)$, for all $x, y, z \in X$

Ω_T	freq.	$\tilde{p}$
S(NP(DET N))	4	1/3
S(VP(WHNP(WH NP(N)) V) PN)	2	1/6
S(PN VP(V NP(DET N)))	2	1/6
S(NP(NUM N) PUNCT)	3	1/4
S(NP(N))	1	1/12

Table 4.1: A tree-bank.

P	freq.	$\tilde{p}'$
S(NP)	5	5/12
S(NP PUNCT)	3	1/4
S(VP PN)	2	1/6
S(PN VP)	2	1/6
NP(DET N)	6	1/2
NP(NUM N)	3	1/4
NP(N)	3	1/4
WHNP(WH NP)	2	1
VP(WHNP V)	2	1/2
VP(V NP)	2	1/2

Table 4.2: The rules applied in the tree-bank.

to estimate the parameters of a PCFG using ML Estimation. ML estimation finds the rule probabilities that maximise the probability of observing the tree-bank. If N_A is the number of times nonterminal A is rewritten and $N_{A\to\alpha}$ is the number of times A is rewritten into α then $N_{A\to\alpha}/N_A$ is an ML estimator for the probability of rule $A \to \alpha$. In table 4.2 we have given the grammar rules and the ML estimates of their probabilities. It is important to notice that the distribution that is induced by the PCFG on Ω_T is related to $\tilde{p}$ through the maximalisation of the likelihood $L(\tilde{p}, p)$ under the constraint that probabilities for rules with the same LHS sum to one; it does not maximise entropy. In table 4.3 we have given the probability distribution induced by the PCFG on the tree-bank. We see that the probabilities p assigned by the PCFG are not equal to the observed probabilities. Furthermore they do not sum to one over the tree-bank. This is the general picture of PCFG-induced probability distributions. They lose probability mass in trees that can be generated by the grammar, but are not in the tree-bank. Or, formulated differently, PCFGs suffer from their pre-occupation of summing the probabilities of their trees to one. However, in the limit of the size of the tree-bank the probability distribution induced by the PCFG inferred from that tree-bank will converge to the observed distribution. The likelihood is $L(\tilde{p}, p) \approx -2.58$, entropy of $\tilde{p}$ is $H(\tilde{p}) \approx 1.52$, and $D(\tilde{p}\|p) = -L(\tilde{p}, p) - H(\tilde{p}) \approx 1.06$. In example 4.8 where we approximate a different

Ω_T	$\tilde{p}$	p
`S(NP(DET N))`	1/3	5/24
`S(VP(WHNP(WH NP(N)) V) PN)`	1/6	1/48
`S(PN VP(V NP(DET N)))`	1/6	1/24
`S(NP(NUM N) PUNCT)`	1/4	1/16
`S(NP(N))`	1/12	5/48

Table 4.3: A PCFG induced from the tree-bank.

type of probabilistic model for the tree-bank, we will see that this value for KL divergence is very large.

4.3 The maximum entropy method

In this section we explain how the principle of maximum entropy can be applied to find a probabilistic model under circumstances where information is sparse. We will see that the exponential form of *maximum entropy* distributions is an immediate consequence of the introduction of Lagrange multipliers. Further, we define two sets of probability distributions, one set containing so-called maximum entropy distributions, and the other containing distributions that meet constraints with respect to our observed data. We will see that maximum likelihood optimisation in the former set results in the same distribution as maximum entropy optimisation in the latter.

Assume we have a sample S of which only know expectations of property functions, i.e. functions that characterise our data, $f_i : \mathcal{X} \to \mathbf{N}$, $i = 1 \dots k$. We denote these expectations by $\tilde{p}[f_i]$, $\tilde{p}$ where $\tilde{p}$ is the reference distribution of the sample. If for some reason we want to compute the expectation of some other function, or want to compute the probability of a member of $\mathcal{X}$, we need to know $\tilde{p}$. The idea is to find a distribution p that approximates $\tilde{p}$ (has the same property expectations) and use this distribution merely as a substitute for $\tilde{p}$. Furthermore we have the constraint $\sum_i p_i = 1$. We now have $k + 1$ constraints and n unknown variables. If n is much larger than k, which is often the case, there are a lot of distributions that will meet the constraints, but if in addition the PME is applied, we can decide on a particular distribution. The constraints and the maximisation of entropy can, like Jaynes (1957) does, be formulated such that Lagrangian multipliers can be introduced. See for instance (Bertsekas 1982) for an explanation of this technique. The explanation below is based on (Jaynes 1996) (chapter 11).

Take g to be minus the entropy of distribution $p = p_1 \dots p_n$:

$$g(p) = -H(p) = \sum_{i=1}^{n} p_i \log(p_i)$$

and h the following function

$$\begin{aligned} h_0(p) &= \sum_i p_i - 1 \\ h_1(p) &= \sum_i p_i f_1(x_i) - \tilde{p}[f_1] \\ &\vdots \\ h_k(p) &= \sum_i p_i f_k(x_i) - \tilde{p}[f_k] \end{aligned}$$

Then, we want to find the minimum of g subject to constraints

$$h(p) = 0 \tag{4.2}$$

For every $x \in \mathbf{R}^n$ that is a local minimum of g, meets constraint 4.2, and has linearly independent gradient vectors, there exist a $\lambda \in \mathbf{R}^k$ such that

$$Dg(x) + \lambda^T Dh(x) = 0^T \tag{4.3}$$

We write out the jacobian matrices of functions g and h:

$$Dg(p) = \left[\frac{\partial g}{\partial p_1} \dots \frac{\partial g}{\partial p_n} \right] = [\log(p_1) + 1 \dots \log(p_n) + 1]$$

$$Dh(p) = \begin{bmatrix} \frac{\partial h_0}{\partial p_1} & \cdots & \frac{\partial h_0}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_k}{\partial p_1} & \cdots & \frac{\partial h_k}{\partial p_n} \end{bmatrix} = \begin{bmatrix} 1 & \dots & 1 \\ f_1(x_1) & \dots & f_1(x_n) \\ \vdots & \ddots & \vdots \\ f_k(x_1) & \dots & f_k(x_n) \end{bmatrix}$$

Applying equality 4.3 and introducing multipliers $\lambda_0 - 1, \lambda_2, \dots, \lambda_k$:

$$[\log(p_1) + 1 \dots \log(p_n) + 1] + [\lambda_0 - 1 \quad \lambda_1 \dots \lambda_k] \begin{bmatrix} 1 & \dots & 1 \\ f_1(x_1) & \dots & f_1(x_n) \\ \vdots & \ddots & \vdots \\ f_k(x_1) & \dots & f_k(x_n) \end{bmatrix} = 0^T$$

And from this we can see:

$$\log(p_i) = -\lambda_0 - \sum_{j=1}^{k} \lambda_j f_j(x_i) \Rightarrow \tag{4.4}$$

$$p_i = \exp[-\lambda_0 - \sum_{j=1}^{k} \lambda_j f_j(x_i)] \tag{4.5}$$

Ω_T	S(NP)	S(NP PUNCT)	S(VP PN)	S(PN VP)	NP(DET N)	NP(NUM N)	NP(N)	WHNP(WH NP)	VP(WHNP V)	VP(V NP)
S(NP(DET N))	1	0	0	0	1	0	0	0	0	0
S(VP(WHNP(WH NP(N)) V) PN)	0	0	1	0	0	0	1	1	1	0
S(PN VP(V NP(DET N)))	0	0	0	1	1	0	0	0	0	1
S(NP(NUM N) PUNCT)	0	1	0	0	0	1	0	0	0	0
S(NP(N))	1	0	0	0	0	0	1	0	0	0

Table 4.4: Sample space of the grammar in example 4.5.

Using that $\sum_i p_i = 1$, we separate e^{λ_0} as follows:

$$\sum_i p_i = \sum_i \exp[-\lambda_0 - \sum_{j=1}^{k} \lambda_j f_j(x_i)] \Rightarrow$$

$$e^{\lambda_0} = \sum_i \exp[-\sum_{j=1}^{k} \lambda_j f_j(x_i)]$$

$$\lambda_0 = \log \sum_i \exp[-\sum_{j=1}^{k} \lambda_j f_j(x_i)]$$

$$\lambda_0 = \log Z(\lambda_1, \ldots, \lambda_k)$$

where

$$Z(\lambda_1, \ldots, \lambda_k) = \sum_i \exp[-\sum_{j=1}^{k} \lambda_j f_j(x_i)]$$

Now we can write p_i in 4.5 as

$$p_i = \frac{1}{Z(\lambda_1, \ldots, \lambda_k)} \exp[-\sum_{j=1}^{k} \lambda_j f_j(x_i)]$$

Example 4.5 (Maximum entropy model of a tree-bank)
We return to our running example. As opposed to probabilities that depend on some generation mechanism, the exponential model we described earlier allows us to choose properties and estimate parameters independent of a generation mechanism. To illustrate the flexibility we have in choosing properties, we introduce a property for each context-free rule present in Ω_T. For instance, the property $f_{\texttt{S(NP)}} : \Omega_T \to \mathbf{N}$ assigns to each tree the number of times rule `S(NP)`

is applied to create it. In table 4.4 the properties for each of the event trees are given. Whereas finding rule probabilities for the classical case is rather simple, finding the parameters of the exponential model is less trivial.

The principle of maximum entropy states that the distribution we should use is that which satisfies the constraints in (4.2) and maximises entropy: if we define $P(f)$ as all those distributions that satisfy the expectation constraints for properties $f = (f_1, \ldots, f_k)$

$$P(f) = \{p | p[f_j] = \tilde{p}[f_j],\ j = \{1, \ldots, k\}\} \tag{4.6}$$

then p^* is the distribution that the maximum entropy principle advises us to take

$$p^* = \arg \max_{p \in P(f)} H(p)$$

Now we extend the idea of using properties for defining probability distributions. With each property f_j a parameter λ_j is associated. We define maximum entropy probability functions as follows:

$$p(x) = \frac{1}{Z} \hat{p}(x)$$

where

$$\hat{p}(x) = \exp[\sum_{j=1}^{k} \lambda_j f_j(x)],\ 0 < \lambda_j < \infty \tag{4.7}$$

$$Z = \sum_{x \in \mathcal{X}} \hat{p}(x) \tag{4.8}$$

We define the set $Q(f)$ of all probability distributions based on k properties and parameters:

$$Q(f) = \{p | p(x) = Z^{-1} \exp[\sum_{j=1}^{k} \lambda_j f_j(x)],\ 0 < \lambda_j < \infty\} \tag{4.9}$$

where $f = (f_1, \ldots, f_k)$ and Z as before.

However, the distribution $q \in Q(f)$ with maximum log-likelihood with respect to the reference distribution $\tilde{p}$ is the same distribution as the distribution $p \in P(f)$ that has maximum entropy. This is a result from research on maximum entropy optimisation.

Theorem 4.6 (duality)
If $p^* \in P(f) \cap Q(f)$, then

$$p^* = \arg \max_{p \in P(f)} H(p) = \arg \max_{q \in Q(f)} L(\tilde{p}, q)$$

and p^* is unique.

For a proof we refer to (Darroch and Ratcliff 1972; Della Pietra et al. 1997; Ratnaparkhi 1997b). This duality states that the optimal distribution p^* with respect to maximum likelihood framework fits the data as closely as possible, whereas p^* does not assume any information beyond the property expectation constraints (equation 4.2). Another reason why this duality is interesting to us is that we can try to find an optimal distribution in either $P(f)$ or $Q(f)$. Because in $Q(f)$ probability distributions are parameterised according the a set of properties, optimisation in $Q(f)$ is more feasible for implementation.

4.4 Parameter estimation

Scaling algorithms are iterative procedures that estimate the parameters of exponential distributions as defined in the previous section. We take a look at the Generalized Iterative Scaling (GIS) algorithm (Darroch and Ratcliff 1972), and the Improved Iterative Scaling (IIS) algorithm (Della Pietra et al. 1997), of which the latter puts less constraints on the sample space.

4.4.1 Generalized Iterative Scaling

The GIS algorithm is an iterative procedure that estimates the weights λ of the properties f of the unique optimal distribution $p^* \in P \cap Q$. It was developed by Darroch and Ratcliff (1972).

The GIS procedure requires that for each event $x \in \mathcal{X}$ the number of properties that are active for x equals some constant value $C \in \mathbf{N}$. Ratnaparkhi (1997b) proposes to introduce a correction property to achieve this.

$$\forall x \in \mathcal{X} : f_{\#}(x) = C \tag{4.10}$$

where $f_{\#}(x) = \sum_{j=1}^{k} f_j(x)$. If this is not the case a *correction* property f_{k+1} has to be added such that

$$\forall x \in \mathcal{X} : f_{k+1}(x) = M - \sum_{j=1}^{k} f_j(x)$$

where M is defined as

$$M = \max_{x \in \mathcal{X}} f_{\#}(x)$$

Note that when the correction property is added, the sum of the total number of properties that are active for each event satisfies (4.10). Property f_{k+1} may be non-binary. Another requirement for the GIS procedure to work is that each event $x \in \mathcal{X}$ has at least one property active:

$$\forall x \in \mathcal{X} : f_{\#}(x) > 0$$

Initially the parameters λ are set to 1, the iterative step is given by

$$\lambda^{(n+1)} = \lambda^{(n)} + \frac{1}{C} \log \left(\frac{\tilde{p}[f]}{p^{(n)}[f]} \right) \tag{4.11}$$

where $p^{(n)}$ is given by

$$\begin{aligned} p^{(n)}(x) &= \frac{1}{Z} \hat{p}^{(n)}(x) \\ \hat{p}^{(n)}(x) &= \exp[\sum_{j=1}^{k+1} \lambda_j^{(n)} f_j(x)] \\ Z &= \sum_{x \in \mathcal{X}} \hat{p}^{(n)}(x) \end{aligned}$$

In practice, the iteration can be stopped if the gain in likelihood becomes smaller than some $\epsilon \in \mathbf{R}$.

Computation

In each iteration of the GIS procedure we need $\tilde{p}[f_j]$ and $p^{(n)}[f_j]$. The computation of the former is straightforward and has to performed only once. Suppose we have a sample $S = \{x_1, x_2, \ldots, x_N\}$, then we can compute:

$$\tilde{p}[f_j] = \sum_{i=1}^{N} \tilde{p}(x_i) f_j(x_i) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i)$$

The computation of $p^{(n)}[f_j]$ involves the complete $\mathcal{X}$:

$$p^{(n)}[f_j] = \sum_{x \in \mathcal{X}} p^{(n)}(x) f_j(x)$$

Given k properties, in the worst case $\mathcal{X}$ has 2^k distinct elements. For large sets of properties the computation of $p^{(n)}[f_j]$ may become in-tractable. Therefore, we show an approximation given in (Lau et al. 1993). It assumes that the sample space is a subset of the cartesian product of two sets, a set of contexts $\mathcal{X}$ and a set of classes $\mathcal{Y}$: $\mathcal{X} \subseteq \mathcal{X} \times \mathcal{Y}$.

$$\begin{aligned} p^{(n)}[f_j] &= \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p^{(n)}(x,y) f_j(x,y) && (4.12) \\ &= \sum_{y \in \mathcal{Y}} p(y) \sum_{x \in \mathcal{X}} p^{(n)}(y|x) f_j(x,y) && (4.13) \\ &\approx \sum_{y \in \mathcal{Y}} \tilde{p}(y) \sum_{x \in \mathcal{X}} p^{(n)}(y|x) f_j(x,y) && (4.14) \end{aligned}$$

Note that in rewriting (4.12) into (4.13) Bayes' rule was used (see section 4.1).

