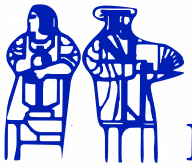
International Workshop on Parsing Technologies

Proceedings of the Conference



IWPT 97

Massachusetts Institute of Technology

INTERNATIONAL WORKSHOP ON PARSING TECHNOLOGY PROCEEDINGS 1997

Organization IWPT'97

General chairman
Harry C. Bunt, ITK, Tilburg University, The Netherlands

Program chairman
Anton Nijholt, University of Twente, The Netherlands

Local chairman
Robert C. Berwick, Massachusetts Institute of Technology, USA

Program committee
Robert C. Berwick, MIT, USA
Harry C. Bunt, Tilburg University, Netherlands
Bob Carpenter, Bell Labs, USA
Eva Hajicová, Charles University, Czech Republic
Mark Johnson, Brown University, USA
Aravind Joshi, University of Pennsylvania, USA
Ronald Kaplan, Rank Xerox, Palo Alto, USA
Martin Kay, Rank Xerox, Palo Alto, USA
Bernard Lang, INRIA, Paris, France
Alon Lavie, Carnegie Mellon University, USA
Makoto Nagao, Kyoto University, Japan
Anton Nijholt, University of Twente, Netherlands
Mark Steedman, University of Pennsylvania, USA
Masaru Tomita, Fujisawa, Japan
K. Vijay-Shanker, University of Delaware, USA
David Weir, University of Sussex, UK
Kent Wittenburg, GTE Laboratories, USA
Mats Wirén, Telia Research AB, Sweden

Acknowledgements
The program committee members were helped during the refereeing process by the following (sub) referees: John Carroll (University of Sussex, UK), Rieks op den Akker, Peter Asveld and Hugo ter Doest (University of Twente, Netherlands), Erik de Clergerie (INRIA, Fr) and Klaas Sikkel (GMD, Germany). WWW pages for SIGPARSE and IWPT'97 were designed and maintained by Gies Bouwman and Hendri Hondorp at the University of Twente and Charles Yang at MIT. Eric Schol of the University of Twente designed the beautiful IWPT'97 logo. Help with making the proceedings was provided by Hugo ter Doest and Hendri Hondorp.

IWPT would especially like to acknowledge the very generous financial support of Sun Microsystems, Inc., for making this conference and banquet possible.

Preface

IWPT'97 is the fifth in a very successful series of workshops, which have become the major forum for specialists in parsing to meet and discuss the advances in the field. In 1989, Masaru Tomita organized the *International Workshop on Parsing Technology* with an original formula: the first part of the workshop took place at CMU, Pittsburgh; the second part at a secluded conference resort (Hidden Valley). This workshop was considered highly successful, and resulted in the book *Current Issues in Parsing Technologies* (Kluwer, Boston 1991).

Inspired by this experience, Masaru Tomita organized the Second International Workshop on Parsing Technology IWPT'91, in Cancun, Mexico, which was successful as well, though it suffered a little from travel restrictions caused by the Gulf War. By then, it seemed clear that these workshops could form an attractive biannual event, and Masaru Tomita and I together organized the third IWPT in 1993, reusing the original '89 formula by having the first part of the workshop at university premises (Tilburg University, the Netherlands), and the second part in a conference resort, in the Belgian Ardennes. IWPT'93 has lead to the follow-up volume *Recent Advances in Parsing Technology*, Harry Bunt and Masaru Tomita, eds., published by Kluwer in 1996.

In 1994, Masaru Tomita left the arena of natural language processing and moved into biogenetical engineering, applying and extending NL parsing techniques to DNA structures. By that time, the Special Interest Group on Parsing (SIGPARSE) had been set up within the Association for Computational Linguistics, with the primary aim to give continuity to the IWPT series.

Under the auspices of SIGPARSE, Eva Hajicova and I organized IWPT'95 in the Czech Republic, with a part at the historical site of Charles University in Prague, and a part in a conference hotel in the thermal resort of Karlovy Vary (Karlsbad). Moving back to the New World, IWPT'97 takes place at one of the world's centers of research in technology: at MIT in Boston.