4.4.2 Improved Iterative Scaling

The Improved Iterative Scaling algorithm was introduced in (Della Pietra et al. 1997). It improves the GIS procedure in that it does not require every event to have the same number of properties; the correction property is now obsolete.

We denote the set of maximum likelihood distributions as usual by $P(f)$ (see (4.6)), and the set of exponential distributions by $Q(f)$.

The IIS procedure is based on an iterative re-estimation of the parameters λ by γ such that the likelihood is increased: $L(\lambda.\gamma) - L(\lambda) > 0$. Because $L(\lambda.\gamma) - L(\lambda) > 0$ is not a concave function, an auxiliary function $A(\gamma, \lambda)$ that is a lower bound on this function, is concave, and hence can be maximised.

$$A(\gamma, \lambda) = 1 + \gamma\tilde{p}[f] - \sum_{\omega} p(\omega) \sum_{i} \frac{f_i(\omega)}{f_{\#}(\omega)} \exp[\gamma_i f_{\#}(\omega)]$$

Initially the parameters λ are initialised randomly, the iterative step is given by

$$\lambda^{(n+1)} = \lambda^{(n)} + \gamma$$

where γ is the solution of

$$\frac{\partial A(\gamma, \lambda)}{\partial \gamma} = p^{(n)}[f \exp[\gamma f_{\#}]] - \tilde{p}[f] = 0 \tag{4.15}$$

In fact, we still have a correction property $f_{\#}$ that compensates for the possibility of different number of properties that are active.

The following theorem states the convergence of the IIS procedure. For a proof we refer the reader to (Della Pietra et al. 1997).

Theorem 4.7 (Della Pietra et al.)
Suppose $p^{(n)}$ is the sequence of distributions in Δ determined by the IIS algorithm. Then $D(\tilde{p}\|p^{(n)})$ decreases monotonically to $D(\tilde{p}\|p^*)$ and $p^{(n)}$ converges to

$$p^* = \arg \min_{q \in Q(f)} D(\tilde{p}\|q) = \arg \min_{q \in P(f)} D(p\|p^{(0)})$$

Computation

Equation (4.15) can be transformed into the following polynomial equation.

$$\sum_{m=0}^{M} a_{j,m}^{(n)} \exp[\lambda_j^{(n)} m] = 0 \tag{4.16}$$

where M is defined as the maximum number of properties an arbitrary event $x \in \mathcal{X}$ may have (see also equation (4.4.1)). The $a_{j,m}^{(n)}$ are defined as

$$a_{j,m}^{(n)} = \begin{cases} p^{(n)}[f_j \delta(m, f_{\#})] & m > 0 \\ -\tilde{p}[f_j] & m = 0 \end{cases} \tag{4.17}$$

where the Kronecker δ is defined as usual and where

$$p^{(n)}[f_j\delta(m, f_\#)] = \sum_{x\in\mathcal{X}} p^{(n)}(x)f_j(x)\delta(m, f_\#(x))$$

If we look a bit closer at (4.17), then we see that $a^{(n)}_{m,j}$ is the expected number of times that property f_j appears in an event for which a total number of m properties are active. We show that (4.16) is equivalent with (4.15):

$$\begin{aligned}
\sum_{m=0}^{M} a^{(n)}_{j,m}\exp[\lambda^{(n)}_j m] &= 0\\
\sum_{m=1}^{M} p^{(n)}[f_j\delta(m, f_\#)]\exp[\lambda^{(n)}_j m] &= \tilde{p}[f_j]\\
\sum_{m=1}^{M}\sum_{f_\#(x)=m} p^{(n)}(x)f_j(x)\exp[\lambda^{(n)}_j m] &= \tilde{p}[f_j]\\
\sum_{x\in\mathcal{X}}\sum_{m\in\{1..M\}:m=f_\#(x)} p^{(n)}(x)f_j(x)\exp[\lambda^{(n)}_j m] &= \tilde{p}[f_j]\\
\sum_{x\in\mathcal{X}} p^{(n)}(x)f_j(x)\exp[\lambda^{(n)f_\#(x)}_j] &= \tilde{p}[f_j]\\
p^{(n)}\left[f_j\exp[\lambda^{(n)}_j f_\#]\right] &= \tilde{p}[f_j]
\end{aligned}$$

Equation (4.16) has no solution if and only if $a^{(n)}_{m,j} = 0$ for $m > 0$. This the case there is no event that has property f_j. Newton's method can be used to solve equation 4.16 iteratively.[4]

Example 4.8
We return to the tree-bank we presented earlier. Applying the IIS algorithm to the sample space defined by the reference distribution and the properties defined in example 4.5, we find the parameters displayed in table 4.5. By theorem 4.7 we know that the distribution we have found minimises KL divergence w.r.t. to the tree-bank (this particular distribution has KL divergence in the order of $1e-16$.), and therefore that it maximises the likelihood of observing the tree-bank and that it is a maximum entropy distribution. Further, the parameters give us an indication of how important they are for inducing a distribution close to the observed one. There is no relation whatsoever to the rule probabilities of the PCFG we inferred in example 4.4. Although both the parameters given in

[4] *Newton's method* is an iterative procedure for approximating the roots of any function f that is differentiable in its domain. The procedure is as follows:

1. $x^{(0)} = x_0$; goto step 2;
2. $x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$; increase n by 1 and goto step 2;

x_0 should be chosen such that $f'(x_0) \neq 0$.

property	parameter
S(NP)	1.75212242386314
NP(DET N)	1.6085311929592
NP(NUM N)	1.53648579804034
S(NP PUNCT)	1.53648579804034
VP(WHNP V)	0.815089768059743
WHNP(WH NP)	0.815089768059743
S(VP PN)	0.815089768059743
VP(V NP)	0.529487658184614
S(PN VP)	0.529487658184614
NP(N)	0.222237152300274

Table 4.5: Parameters of the MaxEnt CFG.

the above table and the rule probabilities of the PCFG are maximum likelihood estimates, they are inferred under different constraints and therefore incomparable. The parameters of the maximum entropy model meet the extra constraint that they induce a maximum entropy distribution. This is captured by the KL divergence very elegantly: in minimising the KL divergence a maximum likelihood distribution is induced that has maximum entropy. The rule probabilities meet the constraint that probabilities for rules that rewrite the same nonterminal sum to one; a grammar that meets this constraint is sometimes called *proper*. So ML estimation for PCFGs and ML estimation for maximum entropy model maximise likelihood in different domains of probability distributions.

An interesting consequence of this fact is that the probability distribution induced by a PCFG, in general, has a smaller likelihood than a maximum entropy model. The PCFG 'loses' probability mass in trees that are not in the tree-bank; alternatively, we can say that the PCFG tries to sum the probabilities of any tree it can derive to one, whereas a maximum entropy model tries to induce a distribution that sums the probabilities of the observed trees to one.

At this point we have to say something about generalisation of the two approaches. Whereas PCFG is a valid probability distribution over its language, we cannot say the same thing of our maximum entropy model. Insofar the tree-bank was representative and large enough for the language of the grammar we induce, we have nothing to worry about. However, if the tree-bank is small when compared to the language of the grammar (think of recursion), we have to re-estimate the parameters of our model. According to Abney such a re-estimation can be done by sampling from the grammar using the PCFG probabilities. The sample of the grammar is then considered as a new sample space, and re-estimation can be done by starting a scaling algorithm with parameters initialised at the already found values.

4.5 Monte Carlo Sampling

For the computation of expectations $p^{(n)}[f_j]$ for the GIS procedure (see equation 4.11), or the $a_{m,j}$ for the IIS algorithm (equation 4.15), the complete sample space Ω should be taken into account. For large k that is computationally not feasible. Instead, we have two alternatives. We can assume that the original sample (from $\tilde{p}$) is large enough to compute the $a_{m,j}$ from. In general, this is a dangerous assumption, because it may result in *overlearning*, i.e. the model will not generalise very well to 'unseen' events. The probability distribution that results from scaling will fit the original sample very well in terms of KL divergence, but property expectations may be all wrong for other samples.

The other solution for computing $a_{m,j}$ for large sample spaces, is using Monte Carlo sampling to find a representative subset of the sample space. Monte Carlo algorithms enable the simulation of processes that involve random factors such that properties of these processes (like function expectations) can be estimated, or states of the process can be collected. See (Sobol' 1994) for an introduction to the Monte Carlo method.

We consider one particular Monte Carlo algorithm, the Metropolis algorithm, because of its suitability for high-dimensional event spaces. Our presentation is based on (Neal 1993). To explain the Metropolis algorithm, we assume that all properties $f_1, \ldots, f_k$ are binary and we consider the sample space as a set of bit vectors: $\mathcal{X} = \{0,1\}^k$. The algorithm 'walks' through $\mathcal{X}$ state by state, and subsequent states differ one bit at most. The acceptation or rejection of a state for the sample that is being produced, depends on the distribution p defined on $\mathcal{X}$.

Algorithm 4.9 (Metropolis)
The input is a density h on $\mathcal{X}$ and an integer N. The output is a sample $x_0, \ldots, x_N$.

1. select a start state x_0; set $k = 0$ and $n = 0$;
2. x is x_n with bit i flipped;
3. compute $R = h(x)/h(x_n)$;
4. sample y from the uniform distribution on $[0,1]$, and set the new state x_{n+1} as follows

$$x_{n+1} = \begin{cases} x, & \text{if } y < R \\ x_n, & \text{otherwise} \end{cases}$$

   now increase n, increase i module k, and if $n < N$ goto step 2.

It can be shown that for $N \to \infty$ the sequence $x_0, x_1, \ldots, x_N$. If the value of bit i is sampled from a proposal distribution q.

4.6 Predictive maximum entropy models

In this section we will slightly reformulate maximum entropy models such that they are suitable for predictive application. Like in section 4.4.1 we assume that we have $(X, Y) : \Omega \to \mathcal{X} \times \mathcal{Y}$. The probability distribution that we want to induce, has the form

$$p(y|x) = \frac{1}{Z(x)} \exp[\sum_{i=1}^{k} \lambda_i f_i(x, y)]$$

where

$$Z(x) = \sum_{y \in \mathcal{Y}} \exp[\sum_{i=1}^{k} \lambda_i f_i(x, y)]$$

The parameters of such a conditional model can be approximated using one of the scaling algorithms. Alternatively we could apply Berger's version of the IIS algorithm that is adapted for approximating conditional models (Berger 1997); it differs from the version we gave previously in that γ_i is the solution of

$$\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in \mathcal{Y}} p^{(n)}(y|x) f_i(x, y) \exp[\gamma_i f_\#(x, y)] = \tilde{p}[f_j]$$

Although a bit pre-occupied with machine translation, (Berger et al. 1996) is a good introduction to the use of such conditional maximum entropy models for classification problems in natural language processing.

4.7 Summary

Constrained optimisation of entropy measures seems a promising paradigm in statistical NLP. We think there are a number of reasons for their growing popularity among computational linguists. The general applicability of maximum entropy modelling is the most important one: entropy models can be applied to events of any structure; the properties take care of the "internal" structure of the events. Another reason is that properties may overlap; the scaling algorithms do not make any assumptions about the dependence or independence of properties.

In this chapter we defined exponential property-based probability distributions and explained how constrained maximum entropy optimisation coincides with maximum likelihood estimation. We discussed two iterative scaling algorithms to determine the optimal parameters of a set of properties. We summarise the most important points here:

properties The sample space and therefore the observed data S is characterised by so-called property functions $f_1, \ldots, f_k$ where $f_i : \mathcal{X} \to \mathbf{N}$; the observed expectation of property f_i is denoted by $\tilde{p}[f_i]$.

constraints We want a probability distribution p such that $p[f_i] = \tilde{p}[f_i]$, $i = 1 \ldots k$.

principle If there are multiple distributions that meet the expectation constraints, then we select the one with maximum entropy.

Lagrange It can be shown by the introduction of Lagrangian multipliers that the distribution p that meets the expectation constraints and has maximum entropy is of the form:

$$p(x) = \frac{1}{Z} \exp[\sum_{i=1}^{k} \lambda_i f_i(x)]$$

classification Often the sample space is defined as a cartesian product of contexts $\mathcal{X}$ and classes $\mathcal{Y}$; conditional maximum entropy distributions have the form

$$p(y|x) = \frac{1}{Z(y)} \exp[\sum_{i=1}^{k} \lambda_i f_i(x, y)]$$

where

$$Z(y) = \sum_{x \in B} \exp[\sum_{i=1}^{k} \lambda_i f_i(x, y)]$$

Part II

Application

Chapter 5
The Grammar Inference Engine

We believe that a corpus-based approach is the best guarantee for obtaining a language model that covers most of the syntactic constructions that are used in a particular domain. Therefore we decided to annotate the SCHISMA Wizard of Oz corpus and automatically generate grammars from the annotated corpus. In this chapter we explain the annotation scheme that we developed for the SCHISMA Treebank, and we present an engine for the inference of grammars from the tree-bank.

In the next section we give an introduction to the Standard Generalized Markup Language (SGML), which is the language we used for the definition of the annotation scheme. Section 5.2 presents the annotation scheme itself. In section 5.3 we compare our annotation scheme to some other well-known schemes. In section 5.4 we discuss the generation of context-free grammars from the tree-bank, and in 5.5 we explain the generation of unification constraints. In section 5.6 we give some facts and figures of the tree-bank.

5.1 A short introduction to SGML

We used the Standard Generalized Markup Language (SGML) for the annotation of the SCHISMA corpus. SGML is a metalanguage for the definition of grammars that describe markup languages. Hypertext Markup Language (HTML), the language for creating documents for the World Wide Web, is an example of a markup language that is defined by an SGML grammar. We refer to (Goldfarb 1990) for the definition of the SGML standard. (Wood 1995) is an investigation into some theoretical and philosophical aspects of SGML.

An SGML document is simply a piece of text with SGML tags in it. An SGML tag is either an opening tag or a closing tag. Opening tags are of the form

```
<tagname>
```

or, if it carries attributes, of the form

```
<tagname attr1=value1 attr2=value2 ...>
```

Closing tags are of the form

```
</tagname>
```

Closing tags cannot carry attributes. An opening tag and a closing tag with the same name may enclose a piece of text data. Suppose we have the following piece of text

```
SGML is a metalanguage
```

and we want to annotate that `metalanguage` is a noun, and that `a` is a determiner then, we can introduce a tags named `noun` and `det` as follows:

```
SGML is <det>a</det> <noun>metalanguage</noun>
```

Introducing the `np` for noun phrases we arrive at:

```
SGML is <np><det>a</det> <noun>metalanguage</noun></np>
```

We make a distinction between tags that annotate text data directly, like `det` and `noun`, and tags that should entail further tags, like `np`. We will call the former tags *PoS tags*, and the latter *nonterminal tags*.

If we want to annotate that `SGML` is a proper name, and that `is` is a finite verb, then we can introduce the tags `name` and `verb`, and associate an attribute with `verb` to say that it is finite:

```
<name>SGML</name>
<verb form=fin>is</verb>
<np><det>a</det> <noun>metalanguage</noun></np>
```

Several types of attributes exist in SGML. For our purposes we distinguish three types of attributes:

- *identifier attributes*: an identifier attribute is an attribute that has a value (a number or a string) that is unique throughout the document. It can be referred to with a referring attribute. We will name identifier attributes `id`.
- *referring attributes*: a referring attribute refers to an identifier attribute.
- *ordinary attributes*: the value is a string or a number. In the example above the attribute `form` is an ordinary attribute.

Returning to our example, we introduce a tag for verb phrases `vp`, and we say that the proper name `SGML` is a noun phrase:

```
<np><name>SGML</name></np>
<vp>
  <verb form=fin>is</verb>
  <np><det>a</det> <noun>metalanguage</noun></np>
</vp>
```

If we want to annotate that SGML is the subject of the verb phrase, then we can give the noun phrase SGML an identifier attribute id, and the verb phrase a referrring attribute subj, and make the verb phrase refer to the noun phrase as follows:

```
<sent>
  <np id="some_id"><name>SGML</name></np>
  <vp subj="some_id">
    <verb form=fin>is</verb>
    <np><det>a</det> <noun>metalanguage</noun></np>
  </vp>
</sent>
```

Note that we added sent tags. We have used the same example sentence as in example 2.1, and we annotated the same syntactic structure as the sentence has w.r.t. the example grammar.

Each SGML marked up document should contain or refer to a Document Type Definition (DTD). A DTD specifies (among other things) the markup tags that may be used in the document, the attributes they may have, the types of the attributes, and the context-free nestings they are allowed to have. So, a DTD describes the language of allowed annotations. A DTD for the running example can be defined as follows:

```
<!ELEMENT sent    - -     (np,vp)>
<!ELEMENT np      - -     (name|(det,noun))>
<!ATTLIST np
          id      ID      #IMPLIED>
<!ELEMENT vp      - -     (verb,np)>
<!ATTLIST vp
          subj    IDREF   #IMPLIED>
<!ELEMENT name    - -     (#PCDATA)>
<!ELEMENT det     - -     (#PCDATA)>
<!ELEMENT noun    - -     (#PCDATA)>
<!ELEMENT verb    - -     (#PCDATA)>
<!ATTLIST verb
          form    (fin
                  |inf
                  |par)   fin>
```

The statements that start with ELEMENT define tags, those starting with ATTLIST define the attributes of a tag. The definition of the np tag says that it should entail either a name tag or a det tag followed by a noun tag. The dashes - - mean that both the closing and opening tags are required. The attribute specified for np is id; it is defined as an identifier attribute (ID). #IMPLIED means that not every np tag is required to have an id attribute.

tag	description
name	proper names
noun	nouns
verb	verbs
adj	adjectives
adv	adverbia
pn	pronoun (except interrogative)
wh	interrogative pronouns (WH words)
prep	prepositions
det	determiners
number	numbers
ordinal	ordinals
yn	yes/no words
conj	connectives
sep	separated words (mostly prepositions, discontinuity)
iject	interjections
punct	punctuation
misc	rest group

Table 5.1: Overview of PoS tags.

5.2 Annotation scheme

We applied a rather flat syntactic structure. The sentences contained in the corpus generally are of such a simple syntactic structure, that a more detailed syntactic annotation cannot be justified by the data we have. Only syntactic properties and relations were annotated. First we will give an overview of the tagset, then in section 5.2.2 we will explain what relational information we included in the attributes of the tags. Section 5.2.3 discusses the rest of the attributes. In appendix A the DTD of the annotation scheme can be found.

5.2.1 Tagset

Now we will discuss the syntactic structure we assigned to the utterances in the corpus by explaining what SGML tags we applied and what they mean. We divide the tags we used in our annotation in *nonterminal* and *Part-of-Speech PoS* tags. PoS tags do not entail any other tags, in grammatical terms they denote lexical categories. Nonterminal tags never contain data directly, they should entail nested tags only. Nonterminal tags correspond to nonterminals in a formal grammar.

Tables 5.1 and 5.2 give an overview of the tagset with a short description of

tag	description
utt	utterances (existing of connectives, punctuation, and sentences)
sent	sentences
emb	sub-ordinate or co-ordinate clauses
whnp	WH word followed by a noun phrase
np	noun phrases
pp	prepositional phrases
cn	compound noun

Table 5.2: Overview of nonterminal tags.

each tag. Most tags have self-explanatory names.

5.2.2 Syntactic relations

Syntactic relations are modelled using identifier and referring attributes. Most of the syntactic relations we distinguished are associated with verbs. Some relations apply to finite verbs only, others put other restrictions on them. A special relation is the one for representing discontinuity. Below we have explained each of the syntactic attributes accompanied by an example. For each example we provide a literal translation, and if necessary a free translation.

Main verb

Finite auxiliary verbs may have the attribute `main` that refers to the main verb (if present). In the following example `wil` is the finite auxiliary verb, and `reserveren` is the main verb. The value of the attribute `main` of the verb `wil` refers to the value of the attribute `id` of the verb `reserveren`.

```
<adv>Dan</adv>
<verb main="mu161s1">wil</verb>
<pn>ik</pn>
<adv>graag</adv>
<np><number>twee</number><noun>kaartjes</noun></np>
<verb id="mu161s1">reserveren</verb>
```

Below is a literal translation and a free translation:

Dan wil ik graag twee kaartjes reserveren
Then would I like two tickets to book
Then I'd like to book two tickets

Subject

A finite verb may have a reference to its subject through the attribute `subj`. In the following example the attribute `subj` of the verb `heeft` refers to the noun phrase `het stuk`.

```
<verb subj="su501s1">heeft</verb>
<np id="su501s1"><det>het</det><noun>stuk</noun></np>
<np><adj>goede</adj><noun>recenties</noun></np>
<punct>?</punct>
```

Translation:

heeft het stuk goede recenties?
has the play good reviews?
does the play have good reviews?

Copulae

Copulae may have a complement. In the following example the complement of the verb `is` is the adjective `beperkt`.

```
<np><det>de</det><noun>reductie</noun></np>
<verb nwg="nu2200s1">is</verb>
<adj id="nu2200s1>beperkt</adj>
```

Translation:

de reductie is beperkt
The reduction is limited

Direct object

Through the attribute `dirobj` a verb may refer to a direct object. In the example below the transitive verb `ophalen` refers to the noun phrase `de kaartjes`:

```
<wh>Wanneer</wh>
<verb>kan</verb>
<pn>ik</pn>
<np id="du881s1"><det>de</det><noun>kaartjes</noun></np>
<verb dirobj="du881s1">ophalen</verb>
<punct>?</punct>
```

Translation:

Wanneer kan ik de kaartjes ophalen?
When can I the tickets pick up?
When can I pick up the tickets?

Indirect object

In the following example `my` is the indirect object of the verb `schikt`.

```
<pn>Dat</pn>
<verb indobj="iu1724s1">schikt</verb>
<pn id="iu1724s1">mij</pn> <adv>wel</adv>
```

Translation:

Dat	*schikt*	*mij*	*wel*
That	*suits*	*me*	*fine*

Discontinuity

Verbs, adverbs, and WH words may be discontinuous. The words separated from them are annotated with the PoS tag `sep`, and have an attribute `prev` that refers to the main part of the word. In case of discontinuous verbs the main part is the part that can function as a verb in itself. In case of WH words and adverbs the main part is the first part of the complete word. In the following example the word `over` is separated from `waar`, and its attribute `prev` refers to the main part `waar`.

```
<wh id="wu1952s1">waar</wh>
<verb>gaan</verb>
<np><pn>deze</pn><noun>opera's</noun></np>
<sep prev="wu1952s1">over</sep>
```

Translation:

Waar	*gaan*	*deze*	*opera's*	*over*
What	*are*	*these*	*operas*	*about*

5.2.3 Other attributes

Now we discuss the most important non-relational attributes. One of the most prominent is the `type` attribute for sentences and clauses. For pronouns, verbs and nouns, some attributes will be discussed as well.

Sentence types

The `type` attribute of sentences and clauses sets the type of sentence or clause we have. First we take a look at the values for `type` that imply the presence of a verb:

- `decl`: the sentence is a statement.
- `declinv`: the sentence is a statement with inverted subject and finite verb. In the example below `wil` is the finite verb and `ik` the pronoun.

```
<sent type=declinv>
  Dan wil ik graag twee kaartjes
    voor dat stuk met Hajo reserveren.
</sent>
```

Translation:

Dan	*wil ik graag*	*twee kaartjes voor dat stuk met Hajo*	*reserveren.*
Then	*I'd like*	*two tickets for that play with Hajo*	*to book.*

- `yndecl`: the sentence is a statement meant as a question. Example:

```
<sent type=yndecl>
  Dat is dus met korting ?
</sent>
```

Translation:

Dat	*is*	*dus*	*met korting ?*
That	*is*	*so*	*with reduction ?*
So that includes reduction ?			

- `ynq`: the sentence puts a yes/no question. The finite verb appears before the subject. Example:

```
<sent type=ynq>
  Is dat met korting ?
</sent>
```

Translation:

Is	*dat*	*met*	*korting ?*
Is	*that*	*with*	*reduction ?*
Does that include reduction ?			

- `whq`: the sentence is a question starting with a WH word.
- `imp`: imperative sentence.
- `rel`: the sentence is a subordinate clause starting with a pronoun.

Further possible values for for the `type` attribute correspond to each of the PoS and nonterminal tags. For instance, a sentence type of `np` is used for a sentence that consists of a noun phrase only.

Pronoun types

Pronouns have a `type` attribute as well. It states what kind of pronoun it is. It may have the following values:

- `wd`: reflexive pronoun; example: `zich` (*himself/herself*);
- `demo`: demonstrative pronoun; example: `dat`, `deze`;
- `ob`: indefinite pronoun; example: `enige` (*some*), `alle` (*all*);
- `wg`: reciprocal pronoun; example: `elkaar` (*each other*);
- `rel`: relative pronoun; example: `die`, `dat` (*that*);
- `pers`: personal pronoun; possessive pronouns were given the `pers` type as well; they have for their `case` attribute `C2`;

Obviously we miss out the interrogative pronouns (WH words). We gave them their own PoS tag `wh`, because they play such an important role in syntactic structure of questions. For personal pronouns the `case` attribute is important: it may have value `C1` to `C4` indicating nominative, genitive, dative and accusative cases, respectively.

Verb attributes

In addition to the several syntactic relations the verb takes part in, we annotated several other syntactic features of which the most important are:

- `pass`: values `pn`, `py`; states whether the verb is in the passive form; if not specified it defaults to `pn`
- `per`: values `p1`, `p2`, `p3`, `pernil`; the person of the verb (finite verbs only); the default is `pernil`;
- `num`: values `u`, `p`, `s`, `numnil`; the number of the verb (finite verbs only); the default is `numnil`; `u` is used for the polite form (as in `u wilt`);
- `type`: values `aux`, `main`; whether the verb is used as a main verb or as an auxiliary verb; defaults to `main`;
- `form`: values `fin`, `inf`, `part`, `imp`; the form of the verb: finite, infinitive, participle, or imperative, resp.; defaults to `fin`;

Noun attributes

A noun has two possible attributes. The `num` attribute for the representation of the number. Its possible values are `s` and `p`. If it is not specified `num` defaults to `s`.

The other attributes is `gen` for gender. The possible values are `m`, `f`, `mf` (male or female, don't care) and `n`. If not specified the attribute defaults to `mf`.

Tagset	PoS	NT	#Words
Susanne	352	31	128k
PTB I	36(48)	15	1,600k
PTB II	36(48)	26	1,000k
C1	119(132)		
C2	154(166)	16	
C5(BNC)	62(73)		100,000k
C7(BNC)	160+		2,000k
WoZ	16(17)	8	4747

Table 5.3: Comparison of tagsets.

5.3 Annotation scheme comparison

Compared to other annotation schemes we applied a fairly small set of tags. For instance the Susanne corpus was tagged with 352 different PoS tags, and 31 nonterminal tags (Sampson 1994). In the second version of the Penn Treebank (PTB II) 36 PoS tags are applied (Santorini 1995), and 26 nonterminal tags (Bies, Ferguson, Katz, and MacIntyre 1995). Penn Treebank I (PTB I) has only 15 nonterminal tags (Santorini 1991). The C1 to C7 figure are CLAWS (Constituent Likelihood Automatic Word-tagging System) tagsets (CLAWS 1998). C5 was applied for the British National Corpus (BNC 1997), C7 for a core corpus of the BNC. The numbers in between braces include tags for punctuation. See table 5.3 for an overview.

Our tagset may seem small, but has a lot of descriptive power if attributes are taken into account and applied in grammar generation. The large number of PoS tags used in the Susanne corpus is a consequence of coding all much information into tags. Interestingly, besides the BNC, none of the other corpora is annotated using SGML.

Although our annotation scheme is purely syntactic, it has some domain dependent aspects. The tagging of proper names is an example. Proper names are of central importance in the SCHISMA domain, as dialogues should provide the user with information on performances, and the artists involved in them. Another domain dependence is in the superficial syntactic structure we chose to annotate. As we stated at the beginning of this section the structure of the language used in the WoZ corpus is rather simple. Sentences are very short, 190 out of a total of 873 sentences have length 1, and the average length is 5.4 words (including punctuation). We put most of the hierarchical structure into noun phrases and prepositional phrases, clearly the most important constituents in an information domain like SCHISMA.

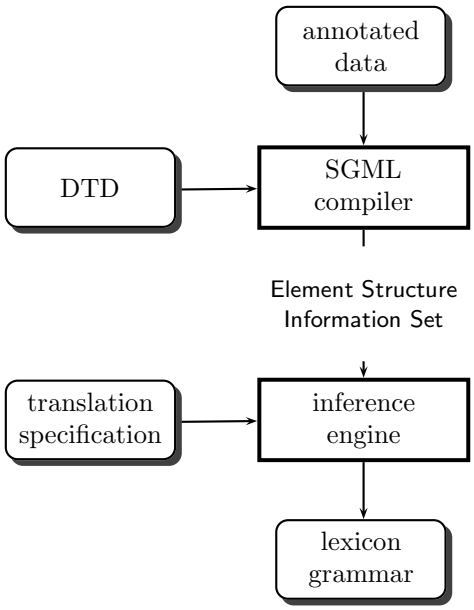


Figure 5.1: Grammar inference schema

5.4 Grammar inference

By tagging the SCHISMA corpus we assigned a syntactic structure to the sentences contained in it. It is quite straightforward to derive a context-free grammar from the annotated corpus. Moreover, as we included several kinds of syntactic relations like subject-verb, direct object, and indirect object, we can attach unification constraints to these rules as well. We developed a *grammar inference engine* that understands the output of the SGML compiler, takes a translation specification as input, and outputs a unification grammar according to the specification. The specification consists of a translation of SGML tags into grammar symbols, and a set of meta-constraints. For the generation of useful unification constraints we applied an interesting technique based on tag pattern matching. See figure 5.1 for an outline of the procedure.

The grammar inference engine we developed is independent of our annotation scheme. If we decide to change the annotation, or to process a corpus that is annotated completely different, we only have to change the translation specification that determine how tags are translated into grammar symbols, and how and when unification constraints are generated.