I would like to use this occasion to thank the people primarily responsible for realizing IWPT'97. Thanks go to the members of the Program Committee, and in particular to chairman Anton Nijholt, for the careful work in reviewing submitted papers and designing the workshop program. Thanks also to Anton Nijholt and his associates at the University of Twente (the Netherlands) for producing the workshop proceedings manuscript. Bob Berwick and his associates at MIT deserve all the credit for taking care of the necessary local arrangements to make IWPT'97 physically possible.

Harry Bunt
SIGPARSE Officer and IWPT'97 General Chair

Contents

Organization iii

Preface v

Contents vii

Author Index ix

Invited Talks xi

The Computation of Movement
Sandiway Fong xiii

Parsing Technology and RNA Folding: a Promising Start
Fabrice Lefebvre xv

Intelligent Multimedia Information Access
Mark T. Maybury xvii

Making Use of Intonation in Interactive Dialogue Translation
Mark Steedman xix

Papers 1

Disambiguating with Controlled Disjunctions
Philippe Blache 1

Encoding Frequency Information in Lexicalized Grammars
John Carroll, David Weir 8

Towards a Reduced Commitment, D-Theory Style TAG Parser
John Chen, K. Vijay-Shanker 18

Controlling Bottom-Up Chart Parsers through Text Chunking
Fabio Ciravegna, Alberto Lavelli 30

Pruning Search Space for Parsing Free Coordination in Categorial Grammar
Crit Cremers 42

Bilexical Grammars and a Cubic-time Probabilistic Parser
Jason Eisner 54

Automaton-based Parsing for Lexicalised Grammars
Roger Evans, David Weir 66

From Part of Speech Tagging to Memory-based Deep Syntactic Analysis
Emmanuel Giguet, Jacques Vergne 77

Probabilistic Feature Grammars
Joshua Goodman 89

Message-passing Protocols for Real-world Parsing - An Object-oriented Model and its Preliminary Evaluation
Udo Hahn, Peter Neuhaus, Norbert Broeker 101

Probabilistic Parse Selection based on Semantic Cooccurrences
Eirik Hektoen 113

A New Formalization of Probabilistic GLR Parsing
Kentaro Inui, Virach Sornlertlamvanich, Hozumi Tanaka, Takenobu, Tokunaga . . 123
Efficient Parsing for CCGs with Generalized Type-raised Categories
Nobo Komagata 135
Probabilistic Parsing using Left Corner Language Models
Christopher Manning, Bob Carpenter 147
Regular Approximations of CFLs: A Grammatical View
Mark-Jan Nederhof 159
A Left-to-right Tagger for Word Graphs
Christer Samuelsson 171
Parsing by Successive Approximation
Helmut Schmid 177
Performance Evaluation of Supertagging for Partial Parsing
B. Srinivas 187
An Earley Algorithm for Generic Attribute Augmented Grammars and Applications
Frederic Tendeau 199
A Case Study in Optimizing Parsing Schemata by Disambiguation Filters
Eelco Visser 210
New Parsing Method using Global Association Table
Juntae Yoon, Seonho Kim, Mansuk Song 225

Posters **237**
Constraint-driven Concurrent Parsing Applied to Romanian VP
Liviu Ciortuz 239
Robustness and Efficiency in AGFL
Caspar Derksen, Cornelis H.A. Koster, Erik Oltmans 241
Language Analysis in SCHISMA
Danny Lie, Joris Hulstijn, Hugo ter Doest, Anton Nijholt 243
Reducing the Complexity of Parsing by a Method of Decomposition
Caroline Lyon, Bob Dickerson 245
Formal Tools for Separating Syntactically Correct and Incorrect Structures
Martin Plátek, Vladislav Kuboň, Tomáš Holan 247
Parsers Optimization for Wide-coverage Unification-based Grammars using the Restriction Technique
Nora La Serna, Arantxa Diaz, Horacio Rodriguez 249