In order to generate a context-free grammar from the annotated data, we parse it with an SGML compiler. The compiler outputs the SGML annotated

PoS	**category**
name	PNAME
noun	N
verb	V
adj	ADJ
adv	ADV
pn	PN
wh	WH
prep	PREP
det	DET
number	NUM
ordinal	ORD
yn	YN
conj	CONJ
sep	SEP
iject	IJECT
punct	PUNCT
misc	UNKNOWN

Table 5.4: Translation of PoS tags to lexical categories (terminals).

data as an Element Structure Information Set which is a standardised ASCII representation consisting of commands with their parameters. The grammar inference engine interprets these commands. The most important commands with their parameters are as follows:

- *open tag*; its parameters are the name of the tag that was opened and the attributes with their values;
- *data*; the only parameter is the data found in between some tags;
- *close tag*; has as parameter the name of the tag that is closed.

In order to keep track of the structure the open tags induce, the generator applies a stack of grammar rules. If an open tag X occurs and the stack is not empty, it is attached to the RHS of the rule on top of the stack. Then, if the tag is not a PoS tag, a new rule with X as LHS is pushed on the stack. For each (partial) rule on the stack additional administration is kept for the (attribute, value) pairs that were specified with the open tags. This information is used for generating unification constraints, as we will see in the next section. If a close tag occurs, and the stack is not empty, we know we have completed a rule. The rule is popped off the stack, and its unification constraints are generated using the (attribute, value) pairs we administered. Note that syntactic relations only

tag	nonterminal
utt	Z
sent	S
emb	EMB
whnp	WHNP
np	NP
pp	PP
cn	CN

Table 5.5: Translation of nonterminal tags to nonterminals.

make sense if they are specified within the same 'tag level', or, in grammatical terms, within the same RHS. In generating the grammar rule with its constraints the SGML tags are translated into terminal and nonterminal symbols as given in the tables 5.4 and 5.5.

5.5 Unification constraints

As we saw in the previous section, we annotated several syntactic relations. Syntactic relations can be used to generate unification constraints. In order to flexibly specify the generation of unification constraints from these syntactic relations, i.e. from tags and their attributes, we applied *meta-constraints* that pair SGML tag patterns with unification constraints. In fact the meta-constraints in such a specification are if-then rules that specify what a constraint should look like if its tag pattern is matched. The meta-constraints may contain variables for the left-hand side nonterminal, nonterminals matched by the meta-constraint, and nonterminals referred to by referential attributes. Meta-constraints come in five flavours:

1. *lexical constraints* specify what features to attach to lexical categories;
2. *RHS nonterminal constraints* state constraints for (tag, attribute, value) triples;
3. *identifier constraints* specify the identifier attributes for the SGML tags that may have one; no constraints can be specified;
4. *referring attribute constraints* tell the system what attributes are reference attributes and what constraints should be generated for them; and
5. *general constraints* give constraints for patterns of two RHS nonterminal in sequence and a LHS nonterminal.

In the subsections that follow we will discuss these five kinds of meta-constraints in detail. Appendix B contains the full meta-constraint specification as applied in our experiments.

5.5.1 Lexical constraints

A lexical constraints consists of three parts: an SGML tag, an attribute, and a (possibly empty) list of constraints. In these constraints the value of the attribute is available in the variable `$val`. Every time the system finds a lexical entry that matches one of the lexical meta-constraints, it will generate and output a constraint by outputting the string in the third column.

```
TAG     ATTR    ...$val...
```

Given the lexical constraint

```
NOUN    NUM     num$val
```

and annotated data (*tickets*)

```
<noun num=p>kaartjes</noun>
```

we obtain lexical entry (in the record format for PC-PATR lexicons; see (McConnel 1995))

```
\w kaartjes
\c N
\g
\f nump
```

This lexical meta-constraint tells the system to add to each lexical entry for `noun` a feature of the form `num$val`; for instance, if the value of the attribute `num` is `p` (plural), the feature output is `nump`.

5.5.2 RHS nonterminal constraints

These meta-constraints consist of a (tag, attribute, value) triple optionally followed by constraints; the value may be a wild card `*`;

```
TAG     ATTR    VAL     ...$val...$lhs...$0...
```

Variable translation:

- `$val`: the value of attribute `ATTR`
- `$0`: nonterminal for `TAG`
- `$lhs`: LHS nonterminal

Consider the following nonterminal constraints

```
PN      TYPE    *       <$0 head type>=$val
```

The star * matches any attribute value. If we observe the following annotated data (*That is fun*):

```
<sent>
  <pn type=demo>dat</pn>
  <verb>is</verb>
  <adj>leuk</adj>
</sent>
```

then the following unification constraint is output (the grammar rule given for clarity):

```
rule S -> PN V ADJ
  <PN head type>=typeDEMO
```

5.5.3 Identifier attribute constraints

Identifier meta-constraints state the identifier attributes reference attributes (see hereafter) may refer to. They are simply (tag, attribute) pairs.

```
TAG     ATTR
```

An example of an identifier attribute specification is

```
PN    ID
```

It states that the attribute `id` is an identifier attribute of SGML tag `verb`. No constraints are generated directly; a look-up table is build up for looking up the right nonterminal given a referring identifier. The following pattern matches the specified tag-attribute pair above.

```
<pn id="u161s1">ik</pn>
```

5.5.4 Referring attribute constraints

Referring attributes are (tag,attribute) pairs, optionally followed by constraints:

```
TAG     ATTR    ...$0...$1...$lhs...
```

We have one new variable here `$1` which is replaced by the nonterminal that is generated for the SGML tag referred to. Variable translation:

- `$0`: nonterminal for `TAG`
- `$1`: nonterminal referred to
- `$lhs`: LHS nonterminal

Here is a referring attribute constraint. It matches subject reference specified with verbs:

position	wild card	meaning
1	*	matches any tag as LHS
2	*	TAG3 is the last in the RHS
3	*	TAG2 is the first in the RHS
2,3	*	matches any rule with LHS TAG1
1,2,3	*	matches anything
2	+	TAG3 is the last tag in RHS, $\|RHS\| > 1$
3	+	TAG2 is the last tag in RHS, $\|RHS\| > 1$
2	>	TAG3 is the last tag of its type in RHS
2	<	TAG3 is the first of its type in RHS

Table 5.6: Meaning of the wild cards. |RHS| denotes the number of nonterminals in the RHS of the rule.

```
V       SUBJ    <$lhs pred subj>=<$1 head>
                <$lhs head>=<$0 head>
                <$0 head agr>=<$1 head agr>
```

The following data will match the above meta-constraint (*Then I'd like ...*):

```
<sent>
  <adv>dan</adv>
  <verb subj="su161s1">wil</verb>
  <pn id="su161s1">ik</pn>
  <adv>graag</graag>
  ...
</sent>
```

The following rule and constraints will be derived:

```
rule S -> ADV_1 V PN ADV_2 ...
  <S pred subj>=<PN head>
  <S head>=<V head>
  <V head agr>=<PN head agr>
```

Intuitively, the meta-constraint states that the head of a sentence is the head of the finite verb (detected by the presence of a subject reference); an agreement constraint is specified, and the subject is identified.

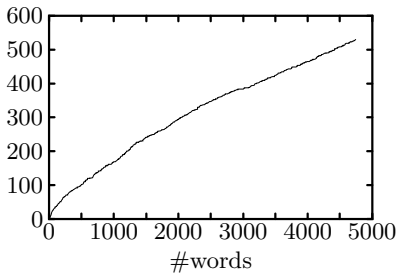


Figure 5.2: Size of the grammar as a function of the number of words.

5.5.5 General constraints

General meta-constraints come in several flavours; they consist of a three-tuple (lhs, rhs_1, rhs_2), optionally followed unification constraints. As usual constraints may contain variables for matched tags.

```
TAG1    TAG2    TAG3    ...$0...$1...$lhs...
```

We added one more powerful feature for generation constraints: instead of tags wild cards may be specified. In table 5.6 the meaning of the wild cards `*`, `+`, `<`, and `>` is given. Variable translation is as follows:

- `$0`: nonterminal for `TAG2`
- `$1`: nonterminal for `TAG3`
- `$lhs`: nonterminal for `TAG1`

Here is an example of a general constraint:

```
NP      >       NOUN    <$lhs head>=<$1 head>
```

It states that the last noun in the RHS of an NP rule is the head of the NP. If the following annotated data occurs (*two tickets*):

```
<np>
  <number>twee</number><noun>kaartjes</noun>
</np>
```

The following constraint will be generated:

```
rule NP -> NUM N
  <NP head>=<N head>
```

LHS	#	%	|**RHS**|
S	386	72.8	4.1
NP	86	16.2	2.8
EMB	26	4.9	4
Z	17	3.2	4.7
PP	10	1.9	2
WHNP	3	0.6	2.3
CN	2	0.4	2

Table 5.7: Percentages of rules per nonterminal, average length of the RHSs of the grammar rules.

5.6 Facts and figures

The unification grammar we obtain from the annotated corpus has 530 rules, the lexicon has 800 entries. If we disable the generation of unification constraint we obtain 462 grammar rules and 731 lexical entries. We have 7 nonterminals, and 17 lexical categories, which means that every PoS and nonterminal tags is mapped to a unique grammar symbol. Given the size of the corpus of 873 utterances the number of rules is large. If we study the growth of the grammar during generation (see figure 5.2), we see that it tends to stabilise at a constant growth. We think the corpus is too small to extrapolate the graph, but probably new rules will be added at a constant rate in case of more annotated data. Gaizauskas (1995) reports similar results for the PTB II grammar and gives possible explanations. We refer to (Herdan 1966) for a more detailed treatment of the relation between text length and the frequency of lexical and grammatical phenomena.

Another issue we have to consider here, is that we have very little data to compute our probabilities from. For instance, 307 out of 386 S-rules (80%) are found only once in the corpus; for NP-rules this figure is 53%. Overall 72% of the rules have frequency 1. However, the experiments that we report on in chapter 6 will show that probabilistic grammars with rule probabilities computed from these frequencies improve the performance of the parser. Charniak (1996) reports that 57% (9168 out of 15953) of the grammar rules he obtained from the Penn Treebank II corpus occurred only once. Charniak presents experiments in which he removed from the grammar the rules that have frequency 1, and still obtained good performance. We believe this approach cannot be fruitful in our case, as our grammar would become too small. It would not cover enough of the syntactic structure present in the SCHISMA Treebank.

5.7 Summary

In this chapter we presented our annotation scheme in detail. First we treated the different PoS and nonterminal tags and their syntactic meaning. Then we continued with defining several syntactic relations, and explained how they are represented in SGML using identifier and referring attributes. We discussed the most important non-relational attributes. We compared our tagset to some other tagsets, and saw that other tagsets tend to have large numbers of tags whereas we have a small number of tags with more detail in the attributes of the tags. We explained how we generate unification grammars from the annotated data. Special attention was given to the derivation of unification constraints.

Chapter 6
Experimental Results

In this chapter we present experiments with probabilistic grammars inferred from the SCHISMA Treebank. First, we will derive unification grammars and context-free grammars from the tree-bank and compare their performance on unseen data. Then we will define a probabilistic extension of context-free grammar based on the MaxEnt method, and compare its performance to the performance of plain context-free grammar and PCFG. Finally, we define a MaxEnt extension of PATR II, and compare it to plain PATR II and a probabilistic extension of PATR II based on a probabilistic extension of its context-free backbone.

This chapter is organised as follows. In section 6.1 we explain what parser we used and why. Also we discuss features we added for robustness and probabilistic parsing. Section 6.2 discusses parameter estimation for both PCFGs and MaxEnt grammars. In section 6.3 we explain the bracket precision, bracket recall, and crossing brackets measures that we applied for the evaluation of the tree-bank grammars. In section 6.4 we present an experiment which compares the performance of unification grammars and context-free grammars derived from the SCHISMA Treebank. Section 6.5 presents experiments with plain CFG, PCFG and MaxEnt models of CFG. In section 6.6 we discuss experiments with probabilistic models of unification grammar. In section 6.7 we discuss the value of our experiments, and of the parsing system we developed.

6.1 The parsing system

We decided to use the PATR II unification grammar formalism, as it is well understood, widely used, and it has proven its merits for natural language parsing (Shieber 1986). The parser we used for our experiments is PC-PATR of the Summer Institute of Linguistics (McConnel 1995). It is a left-corner chart parser for PATR II style unification grammars. We adapted it such that it supports probabilistic parsing, and ranks alternative parses by probability. These and other features were added to increase the *robustness* of the parser.

When do we consider a parsing system robust? If it finds for every possible utterance exactly one parse: the right one! That would make a robust parser, but is unrealistic at the same time, as we cannot model all knowledge the parser

would need to always produce the right parse. Even we ourselves cannot always decide on the right interpretation of an utterance, if it is structurally or semantically ambiguous. But we don't need to, we can ask for explanation, and so can a dialogue system. It's all right if the parser returns several alternative parses. It passes its results to some semantic/pragmatic interpreter which tries to understand the utterance in the context of the dialogue. We simply require a robust parser to always produce a parse, and, if it finds more alternative parses, to rank them. Below we discuss some features of our parsing system.

6.1.1 Unknown words and parsing failure

We built in a feature for handling unknown words, i.e. words not in the lexicon. If the parser finds a word not in the lexicon, it hypothesises four possible categories: noun, verb, adjective, or adverb. Features are left unspecified. We took these four as hypothetical categories, as we are almost sure that we have all function words in the lexicon. A drawback of this approach is that the number of possible category assignments for the words of a sentences grows exponentially with the number of unknown words.

Another important feature of PC-PATR with respect to robustness, is the graceful degradation it exhibits if it cannot construct a parse. The first thing it does is turn off unification. If that helps, it returns the parses it found with feature structures in which unification failures have been indicated. If disabling unification is not successful, it turns of top-down filtering, and builds a *parse bush*. A parse bush is built by finding the longest component beginning at the left side of the sentence and making it the leftmost branch of a bush. The parser then goes to the place where that component ends and picks a new longest component adding it as the next branch of the bush, etc., until the complete sentence is covered. Finally, a dummy top node is added. If parse ranking is enabled, it selects, in case of alternative partial parses of the same length, the one with the highest probability. This feature combined with the *Unknown words* feature gives PC-PATR 100% *coverage*, independent of the grammar and lexicon.

6.1.2 Probabilistic parsing

Adapting PC-PATR for parsing with PCFGs was quite simple. At two moments during parsing probabilities have to be set and/or computed. At the moment of building the initial chart, initial items (or edges) receive a probability 1. The other moment is when an active edge is extended by a passive (completed) edge: the probability of the active edge becomes the product of its original probability and the probability of the passive edge. Optionally, logarithmic probabilities can be used; in that case probabilities are added.

PC-PATR understands disjunctive feature structures and disjunctive constraints. Disjunctive feature structure are converted to Disjunctive Normal Form (DNF). Disjunctive constraints are also converted to DNF thereby introducing new grammar rules. Below is an example of a probabilistic grammar

rule with disjunctive constraints:

```
rule {probnp1} (0.6) NP -> {0.7 ADJ /0.3 PN} N
<NP head> = <N head>
<N head modify adj>=<ADJ head>
<N head agr>=<ADJ head agr>
<N head modify pn>=<PN head>
{0.4
 <PN head type>=Poss
/0.6
 <PN head type>=Demo
}
```

The parser expands this rule into 4 standard ones without optional/alternative constituents, and without disjunctive constraints; one of the rules is (the probability is given by $0.6 \times 0.3 \times 0.4$)

```
rule {probnp_exp1} (0.072) NP -> PN N
<NP head> = <N head>
...
<PN head type>=Poss
```

For the implementation of the property functions necessary for parsing with probabilistic grammars based on the MaxEnt method, we exploited the fact that properties are binary functions. In addition, we assumed that partial parses cannot lose a property if they are combined with other partial parses. Further, we assumed that properties are introduced by grammar rules, and cannot be lost once they have become active. The following example illustrates how properties are integrated into the PATR II language

```
rule {me_np1} (3,4,5,6) NP -> ADJ N
<NP head> = <N head>
<N head modify adj>=<ADJ head>
<N head agr>=<ADJ head agr>
```

This rule introduces 4 properties numbered 3,4,5, and 6. The administration of the description of the properties is kept outside from the parser. The properties of the `ADJ` and `N` constituents are propagated to the LHS constituent by set union with the properties introduced by the rule. In addition to a rule-wise specification of new properties, the parser needs a specification of the parameters of the properties used for computing the probability of a feature structure. In section 6.6 we will see how this simple extension of PATR II can be applied effectively for the probabilistic extension of unification grammar.

For more detailed information on PC-PATR, the extensions, and its user-interface we refer to (McConnel 1995; ter Doest 1998c).

6.2 Parameter estimation

The parameters of the PCFGs are estimated by determining normalised rule frequencies. Given a tree-bank, the probability of a rule $A \rightarrow \alpha$ is the number of times the rule is applied in the observed data divided by the number of times a rule is applied to rewrite an A. Formally:

$$p(A \rightarrow \alpha) = \frac{N(A \rightarrow \alpha)}{N(A)}$$

where $N(A \rightarrow \alpha)$ denotes the number of times $A \rightarrow \alpha$ is observed, and $N(A)$ the number of times A is observed.

For parameter estimation of the maximum entropy models we applied the Improved Iterative Scaling algorithm (Della Pietra et al. 1997). For the extension of context-free grammar, the event space consists of the trees of the grammar, and for unification grammar the event space is the set of feature structures. For simplicity, we did not apply sampling, and assumed the tree-bank to be representative for the language of the grammar. A description of the software for parameter estimation of MaxEnt models can be found in (ter Doest 1998b).

6.3 Parser evaluation

We evaluated the grammars we obtained according to bracket precision, bracket recall, and crossing bracket measures, also called the Grammar Evaluation Interest Group (GEIG) scheme (Grishman et al. 1992). The idea of the evaluation is to feed the parser a set of sentences (not annotated!), and compare the derivation trees it assigns to these sentences to the previously annotated ones in terms of bracketings (to be defined next). In most cases a parser will assign more than one tree to each sentence. We assume that the parser selects one for evaluation. Our probabilistic parsers select the tree with the highest probability, and non-probabilistic parsers select an arbitrary one.

A bracketing b of a sentence of length n is defined as a tuple (i, j), where $0 \leq i < j \leq n$. A labelled bracketing is defined as a triple (X, i, j) where i and j as before, and X is a bracketing label, which is a nonterminal or lexical category.

A derivation tree (as well as an annotated sentence) can conveniently be represented by a set of bracketings. If we consider the parsing process as an information retrieval procedure, namely the retrieval of the right set of bracketings, it is a small step to the idea of applying precision and recall measures to evaluate the parsing process. To explain this idea further we introduce the following figures:

N_c The number of bracketings correctly found by the parser.

N_t The number of bracketings in the annotated test data.

N_p The number of bracketings found by the parser.

N_{cross} The number of bracketings found by the parser which cross a bracketing in the test data.

Given the above absolute figures, we define the precision, recall and crossing bracketing measures as follows.

precision The ratio of the bracketings appearing in the most higly ranked parses also appearing in the corresponding parse in the testing data:

$$P = \frac{N_c}{N_p}$$

Labelled precision, denoted by P', means that nonterminals are taken into account while comparing bracketings.

recall The ratio of the bracketings in the annotated corpus also appearing in the most highly ranked parse:

$$R = \frac{N_c}{N_t}$$

Labelled recall, denoted by R', is defined similar to labelled precision.

crossing brackets The ratio of bracketings in the most highly ranked parses crossing over bracketings in the corpus.

$$C = \frac{N_{cross}}{N_p}$$

Developing a parser that optimises the recall measure is a trivial exercise: the set of all possible bracketings results in a recall of 1. Of course, we would get a very poor performance on the bracket precision and crossing brackets measures. Parsing, if considered a bracketing process and well-described in terms of the bracketing measures, is a balancing act between precision and recall. Precision measures to what extent the parser filters out wrong bracketings, recall measures the ratio of right bracketings. The crossing brackets measure gives an indication of how consistent parses were with the corpus analyses.

An advantage of evaluating parsers according to this scheme is that it less restrictive than counting the parses that are fully identical to those in the corpus. Parses that are only partially correct may still be of value for further processing (semantic interpretation, for instance). On the other hand, a 'flat' annotation like ours easily results in a low crossing brackets score. For a more detailed discussion of these problems with the GEIG scheme we refer to (Carroll and Briscoe 1998).

measure	(2)**CFG**	(1)**UG**
R	$92.2_{[1.27e\text{-}1]}$	$93.5_{[5.57e\text{-}2]}$
R'	$82.3_{[3.58e\text{-}1]}$	$84.6_{[1.97e\text{-}1]}$
P	$93.9_{[1.00e\text{-}1]}$	$94.6_{[6.08e\text{-}2]}$
P'	$87.2_{[1.96e\text{-}1]}$	$89.0_{[1.42e\text{-}1]}$
C	$4.48_{[1.26e\text{-}1]}$	$3.58_{[1.13e\text{-}1]}$

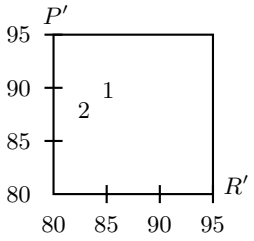


Table 6.1: Bracketing scores context-free grammar versus unification grammar. Values in between square bracketings are the score variances. The numbers (1) and (2) before the names CFG and UG correspond to the numbers in the graph to the right.

6.4 Experiment 1

In this section we describe an experiment to investigate the difference in performance between context-free grammars and unification grammars derived from the SCHISMA Treebank. We split the corpus randomly in a train and test set of 85% and 15% resp.; this corresponds to 743 utterances training data and 130 sentences testing data. Then we derived both a context-free grammar and a unification grammar from the train set, and tested it on the test set. We repeated this 10 times to compensate for the small size of the tree-bank. In table 6.1 the average bracketing scores of these 10 experiments are given. We see that unification grammar scores considerably better than context-free grammar. The variances of the experiments confirm that the improvement is not accidental or due to a fortunate choice of train and test data.

6.5 Experiment 2

Again we split the tree-bank randomly in a train and test set (85% and 15%), and did 10 train/test iterations. This time we compare three grammar formalisms: plain CFG, PCFG, and a MaxEnt model of probabilistic CFG For the maximum entropy model, we defined a binary property function for each rule of the grammar like we did for the example of the tree-bank in chapter 4.

The table shows that the results we obtained using the IIS algorithm are as good (or as bad) as those obtained by the classical approach; they are not better, but this is no surprise! The properties that define the maximum entropy distribution are less informative than rule frequencies (probabilities): properties *detect* the use of a rule in a tree, but do not say how often it is applied, whereas rule probabilities carry information on the frequency with which they are applied (globally).

The most important advantage of the statistical technique for inducing a

measure	(3)**CFG**	(1)**PCFG**	**MaxEnt** (2)**CFG**
R	$92.6_{[8.47e\text{-}2]}$	$94.9_{[3.69e\text{-}2]}$	$93.2_{[2.56e\text{-}2]}$
R'	$83.8_{[2.88e\text{-}1]}$	$90.8_{[7.58e\text{-}2]}$	$87.5_{[9.52e\text{-}2]}$
P	$94.0_{[1.03e\text{-}1]}$	$96.2_{[3.01e\text{-}2]}$	$95.7_{[2.64e\text{-}2]}$
P'	$88.5_{[2.46e\text{-}1]}$	$92.7_{[7.25e\text{-}2]}$	$91.0_{[9.80e\text{-}2]}$
C	$4.30_{[3.64e\text{-}2]}$	$2.68_{[2.08e\text{-}2]}$	$3.75_{[3.02e\text{-}2]}$

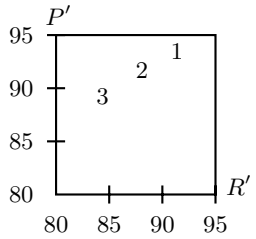


Table 6.2: Bracketing scores PCFG versus MaxEnt CFG.

probabilistic grammar should not be measured by the results of such a rather small set of experiments only. We are confident that the method will show favourable on most real data because of the fact that it does not assume any property of the grammar that is not supported by the data; the method uses all and only the information expressed by the constraints on the property expectations. Moreover the method is a very general one: the features can represent any property of sentences in the corpus; not only the properties that are related to the use of a grammar rule in the syntax tree of a sentences, but also properties that are beyond those expressible in a pure context-free grammar model. The experiment in the next section shows a glimpse of this potential expressiveness.

6.6 Experiment 3

In this section we present an experimental comparison of two probabilistic extensions of unification grammar. One extension, called Probabilistic Unification Grammar (PUG), is based on the probabilistic extension of the context-free backbone. The other extension, MaxEnt UG, is based on the MaxEnt method. In this case, the event space is the set of feature structures that can be generated by the grammar we inferred from the SCHISMA Treebank. This means that we have to define properties of feature structures that characterise the feature structures that can be generated by the grammars that we infer from the tree-bank. The context-free grammar rules and the unification constraints inferred from the tree-bank entail the tree-bank feature structures. Although properties may detect any structure within feature structures, for practical reasons we restrict properties to unification constraints. Each unification constraint is a property to be included in the probabilistic model. As a consequence, two types of properties can be distinguished: *coreference* properties and *constant* properties. An example of a constant property:

```
<N head agr num> = pl
```

and an example of a coreference property is

measure	(3)**UG**	(1)**PUG**	**MaxEnt** (2)**UG**
R	$92.7_{[4.80e\text{-}2]}$	$94.9_{[6.69e\text{-}2]}$	$92.9_{[5.96e\text{-}2]}$
R'	$83.4_{[1.93e\text{-}1]}$	$89.6_{[19.3e\text{-}1]}$	$84.2_{[1.11e\text{-}1]}$
P	$94.3_{[1.15e\text{-}2]}$	$96.2_{[2.30e\text{-}2]}$	$93.8_{[5.43e\text{-}2]}$
P'	$88.0_{[1.29e\text{-}1]}$	$91.1_{[1.94e\text{-}1]}$	$88.0_{[1.11e\text{-}2]}$
C	$3.83_{[4.88e\text{-}2]}$	$2.31_{[3.71e\text{-}2]}$	$3.96_{[3.50e\text{-}2]}$

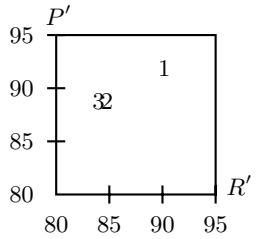


Table 6.3: Bracketing scores PUG versus MaxEnt UG.

```
<NP head agr> = <VP head agr>
```

A nice thing about selecting properties according to this scheme, is that each grammar rule that is applied adds some properties to the feature structure, and that these properties depend on the constraints of the rule directly. In practice, for each partial parse a set of properties is kept. Each application of a rule is expressed in terms of properties by set union. As can be seen in table 6.3 we obtain superior performance for the PCFG model of unification grammar. Although theoretically wrong, the PCFG extension of unification shows a considerable performance improvement over plain unification grammar, and over MaxEnt unification grammar. An explanation of these good results for a wrong model might be that unification failure does not harm the expected frequency with which a rule will be applied too much.

If we look a bit closer we see that MaxEnt UG has a slightly increased recall rate, but that it has to pay for that with a worse performance on precision. It is extremely difficult to improve recall and precision rates that are already very high. Each bracketing that is added due to a changed behaviour of the parser, may improve recall. If it improves recall, that is good. However, if the added bracketing is wrong, it will decrease the precision rate, and possibly increase the number of crossing brackets. This is probably what happens in the case of the MaxEnt model of unification grammar.

We believe a bad choice of properties is the main reason for this behaviour. The properties we have chosen have difficulties capturing context-sensitive phenomena: they are local to the grammar rules, and as we did not express potentially context-sensitive information in the constraints, the probabilistic model cannot exploit any context-sensitive properties. Unfortunately, we currently do not dispose of software to test this hypothesis empirically. We defer this to future research. For one thing, the experiments should not be taken as a motivation for abandoning the MaxEnt approach.

6.7 Discussion

To interpret the results of our experiments we have to keep the following in mind: the task of parsing sentences from the Wizard of Oz corpus is a peculiar one for a number of reasons:

- We used a rather small domain-dependent tree-bank for our experiments. We think we have to be careful with generalisations and comparisons to other parsing tasks, specifically parsing tasks in other domains.
- The average length of the sentences is approximately 5.4 which is rather short.
- The annotation, and therefore the grammars that we derive is flat; this, together with the length of the sentences, explain partly the high precision and recall scores.
- The property functions of the MaxEnt model that we have chosen do not exploit the full potential.

To value the parsing system we developed we have to remember the original goal of developing a robust parsing system for integration in the SCHISMA dialogue system. Assuming that our annotation scheme is rich enough for the SCHISMA domain, taking into account the results presented in the previous sections, we can conclude that we managed to develop such a system.

A comparison of our experimental results to related work is a bit difficult. As far as we know, experimental evaluation of probabilistic extensions of unification grammar based on MaxEnt models has not been performed. In chapter 2 we have seen some related probabilistic extensions of unification grammar. We mention here probabilistic feature grammar, probabilistic LR parsing with unification grammars, probabilistic CUF, and probabilistic ALE. Moreover our application domain and the tree-bank are too specialised to compare our results to that of experiments with tree-banks like the Lancaster Treebank or the Penn Treebank.

6.8 Summary

In this chapter we have presented our probabilistic parsing system, and experiments with grammars generated from the SCHISMA Treebank. We saw that:

- plain context-free grammars perform better than plain unification grammars;
- probabilistic context-free grammars perform better than plain context-free grammars and better than probabilistic context-free grammars based on the MaxEnt method;
- the probabilistic extension of unification grammars based on rule frequencies performs better than the extension based on the MaxEnt method;

Summarising the results of our experiments, we conclude that probabilistic extensions of grammars obtained from the SCHISMA Treebank perform better than their non-probabilistic versions.

Part III

Epilogue

Chapter 7
Conclusions

The main conclusion of this thesis is that the extension of context-free grammars and unification grammars with probabilistic information, even if this information is based on little data, can improve the performance of a syntactic parser in a domain like that of SCHISMA. The probabilistic extensions based on rule probabilities has shown to perform better than the extensions based on the maximum entropy method.

We have specified a scheme for the annotation of user utterances in the SCHISMA domain, and have annotated a collection of user utterances. We have presented a grammar inference engine that, given a translation specification, generates grammars from language data that is annotated using SGML. For the generation of unification constraints we developed a method based on pattern matching of SGML tags and their attributes.

Parse ranking and statistics in general are not a definitive solution to ambiguity resolution and overgeneration. It helps to make the problem tractable and force interpretation decisions in case of overgeneration. Further it enables a modular architecture of dialogue systems as we proposed in the introduction. The extra-syntactical knowledge otherwise received from the other modules can be captured by statistics.

7.1 Towards integration in a SCHISMA prototype

In our experiments we applied a minimal lexicon managed by the parser itself. In practical applications like SCHISMA we need a wide coverage lexicon like CELEX (Burnage 1990). For an experimental evaluation of our parsing system with a large lexicon it is necessary to work with a pre-processor that performs part-of-speech tagging for the parser, like we proposed in section 1.3 on the architecture of SCHISMA.

For PCFG models finding the most probable parse can be done efficiently by Viterbi search (Viterbi 1967). For MaxEnt models this is less straightforward. The probability of a partial parse tree or a partial feature structure is not guaranteed to be greater than the probability of the final structure; properties may both increase and decrease the probability. It is necessary to find out how the most probable parse can be found in the MaxEnt case. Riezler (1998a)

presents a heuristic approach to the related problem of finding the most probable proof of a query given a constraint logic program.

A more efficient representation of disjunctive feature structures will result in a less memory consuming parser. It may become slower because disjunctive unification has a non-polynomial time complexity (Kasper and Rounds 1986). An approach based on term unification can be found in (Nakano 1993); a graph-based approach using contexted descriptions as developed in (Eisele and Dörre 1990) can be found in (Matiasek 1993).

In chapter 6 we applied recall and precision measures to determine the performance of our parsing system. Although performance on the syntactic level is important, it would have been interesting to see the parsing system's influence on the performance of a dialogue system as a whole. Also, it would be interesting to see how the parsing system performs in a more realistic setting.

7.2 Recommendations for future research

Although we consider the performance of our parsing system satisfactory, we believe that more annotated data will improve the performance. We saw in section 5.6 that the percentage of rules that appeared only once in the tree-bank is 72%. More data should bring down this percentage, and as a consequence result in a more accurate probability distribution over the language of the grammar.

The induction of relevant properties is important for the inference of good distributions. The experiments with MaxEnt models that we presented in the previous chapter were done with a fixed set of properties. It would be interesting to see whether experiments with property induction results in a set of properties that results in a better performance.

Part IV

Appendices

Appendix A
SCHISMA Treebank DTD

Below is the DTD that defines the structure of the SCHISMA Treebank.

```
<!ELEMENT corpus        - -     (utt)+>
<!ATTLIST corpus
          nr            NUMBER                  #REQUIRED>

<!ELEMENT utt           - -     (sent |
                                 punct |
                                 conj)+>
<!ATTLIST utt
          nr            NUMBER                  #REQUIRED
          index         NUMBER                  #REQUIRED>

<!ELEMENT (sent |
           emb)         - -     (np |
                                 pp |
                                 verb |
                                 punct |
                                 conj |
                                 det |
                                 whnp |
                                 emb |
                                 adv |
                                 adj |
                                 name |
                                 number |
                                 ordinal |
                                 wh |
                                 yn |
                                 misc |
                                 iject |
                                 pn |
                                 abbrev |
                                 prep)+ +(typo)>
<!ATTLIST (sent | emb)
          type          (rel |
                         whq |
```

```
                    pn |
                    np |
                    pp |
                    number |
                    ordinal |
                    wh |
                    ynq |
                    misc |
                    yndecl |
                    decl |
                    iject |
                    yn |
                    adj |
                    adv |
                    declinv |
                    num |
                    ord |
                    imp)                    #REQUIRED>

<!ELEMENT (np |
           whnp)        - -     (det |
                                 noun |
                                 np |
                                 pp |
                                 conj |
                                 adv |
                                 iject |
                                 misc |
                                 name |
                                 number |
                                 pn |
                                 cn |
                                 adj |
                                 punct |
                                 ordinal |
                                 wh |
                                 emb)+ +(typo)>
<!ATTLIST (np |
           whnp)
          id            ID                      #IMPLIED
          share         IDREF                   #IMPLIED
          det           (y |
                        n)                      n
          wh            (nil |
                        wat |
                        wie |
                        waar |
                        watvoor |
                        welke |
                        wanneer |
```

```
                    hoelaat |
                    hoeveel |
                    hoe |
                    waarvoor |
                    waarom |
                    waarover)              nil
          number    CDATA                  #IMPLIED>

<!ELEMENT cn        - -     ((noun | name), noun)>

<!ELEMENT pp        - -     (prep, ((wh,adj?) |
                                np |
                                verb |
                                pn |
                                whnp)?)>
<!ATTLIST pp
          id        ID                     #IMPLIED
          prep      (door |
                    other)                 other>

<!ELEMENT noun      - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST noun
          num       (s |
                    p)                     s
          gen       (m |
                    f |
                    n |
                    mf)                    mf>

<!ELEMENT prep      - -     (#PCDATA) +(typo | abbrev)>

<!--
ref is voor typen wederkerend, wederkerig, demo
discont is voor discontinue aanwijzende voornaamwoorden
-->
<!ELEMENT pn        - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST pn
          id        ID                     #IMPLIED
          case      (c1 |
                    c2 |
                    c3 |
                    c4)                    c1
          type      (wd |
                    demo |
                    ob |
                    wg |
                    rel |
                    pers)                  pers
          ref       IDREF                  #IMPLIED
          prev      IDREF                  #IMPLIED
```

```
          discont       (n |
                        1 |
                        2)                    n>

<!ELEMENT adj           - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST adj
          id            ID                    #IMPLIED>

<!ELEMENT adv           - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST adv
          id            ID                    #IMPLIED
          prev          IDREF                 #IMPLIED
          discont       (n |
                        1 |
                        2)                    n>

<!ELEMENT det           - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST det
          id            ID                    #IMPLIED>

<!ELEMENT number        - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST number
          id            ID                    #IMPLIED
          type          (an |
                        alpha |
                        num)                  #REQUIRED>

<!ELEMENT ordinal       - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST ordinal
          type          (an |
                        alpha |
                        num)                  #REQUIRED>

<!ELEMENT name          - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST name
          quote         (qs |
                        qd |
                        qn)                   qn
          caps          (cy |
                        cn)                   cn
          part          (py |
                        pn)                   pn>


<!ELEMENT wh            - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST wh
          id            ID                    #IMPLIED
          prev          IDREF                 #IMPLIED
          type          (wat |
                        wie |
```

```
                        waar |
                        watvoor |
                        welke |
                        wanneer |
                        hoelaat |
                        hoeveel |
                        hoe |
                        waarvoor |
                        waarom |
                        waarover)               #IMPLIED
          discont       (n | 1 | 2)             n>

<!ELEMENT verb          - -     (#PCDATA) +(typo | abbrev)>
<!--
subj: echte onderwerp
lsubj: loos onderwerp: [het] is een voorstelling die ..
psubj: plaatsonderwerp: [er] is een plaats vrij voor ...
indobj: indirect object, meewerkend voorwerp
dirobj: direct object, lijdend voorwerp
-->
<!ATTLIST verb
          subj          IDREF                   #IMPLIED
          psubj         IDREF                   #IMPLIED
          lsubj         IDREF                   #IMPLIED
          dirobj        IDREF                   #IMPLIED
          indobj        IDREF                   #IMPLIED
          nwg           IDREF                   #IMPLIED
          id            ID                      #IMPLIED
          prev          IDREF                   #IMPLIED
          main          IDREF                   #IMPLIED
          pass          (py |
                        pn)                     pn
          discont       (n |
                        1 |
                        2)                      n
          per           (p1 |
                        p2 |
                        p3 |
                        pernil)                 pernil
          num           (u |
                        p |
                        s |
                        numnil)                 numnil
          type          (aux |
                         main)                  main
          form          (fin |
                        inf |
                        part |
                        imp)                    fin>
```

```
<!ELEMENT conj          - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST conj
          type          (co |
                        sub)            sub>

<!ELEMENT punct         - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST punct
          id            ID                      #IMPLIED
          prev          IDREF                   #IMPLIED
          discont       (n |
                        1 |
                        2)                      n>

<!ELEMENT yn            - -     (#PCDATA) +(typo | abbrev)>

<!ELEMENT iject         - -     (#PCDATA) +(typo | abbrev)>
<!ATTLIST iject
          type          (thanx |
                        greet)                  greet>

<!ELEMENT misc          - -     (#PCDATA) +(typo | abbrev)>

<!ELEMENT typo          - -     (#PCDATA) +(noun | number | verb |
                                            np | det)>
<!ATTLIST typo
          type          (del |
                        subst |
                        flip |
                        ins |
                        unknown)                unknown>
```

Appendix B
Meta-constraints

In this appendix we give the specification of the meta-constraints we used for inferring unification grammars from the SCHISMA Treebank. It has five sections, each section conforms to a meta-constraint type as defined in chapter 5. Comments start with a #.

```
# Lexical constraints
<lexlex>
PUNCT   DISCONT discont$val
VERB    FORM    form$val
VERB    DISCONT discont$val
VERB    PASS    passive$val
ADV     DISCONT discont$val
WH      DISCONT discont$val
PN      DISCONT discont$val
PN      CASE    case$val
PN      TYPE    type$val
NOUN    NUM     num$val
NOUN    GEN     gen$val
VERB    NUM     num$val
VERB    GEN     gen$val
VERB    TYPE    type$val
</lexlex>

# RHS nonterminal constraints
<lexgram>
PUNCT   DISCONT *       <$0 head discont>=$val
VERB    TYPE    MAIN    <$0 head vtype>=$val
                        <$lhs pred name>=<$0 head gloss>
VERB    TYPE    AUX     <$0 head vtype>=$val
VERB    PASS    PY      <$0 head passive>=$val
                        <$lhs pred AGENT>=<$lhs pred doorobj>
                        <$lhs pred dirobj>=<$lhs pred subj>
VERB    PASS    PN      <$0 head passive>=$val
VERB    DISCONT *       <$0 head discont>=$val
VERB    FORM    *       <$0 head form>=$val
ADV     DISCONT *       <$0 head discont>=$val
WH      DISCONT *       <$0 head discont>=$val
```

```
PN      DISCONT *       <$0 head discont>=$val
PN      CASE    *       <$0 head agr case>=$val
PN      TYPE    *       <$0 head type>=$val
CONJ    TYPE    *       <$0 head type>=$val
PP      PREP    DOOR    <$lhs pred doorobj>=<$0 head>
</lexgram>

# Identifier attributes
<id>
WHNP    ID
NUMBER  ID
PUNCT   ID
WH      ID
NP      ID
PP      ID
ADV     ID
ADJ     ID
VERB    ID
PN      ID
DET     ID
</id>

# Referring attributes
<idref>
PUNCT   PREV    <$0 head gloss>=<$1 head gloss>
WH      PREV    <$0 head gloss>=<$1 head gloss>
VERB    PREV    <$0 head gloss>=<$1 head gloss>
ADV     PREV    <$0 head gloss>=<$1 head gloss>
PN      PREV    <$0 head gloss>=<$1 head gloss>
VERB    SUBJ    <$lhs pred subj>=<$1 head>
                <$lhs head>=<$0 head>
                <$0 head agr>=<$1 head agr>
VERB    PSUBJ   <$lhs pred psubj>=<$1 head>
                <$1 head gloss>=er
VERB    LSUBJ   <$lhs pred lsubj>=<$1 head>
VERB    NWG     <$lhs pred nwg>=<$1 head>
                <$0 head cop>=+
                <$1 head agr>=<$lhs pred subj agr>
VERB    DIROBJ  <$lhs pred dirobj>=<$1 head>
VERB    INDOBJ  <$lhs pred indobj>=<$1 head>
</idref>

# General constraints
<propagation>
*       ADJ     NOUN    <$0 head agr>=<$1 head agr>
                        <$1 head modify adj>=<$0 head>
*       NUMBER  NOUN    <$0 head agr>=<$1 head agr>
                        <$1 head modify num>=<$0 head>
*       ADJ     ADJ     <$0 head agr>=<$1 head agr>
                        <$1 head modify adj>=<$0 head>
```

```
*       DET     NOUN    <$0 head agr>=<$1 head agr>
                        <$1 head modify det>=<$0 head>
*       WH      NOUN    <$0 head agr>=<$1 head agr>
                        <$1 head modify wh>=<$0 head>
*       WH      ADJ     <$0 head agr>=<$1 head agr>
                        <$1 head modify wh>=<$0 head>
*       DET     ADJ     <$0 head agr>=<$1 head agr>
                        <$1 head modify det>=<$0 head>
                        <$0 head det>=<$1 head e>
*       NUMBER  ADJ     <$1 head modify number>=<$0 head>
                        <$1 head modify e>=+
*       ADV     ADJ     <$1 head modify adv>=<$0 head>
NP      *       WEEKDAY <$lhs head>=<$1 head>
NP      *       MONTH   <$lhs head>=<$1 head>
NP      *       NAME    <$lhs head>=<$1 head>
NP      >       NOUN    <$lhs head>=<$1 head>
NP      *       CN      <$lhs head>=<$1 head>
NP      *       NP      <$lhs head>=<$1 head>
PP      PREP    NP      <$lhs head>=<$1 head>
                        <$1 head modify prep>=<$0 head>
PP      PREP    VERB    <$lhs head>=<$1 head>
                        <$1 head modify prep>=<$0 head>
PP      PREP    WHNP    <$lhs head>=<$1 head>
                        <$1 head modify prep>=<$0 head>
PP      PREP    WH      <$lhs head>=<$1 head>
                        <$1 head modify prep>=<$0 head>
WHNP    WH      NP      <$lhs head>=<$1 head>
                        <$1 head wh>=<$0 head>
CN      NAME    NOUN    <$lhs head>=<$1 head>
                        <$1 head modify noun>=<$0 head>
CN      NOUN    NOUN    <$lhs head>=<$1 head>
                        <$1 head modify noun>=<$0 head>
*       NOUN    PP      <$0 head modify pp>=<$1 head>
SENT    *       *       <$lhs head type>=$lhs$TYPE
*       SENT    SENT    <$1 head left>=<$0 head>
UTT     >       SENT    <$lhs head>=<$1>
*       SENT    PUNCT   <$1 head left>=<$0 head>
*       PUNCT   SENT    <$1 head left>=<$0 head>
*       SENT    CONJ    <$1 head left>=<$0 head>
*       CONJ    SENT    <$1 head left>=<$0 head>
</propagation>
```

Appendix C
Samenvatting

Dit proefschrift gaat over het ontleden van natuurlijke taal en in het bijzonder over probabilistische uitbreidingen van corpus-gebaseerde grammatica's ten behoeve van syntactische ontleding. Het onderzoek is uitgevoerd in de context van het SCHISMA-project, een project waarin een dialoogsysteem wordt ontwikkeld dat gebruikers de mogelijkheid biedt dialogen aan te gaan in het Nederlands om informatie te verkrijgen over schouwburgvoorstellingen en om plaatskaarten te reserveren.

Het uiteindelijke doel van het onderzoek is de ontwikkeling van een efficiënte en effectieve syntactische ontleder voor het SCHISMA taakdomein. Daartoe hebben wij ons beziggehouden met de acquisitie van grammatica's uit een corpus van geannoteerde taaluitingen. We hebben een methode ontwikkeld voor de afleiding van unificatie-grammatica's in het PATR II formalisme, en voor het uitbreiden van deze grammatica's met probabilistische informatie. De belangrijkste bijdrage van dit onderzoek aan de computationele linguïstiek is het empirisch verkregen resultaat dat de uitbreiding van contextvrije grammatica's en unificatie-grammatica's met probabilistische informatie, zelfs als deze gebaseerd is op een kleine hoeveelheid gegevens, de prestaties van een ontleder in een taakdomein als SCHISMA aanzienlijk kan verbeteren.

Om tot dit resultaat te komen, hebben we allereerst een corpus van taaluitingen geannoteerd met de juiste syntactische structuur door gebruik te maken van de Standard Generalized Markup Language (SGML). Een essentieel onderdeel van annotie met behulp van SGML is de formele specificatie van het annotatieschema in een Document Type Definition (DTD). Zo'n specificatie is nodig voor de validatie van de geannoteerde data, maar speelt ook een belangrijke rol bij het vertalen van de geannoteerde data naar lexicons en grammatica's.

Vervolgens hebben we een methode ontwikkeld voor de automatische generatie van grammatica's uit de geannoteerde data. Deze is gebaseerd op de herkenning van patronen van SGML-markeringen en hun attributen. Patronen en de daarvoor te genereren productie-regels en unificatie-constraints kunnen worden gespecificeerd in een flexibele specificatietaal.

Contextvrije grammatica's kunnen eenvoudig worden uitgebreid door waarschijnlijkheden toe te kennen aan de regels van de grammatica (het 'klassieke model'). Unificatie-grammatica's kunnen op dezelfde manier worden uitgebreid door aan hun contextvrije regels waarschijnlijkheden toe te kennen. Echter,

omdat unificatie-operatie kunnen mislukken, is de kansverdeling die zo'n probabilistische unificatie-grammatica definieert op de taal die zij genereert niet correct. Daarom hebben we een probabilistische uitbreiding gedefinieerd met behulp van de maximum-entropie-methode.

We hebben experimenten uitgevoerd waarbij de geannoteerde data wordt gesplitst in een 'train set' en 'test set'. De grammatica en de eventuele parameters van een probabilistische uitbreiding worden afgeleid uit de train set en getest op de test set. We hebben klassieke uitbreidingen van zowel contextvrije grammatica's als unificatie-grammatica's vergeleken met door ons gedefinieerde uitbreidingen die gebaseerd zijn op de maximum-entropie-methode. Daarbij laten de klassieke uitbreidingen betere resultaten zien, ook al is de klassieke uitbreiding van unificatie-grammatica niet correct. In het algemeen presteren probabilistisch uitgebreide grammatica's beter dan niet-probabilistische.

References

Abney, S. (1996). Statistical methods and linguistics. In J. Klavans and P. Resnik (Eds.), *The Balancing Act*. Cambridge, Massachussetts: The MIT Press.

Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics 23*(4), 597–618.

op den Akker, R. (1988). *Parsing attribute grammars*. Ph. D. thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands.

op den Akker, R. and H. ter Doest (1994). Weakly restricted stochastic grammars. In *Proceedings of the 15th International Conference on Computational Linguistics*, pp. 927–934.

op den Akker, R., H. ter Doest, M. Moll, and A. Nijholt (1995). Parsing in dialogue systems using typed feature structures. In *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague/Karlovy Vary, Czech Republic, pp. 10–11.

Alshawi, H. (Ed.) (1992). *The Core Language Engine*. Cambridge, Massachussetts: The MIT Press.

Andernach, T. (1996). A machine learning approach to the classification and prediction of dialogue utterances. In *Proceedings of the Second International Conference on New Methods in Language Processing*, pp. 98–109.

Andernach, T. and M. van Steenbergen (1994). Domain and dialogue knowledge in a natural language information system. Memoranda Informatica 94-05, Department of Computer Science, University of Twente, Enschede, The Netherlands.

Androutsopoulos, I., G. Ritchie, and P. Thanisch (1995). Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering 1*(1), 29–81.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process. *Inequalities 3*, 1–8.

Berger, A. (1997). The Improved Iterative Scaling algorithm: a gentle introduction. URL: `http://www.cs.cmu.edu/~aberger/maxent.html`.

Berger, A., V. Della Pietra, and S. Della Pietra (1996). A maximum entropy approach to natural language processing. *Computational Linguistics 22*(1), 39–71.

Bertsekas, D. (1982). *Constrained optimization and lagrange multiplier methods*. Computer Science and Applied Mathematics. New York: Academic Press.

Bies, A., M. Ferguson, K. Katz, and R. MacIntyre (1995). Bracketing guidelines for Treebank II style Penn Treebank Project. Technical report, University of Pennsylvania. URL: `http://www.cis.upenn.edu/~treebank/`.

Black, E., F. Jelinek, J. Lafferty, and D. M. Magerman (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the DARPA Speech and Natural Language Workshop, 1992*, pp. 31–37.

BNC (1997). British National Corpus. Technical report, Oxford University. URL: `http://info.ox.ac.uk/bnc/`.

Bod, R. (1996). Efficient algorithms for parsing the DOP model? A reply to Joshua Goodman. `http://xxx.lanl.gov/abs/cmp-lg/9605031`.

Bod, R. (1998). *Beyond grammar. An experience-based theory of language.* CSLI Publications. Cambridge, New York: Cambridge University Press.

Bod, R. and R. Scha (1997). Data-Oriented Language Processing. In S. Young and G. Bloothooft (Eds.), *Corpus-based methods in language and speech processing*, Volume 2 of *Text, Speech and Language Technology*, pp. 137–173. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Booth, T. and R. Thompson (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers C-22*(5), 442–450.

Bradley, N. (1998). *XML Complete*. Reading, MA: Addison-Wesley.

Brew, C. (1995). Stochastic HPSG. In *7th European Chapter of the Association for Computational Linguistics*, pp. 83–89.

Brill, D. (1993). LOOM reference manual version 2.0. Technical report, University of Southern California.

Briscoe, T. and J. Carroll (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics 19*(1), 25–59.

Briscoe, T. and N. Waegner (1992). Robust stochastic parsing using the inside-outside algorithm. In *Proceedings AAAI Workshop on Statistically-based NLP Techniques*, San Jose, CA.

Burnage, G. (1990). *CELEX: A guide for users*. CELEX - Centre for Lexical Information, University of Nijmegen.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Number 32 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Carpenter, B. and G. Penn (1994). Ale 2.0 user's guide. Technical report, Carnegie Mellon University Laboratory for Computational Linguistics, Pittsburgh, PA.

Carroll, J. and T. Briscoe (1992). Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, Cambridge, MA, pp. 33–38.

Carroll, J. and T. Briscoe (1998). A survey of parser evaluation methods. In *Proceedings The Evaluation of Parsing Systems, Workshop at the 1st International Conference on Language Resources and Evaluation*, Granada, Spain.

Charniak, E. (1993). *Statistical Language Learning*. Cambridge, Massachussetts: The MIT Press.

Charniak, E. (1996). Tree-bank grammars. In *Proceedings of the 14th National Conference on Artificial Intelligence*, Volume 2, pp. 1031–1036.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control 2*, 137–167.

CLAWS (1998). Corpus annotation. Technical report, University Centre for Computer Corpus Research on Language (UCREL), Lancaster University, Lancaster, UK. URL: `http://www.comp.lancs.ac.uk/computing/research/ucrel/`.

Darroch, J. and D. Ratcliff (1972). Generalised Iterative Scaling for log-linear models. *The Annals of Statistics 43*(5), 1470–1480.

Davey, B. and H. Priestley (1990). *Introduction to lattices and order*. Cambridge, New York: Cambridge University Press.

Della Pietra, S., V. Della Pietra, and J. Lafferty (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*(4), 380–393.

ter Doest, H. (1994). Stochastic grammars: Consistency and inference. Master's thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands.

ter Doest, H. (1998a). A corpus-based probabilistic unification grammar. See Keller (1998), pp. 25–32. ESSLLI98 workshop.

ter Doest, H. (1998b). `Statistics::MaxEntropy`*, a Perl5 module for Maximum Entropy Modeling*. Enschede, The Netherlands: Department of Computer Science, University of Twente. Comprehensive Perl Archive Network (CPAN).

ter Doest, H. (1998c). *Xpatr - reference manual*. Enschede, The Netherlands: Department of Computer Science, University of Twente.

ter Doest, H., M. Moll, R. Bos, S. van de Burgt, and A. Nijholt (1996). Language engineering in dialogue systems. In *Energy Week Conference & Exhibition, Energy Week Information Management*, Volume 1, pp. 68–79.

Dörre, J. and M. Dorna (1993). CUF, a formalism for linguistic knowledge representation. In J. Dörre (Ed.), *Computational Aspects of Constraint-Based Linguistic Description I*, pp. 3–22. DYANA-2 deliverable R1.2.A.

Dörre, J., M. Dorna, and J. Junger (1994, july). *The CUF User's Manual* (1.6 ed.). Stuttgart, Germany: Institut für maschinelle Sprachverarbeitung (IMS), Universität Stuttgart, Germany.

Eisele, A. (1994). Towards probabilistic extensions of constraint-based grammars. In J. Dörre (Ed.), *Computational Aspects of Constraint-Based Linguistic Description II*, pp. 3–21. DYANA-2 Deliverable R1.2.B.

Eisele, A. and J. Dörre (1990). Feature logic with disjunctive unification. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki, pp. 100–105.

van der Ende, D. (1995). Robust parsing: An overview. Memoranda Informatica 95-03, Department of Computer Science, University of Twente, Enschede, The Netherlands.

Fu, K.-S. and T. Booth (1975). Grammatical inference: Introduction and survey, part 2. *IEEE Transactions on Systems, Man, and Cybernetics 5*(4), 409–423.

Gaizauskas, R. (1995). Investigations into the grammar underlying the Penn Treebank II. Technical Report CS-95-25, Dept. of Comp. Sc., The University of Sheffield, UK.

Genesereth, M. R. (1998). Knowledge interchange format, draft proposed American National Standard. Technical Report NCITS.T2/98-004, Logic Group, Stanford University.

Ginsburg, S. (1966). *The Mathemetical Theory of Context-Free Languages*. McGraw-Hill.

Goldfarb, C. F. (1990). *The SGML handbook*. Oxford: Clarendon Press.

Goodman, J. (1996). Efficient algorithms for parsing the DOP model. In *Proceedings Empirical Methods in Natural Language Processing*, Philadelphia.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pp. 89–100.

Grenander, U. (1967). Syntax-controlled probabilities. Technical report, Division of Applied Mathematics, Brown University, Providence, Rhode Island.

Grice, H. (1975). Logic and conversation. In *Syntax and Semantics*, Volume 3: Speech Acts, pp. 41–58. New York: Academic Press.

Grishman, R., C. MacLeod, and J. Sterling (1992). Evaluating parsing strategies using standardized parse files. In *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing*, Trento, Italy, pp. 156–161.

Herdan, G. (1966). *The advanced theory of language as choice and chance*. Number 4 in Kommunikation und Kybernetik in Einzeldarstellungen. Berlin; Heidelberg; New York: Springer.

van der Hoeven, G., T. Andernach, S. van de Burgt, G. Kruijff, A. Nijholt, A. Schaake, and F. de Jong (1994). SCHISMA: A natural language accessible theatre information and booking system. In *Speech and Language Engineering, Proceedings of the Twente Workshop on Language Technology 8*, pp. 137–149.

Hogenhout, W. R. and Y. Matsumoto (1998). A fast method for statistical grammar induction. *Journal of Natural Language Engineering 4*(3), 191–209.

Hoppe, T., C. Kindermann, and J. J. Quantz (1993). BACK V5: tutorial and manual. Technical Report KIT-report 100, Technische Universität Berlin, Berlin, Deutschland.

Hulstijn, J. (1997). Structured information states - raising and resolving issues. In *Proceedings Munich Workshop on Formal Semantics and Pragmatics of Dialogue (MUNDIAL)*, Munich, Germany, pp. 99–117.

Hulstijn, J., R. Steetskamp, H. ter Doest, S. van de Burgt, and A. Nijholt (1996). Topics in SCHISMA dialogues. In *Dialogue Management in Natural Language Systems, Proceedings of the Twente Workshop on Language Technology 11*, pp. 89–99.

Inui, K., V. Sornlertlamvanich, H. Tanaka, and T. Tokunaga (1997). A new formalization of Probabilistic GLR parsing. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pp. 123–134.

Jaynes, E. (1957). Information theory and statistical mechanics I. *Physical Review 106*, 620–630.

Jaynes, E. (1996). Probability theory: the logic of science. Unpublished manuscript, URL: `http://bayes.wustl.edu/`.

Joshi, A. K. (1987). An introduction to tree adjoining grammars. In Manaster-Ramer (Ed.), *Mathematics of Language*. Amsterdam: John Benjamins.

Joshi, A. K. and Y. Schabes (1992). Tree-adjoining grammars and lexicalised grammar. In M. Nivat and A. Podelski (Eds.), *Tree Automata and Languages*, pp. 409–431. New York: Elsevier.

Kasper, R. T. and W. C. Rounds (1986). A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the ACL*, pp. 257–266.

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

Keller, B. (Ed.) (1998). *Automated Acquisition of syntax and parsing*. ESSLLI98 workshop.

Komen, E. (1995). Evaluation of Natural Language™ for the SCHISMA domain. Memoranda Informatica 95-14, Department of Computer Science, University of Twente, Enschede, The Netherlands.

Kupiec, J. (1992). An algorithm for estimating the parameters of unrestricted hidden stochastic context-free grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, pp. 387–393.

Labrou, Y. and T. Finin (1997). A proposal for a new KQML specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, Baltimore.

Lari, K. and S. Young (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language 4*, 35–56.

Lau, R., R. Rosenfeld, and S. Roukos (1993). Adaptive language modeling using the maximum entropy principle. In *Proceedings of Human Language Technology Workshop*, pp. 108–113.

Lee, E. and L. Zadeh (1969). Note on fuzzy languages. *Information Sciences 1*, 421–434.

Lie, D., J. Hulstijn, R. op den Akker, and A. Nijholt (1998). A transformational approach to natural language understanding in dialogue systems. In *Proceedings Natural Language Processing and Industrial Applications (NLP+IA'98)*, Volume II, pp. 163–168.

Magerman, D. M. (1994). *Natural language parsing as statistical pattern recognition*. Ph. D. thesis, Dept. of Computer Science, Stanford University.

Manning, C. D. and B. Carpenter (1997). Probabilistic parsing using left corner language models. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pp. 147–158.

Mark, K., M. Miller, U. Grenander, and S. Abney (1991). Parameter estimation for constrained context-free language models. In *Proceedings of the DARPA Speech and Natural Language Workshop, 1991*, San Mateo, CA, pp. 146–149. Morgan Kaufmann.

Matiasek, J. (1993). Structure sharing unification of disjunctive feature descriptions. In H. Trost (Ed.), *Feature formalisms and linguistic ambiguity*, Chapter 6, pp. 93–102. New York: Ellis Horwood.

McConnel, S. (1995). *PC-PATR reference manual*. Dallas, Texas, USA: Summer Institute of Linguistics. URL: `http://www.sil.org/ftp/software/`.

Moll, M. (1995). Head-corner parsing using typed feature structures. Master's thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands.

Nakano, M. (1993). Constraint projection for efficient unification-based parsing. In H. Trost (Ed.), *Feature formalisms and linguistic ambiguity*, Chapter 7, pp. 103–122. New York: Ellis Horwood.

Neal, R. M. (1993). Probabilistic inference using Markov Chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. of Comp. Science, Univ. of Toronto. URL: `www.cs.utoronto.ca/~radford`.

Nederhof, M. (1994). *Linguistic parsing and program transformations*. Ph. D. thesis, Katholieke Universiteit Nijmegen, Nijmegen, The Netherlands.

Nijholt, A. (1988). *Computers and languages, theory and practice*, Volume 4 of *Studies in Computer Science and Artificial Intelligence*. Amsterdam; New York: North-Holland.

Nijholt, A., A. van Hessen, and J. Hulstijn (1998). Speech and language interaction in a (virtual) cultural theatre. In *Proceedings Natural Language Processing and Industrial Applications (NLP+IA'98)*, Volume II, pp. 176–182.

van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics 23*(3), 425–456.

Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pp. 122–127.

Pereira, F. C. N. and D. H. D. Warren (1980). Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence 13*, 231–278.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Ratnaparkhi, A. (1997a). A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*.

Ratnaparkhi, A. (1997b). A simple introduction to maximum entropy models for natural language processing. Technical Report IRCS Report 97-08, University of Pennsylvania, Philadelphia.

Ratnaparkhi, A. (1998). *Maximum entropy models for natural language ambiguity resolution*. Ph. D. thesis, University of Pennsylvania.

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*, pp. 418–424.

Reynar, J. and A. Ratnaparkhi (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings Fifth Conference on Applied Natural Language Processing*, pp. 16–19.

Riezler, S. (1996). Quantitative constraint logic programming for weighted grammar applications. In C. Retor (Ed.), *Logical Aspects of Computational Linguistics (LACL ' 96)*, Number 1328 in Lecture Notes in Computer Science, pp. 346–365.

Riezler, S. (1998a). *Probabilistic Constraint Logic Programming, formal foundations of quantitative and statistical inference in constraint-based natural language processing*. Ph. D. thesis, Seminar für Sprachwissenschaft, Universität Tübingen.

Riezler, S. (1998b). Statistical inference and probabilistic modeling for constraint-based nlp. In B. Schröder, W. Lenders, W. Hess, and T. Portele (Eds.), *Computers, Linguistics, and Phonetics between Language and Speech. Proceedings of the 4th Conference on Natural Language Processing (KONVENS98)*, pp. 111–124.

Riezler, S. (1998c). Statistical inference for probabilistic constraint logic programming. In H. Wiklicky and A. di Pierro (Eds.), *Probabilistic logic and randomised computation*, pp. 1–10. ESSLLI98 workshop.

Rosenfeld, R. (1994). *Adaptive statistical language modeling*. Ph. D. thesis, Carnegie Mellon University, School of Computer Science, Pittburgh, PA.

Rosenkrantz, D. and P. Lewis-II (1970). Deterministic left corner parsing. In *IEEE Conference of the 11th Annual Symposium on Switching and Automata Theory*, pp. 139–152.

Salomaa, A. (1969). Probabilistic and weighted grammars. *Information and Control 15*, 529–544.

Sampson, G. (1994). SUSANNE: A domesday book of English grammar. In N. Oostdijk and P. de Haan (Eds.), *Corpus-based research into language: in honour of Jan Aarts*, Chapter 11, pp. 169–187. Amsterdam: Rodopi.

Santorini, B. (1991). Bracketing guidelines for the Penn Treebank Project. Technical report, University of Pennsylvania. URL: `http://www.cis.upenn.edu/~treebank/`.

Santorini, B. (1995). Part-of-Speech tagging guidelines. Technical report, University of Pennsylvania. URL: `http://www.cis.upenn.edu/~treebank/`.

Schabes, Y. (1992). stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, pp. 426–432.

Schabes, Y. and R. C. Waters (1993). Stochastic lexicalized tree-insertion grammar. In *Proceedings of the Third International Workshop on Parsing Technologies*, pp. 257–265.

Schabes, Y. and R. C. Waters (1994). Tree-insertion grammar: a cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. Technical Report 94-13, Mitsubishi Electric Research Labs, Cambridge MA.

See-Kiong and M. Tomita (1991). Probabilistic LR parsing for general context-free grammars. In *Proceedings of the Second International Workshop on Parsing Technologies*, Cancun, pp. 154–163.

Shannon, C. (1948). A mathematical theory of communication. *Bell Systems Technical Journal 27*, 379–423.

Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. Stanford, CA: Center for the Study of Language and Information, Stanford University.

Shieber, S. M. (1992). *Constraint-Based Grammar Formalisms : Parsing and Type Inference for Natural and Computer Languages*. Cambridge, MA: MIT Press.

Shieber, S. M., Y. Schabes, and F. C. N. Pereira (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming 24*(1–2), 3–36.

Sikkel, K. (1997). *Parsing Schemata. A framework for specification and analysis of parsing algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Berlin; Heidelberg; New York: Springer.

Sikkel, K. and R. op den Akker (1993). Predictive head-corner chart parsing. In *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg (The Netherlands), Durbuy (Belgium), pp. 267–275.

Skut, W. and T. Brants (1998). A maximum-entropy partial parser for unrestricted text. In *Proceedings of the Sixth Workshop on Very Large Corpora*.

Sobol', I. M. (1994). *A primer for the Monte Carlo Method*. London: CRC Press.

Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. The Systems Programming Series. Reading, MA: Addison-Wesley.

Suppes, P. (1972). Probabilistic grammars for natural languages. In *Semantics for Natural Language*, pp. 741–762. Dordracht-Holland: D. Reidel Publishing Company.

Veldhuijzen van Zanten, G. and R. op den Akker (1994). Developing natural language interfaces: a test case. In L. Boves and A. Nijholt (Eds.), *Twente Workshop on Language Technology 8: Speech and Language Engineering*, pp. 121–135.

Verlinden, M. (1993a). Head-corner parsing of unification grammars: a case study. In *Twente Workshop on Language Technology 8: Speech and Language Engineering*, pp. 71–84.

Verlinden, M. (1993b). Ontwerp en implementatie van een head-corner ontleder voor grammatica's met feature structures. Master's thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands. In Dutch.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. *IEEE Transactions on Information Theory 13*, 260–267.

Wechler, W. (1992). *Universal algebra for computer scientists*, Volume 25 of *EATCS Monographs on Theoretical Computer Science*. Berlin; Heidelberg; New York: Springer.

Wetherell, C. (1980). Probabilistic languages: A review and some open questions. *ACM Computing Surveys 12*(4), 361–379.

Winkler, G. (1995). *Image analysis, random fields and dynamic Monte Carlo methods, a mathematical introduction*. Number 27 in Applications of Mathematics. Berlin; Heidelberg; New York: Springer.

Wintner, S. (1997). *An abstract machine for unification grammar; with applications to an HPSG grammar for Hebrew*. Ph. D. thesis, Technion, Israel Institute of Technology.

Wood, D. (1995). Standard Generalized Markup Language: Mathematical and philosophical issues. *Lecture Notes in Computer Science 1000*, 344–365.

Woods, W. and J. Schmolze (1992). The KL-ONE family. *Computers and Mathematics with Applications, Special Issue on Semantic Networks in Artificial Intelligence, Part 1 23*(2–5), 133–178.

Wright, J. and E. Wrigley (1991). GLR parsing with probability. In M. Tomita (Ed.), *Generalised LR Parsing*, Chapter 8, pp. 113–128. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Zadeh, L. (1965). Fuzzy sets. *Information and Control 8*, 338–353.

Index

annotation scheme, 76–81
Attribute Logic Engine (ALE), 27
attribute value matrix, *see* feature structure

constraint logic, 30
  probabilistic -, 32
  weighted -, 31
context-free grammar, 15–16
  language of -, 15
  probabilistic -, 17–20

data-oriented parsing, 25
derivation
  leftmost -, 15
  relation, 15, 44
  tree, 16
dialogue system, 3–4
distribution
  reference -, 57
DMRFS, 46
Document Type Definition (DTD), 75

entropy, 55–57
  cross -, 56
  principle of maximum -, 55
  relative -, *see* Kullback-Leibler divergence
  Shannon's -, 55
Extensible Markup Language (XML), 9

Feat, 36
feature structure
  multi-rooted -
    subsumption, 42
feature structure
  multi-rooted -
    substructure, 42
feature structure
  attribute value matrix, 37
  multi-rooted -, 41
    dotted -, 46
    unification, 42
  typed -, 36
    feature graph morphism, 38
    greatest lower bound, 40
    least upper bound, 40
    substructure, 38
    subsumption, 38
    unification, 40
FS, 36

Generalized Iterative Scaling (GIS), 63–64
  computation, 64

history-based grammar, 21

Improved Iterative Scaling (IIS), 65–66
  computation, 65
inside-outside algorithm, 18–19
item, 45
Iterative Maximization (IM), 32

Knowledge Interchange Format, 9
Knowledge Query and Manipulation Language, 9
Kullback-Leibler divergence, 57

likelihood
  principle of maximum -, 57

Monte Carlo sampling, 68
MRFS, 42

overgeneration, 2

parsing
  chart -, 45–46
  head-corner -, 49–50
  left-corner -, 46–49
    probabilistic -, 23
  probabilistic GLR -, 22
probabilistic feature grammar, 29
property, 59–62
  correction -, 63

SCHISMA, 2–3
Standard Generalized Markup Language, 73–75
Standard Generalized Markup Language (SGML), 9
stochastic tree substitution grammar, 25
stochastic variable, 54
  expectation, 55
subsumption
  multi-rooted feature structure, 42
  typed feature structure, 38
  types, 36

THIS, 3
tree adjoining grammar, 24
  stochastic lexicalised -, 24
tree insertion grammar, 25
  stochastic lexicalised -, 25
Type, 36
type hierarchy, 36

unification
  feature structure, 40
  multi-rooted feature structure, 42
unification grammar, 43
  language of -, 44
  probabilistic -, 27
  probabilistic-, 50
  rule, 43

weakly restricted stochastic grammar, 20

Abbreviations

ALE	Attribute Logic Engine
AVG	attribute-value grammar
AVM	attribute-value matrix
CFG	context-free grammar
CG	conceptual graph
CLP	constraint logic program(ming)
CNF	Chomsky Normal Form
CUF	Comprehensive Unification Formalism
CYK	Cocke Younger Kasami
DAG	directed acyclic graph
DCG	definite clause grammar
DMRFS	dotted multi-rooted feature structure
DNF	disjunctive normal form
DOP	data-oriented parsing
DTD	Document Type Definition
ERF	expected rule frequency
FS	feature structure
GIS	Generalized Iterative Scaling
glb	greatest lower bound
GLR	Generalised LR
GPSG	Generalised Phrase Structure Grammar
HC	head-corner
HBG	history-based grammar
HPSG	Head-driven Phrase Structure Grammar
HTML	Hypertext Markup Language
ID/LP	Immediate Dominance/Linear Precedence
IIS	Improved Iterative Scaling
IM	Iterative Maximization
KIF	Knowledge Interchange Format
KL	Kullback-Leibler
KQML	Knowledge Query and Manipulation Language
LALR	look ahead LR

LC	left-corner
LHS	left-hand side
LIG	linear indexed grammar
LTIG	lexicalised tree insertion grammar
lub	least upper bound
MaxEnt	Maximum Entropy
MRFS	multi-rooted feature structure
NLP	natural language processing
NLU	natural language understanding
PCFG	probabilistic context-free grammar
PFG	probabilistic feature grammar
PME	principle of maximum entropy
PoS	part-of-speech
PP	prepositional phrase
PUG	probabilistic unification grammar
QLF	Quasi-Logical Form
RHS	right-hand side
SAVG	stochastic attribute-value grammar
SCSG	stochastic context-free grammar
SGML	Standard Generalized Markup Language
SLIG	stochastic linear indexed grammar
SLR	simple LR
SLTAG	stochastic lexicalised tree-adjoining grammar
SLTIG	stochastic lexicalised tree-insertion grammar
STSG	stochastic tree-substitution grammar
TAG	tree-adjoining grammar
THIS	Theatre Information System
UG	unification grammar
WRSG	weakly restricted stochastic grammar
XML	Extensible Markup Language

Titles in the IPA Dissertation Series

The State Operator in Process Algebra
J. O. Blanco
Faculty of Mathematics and Computing Science, TUE, 1996-1

Transformational Development of Data-Parallel Algorithms
A. M. Geerling
Faculty of Mathematics and Computer Science, KUN, 1996-2

Interactive Functional Programs: Models, Methods, and Implementation
P. M. Achten
Faculty of Mathematics and Computer Science, KUN, 1996-3

Parallel Local Search
M. G. A. Verhoeven
Faculty of Mathematics and Computing Science, TUE, 1996-4

The Implementation of Functional Languages on Parallel Machines with Distrib. Memory
M. H. G. K. Kesseler
Faculty of Mathematics and Computer Science, KUN, 1996-5

Distributed Algorithms for Hard Real-Time Systems
D. Alstein
Faculty of Mathematics and Computing Science, TUE, 1996-6

Communication, Synchronization, and Fault-Tolerance
J. H. Hoepman
Faculty of Mathematics and Computer Science, UvA, 1996-7

Reductivity Arguments and Program Construction
H. Doornbos
Faculty of Mathematics and Computing Science, TUE, 1996-8

Functorial Operational Semantics and its Denotational Dual
D. Turi
Faculty of Mathematics and Computer Science, VUA, 1996-9

Single-Rail Handshake Circuits
A. M. G. Peeters
Faculty of Mathematics and Computing Science, TUE, 1996-10

A Systems Engineering Specification Formalism
N. W. A. Arends
Faculty of Mechanical Engineering, TUE, 1996-11

Normalisation in Lambda Calculus and its Relation to Type Inference
P. Severi de Santiago
Faculty of Mathematics and Computing Science, TUE, 1996-12

Abstract Interpretation and Partition Refinement for Model Checking
D. R. Dams
Faculty of Mathematics and Computing Science, TUE, 1996-13

Topological Dualities in Semantics
M. M. Bonsangue
Faculty of Mathematics and Computer Science, VUA, 1996-14

Algorithms for Graphs of Small Treewidth
B. L. E. de Fluiter
Faculty of Mathematics and Computer Science, UU, 1997-01

Process-algebraic Transformations in Context
W. T. M. Kars
Faculty of Computer Science, UT, 1997-02

A Generic Theory of Data Types
P. F. Hoogendijk
Faculty of Mathematics and Computing Science, TUE, 1997-03

The Evolution of Type Theory in Logic and Mathematics
T. D. L. Laan
Faculty of Mathematics and Computing Science, TUE, 1997-04

Preservation of Termination for Explicit Substitution
C. J. Bloo
Faculty of Mathematics and Computing Science, TUE, 1997-05

Discrete-Time Process Algebra
J. J. Vereijken
Faculty of Mathematics and Computing Science, TUE, 1997-06

A Functional Approach to Syntax and Typing
F. A. M. van den Beuken
Faculty of Mathematics and Informatics, KUN, 1997-07

Ins and Outs in Refusal Testing
A.W. Heerink
Faculty of Computer Science, UT, 1998-01

A Discrete-Event Simulator for Systems Engineering
G. Naumoski and W. Alberts
Faculty of Mechanical Engineering, TUE, 1998-02

Scheduling with Communication for Multiprocessor Computation
J. Verriet
Faculty of Mathematics and Computer Science, UU, 1998-03

An Asynchronous Low-Power 80C51 Microcontroller
J. S. H. van Gageldonk
Faculty of Mathematics and Computing Science, TUE, 1998-04

In Terms of Nets: System Design with Petri Nets and Process Algebra
A. A. Basten
Faculty of Mathematics and Computing Science, TUE, 1998-05

Inductive Datatypes with Laws and Subtyping – A Relational Model
E. Voermans
Faculty of Mathematics and Computing Science, TUE, 1999-01