Previous Workshops **251**
IWPT'89 253
IWPT'91 255
IWPT'93 256
IWPT'95 257

Index of Authors

Blache, Philippe 1
Broeker, Norbert* 101
Carroll, John* 8
Carpenter, Bob* 147
Chen, John* 18
Ciravegna, Fabio* 30
Ciortuz, Liviu 239
Cremers, Crit 42
Diaz, Arantxa* 249
Derksen, Capar* 241
Dickerson, Bob* 245
Doest, Hugo ter* 243
Eisner, Jason 54
Evans, Roger* 66
Fong, Sandiway xiii
Giguet, Emmanuel* 77
Goodman, Joshua 89
Hahn, Udo* 101
Hektoen, Eirik 113
Holan, Tomáš* 247
Hulstijn, Joris* 243
Inui, Kentaro* 123
Kim, Seonho* 225
Komagata, Nobo 135
Koster, Cornelis* 241
Kuboň, Vladislav* 247
Lavelli, Alberto* 30
Lefebvre, Fabrice xv
Lie, Danny* 243
Lyon, Caroline* 245
Manning, Christopher* 147
Maybury, Mark T. xvii
Nederhof, Mark-Jan 159
Neuhaus, Peter* 101
Nijholt, Anton* 243
Oltmans, Erik* 241
Plátek, Martin* 247
Samuelsson, Christer 171
Schmid, Helmut 177
La Serna, Nora* 249
Song, Mansuk* 225
Sornlertlamvanich, Virach* 123
Srinivas, B. 187
Steedman, Mark xix
Tanaka, Hozumi* 123
Tendeau, Frederic 199
Tokunaga, Takenobu* 123
Vergne, Jacques* 77
Vijay-Shanker, K.* 18
Visser, Eelco 210
Weir, David* 8,66
Yoon, Juntae* 225

*co-authors

Invited Talks

The Computation of Movement

Sandiway Fong*
NEC Research Institute, Inc.
4 Independence Way, Princeton NJ 08540. USA
sandiway@research.nj.nec.com

Abstract

A central goal of parsing is to recover linguistic structure for interpretation. One property of language that seems to be prevalent is the so-called *displacement* property. That is, syntactic items commonly appear in places other than where we would normally expect for interpretation. Some examples of phenomena involving displacement include *Wh*-movement, raising, passivization, scrambling, topicalization and focus. As Chomsky (1995) points out, displacement is an irreducible fact about human language that every contemporary theory of language has to address. In the principles-and-parameters framework, it is customary to posit a general movement operation, Move-α, that in concert with conditions on its application serve to link displaced elements with their base positions. In terms of parsing, the task is to decode or unravel the effects of Move-α from the surface order. More specifically, for each element, we have to determine whether that element has been displaced or not, and, if so, determine the original position it was displaced from and reconstruct the path it took — including any intermediate positions or landing sites. In general, each displaced element is said to head a (non-trivial) *chain* with one or more empty categories known as *traces* occupying the positions that it passed through. Note that in such theories, empty categories are not just simple placeholders, but elements with much of the same type and range of syntactic properties displayed by their overt counterparts. For example, empty categories in argument positions, like anaphors and pronouns, participate in binding theory and theta role discharge. Hence, the well-formedness of a given sentence will depend, in general, in recovering both the visible and non-visible parts of syntactic structure.

In this talk, we will describe how PAPPI, a multi-lingual parser for theories in the principles-and-parameters frameworks, deals with the computation of movement chains and empty categories in general. Drawing from implemented examples across a variety of languages, we will discuss the mechanism used to handle standard cases of phrasal movement commonly discussed in the literature such as *Wh*-movement, passivization, raising and verb second (V2) phemomena. We will also describe how this mechanism is adapted to handle instances of argument scrambling in languages like Korean and Japanese. We will also focus our attention on head movement. Here, following Pollock (1989), we will discuss the mechanism used to handle the surface differences in the behaviour of verbal inflection in English and French. Following Pesetsky (1995), we will also discuss the implementation of a theory of double object constructions involving the incorporation of both overt and non-overt prepositions into verbal heads.

Finally, we will describe two recent additions to the movement mechanism in the PAPPI system. Moving towards a theory of goal-driven movement — as opposed to the free movement system implied by Move-α, we will discuss an implementation of Case-driven movement within the VP-shell to handle examples involving focus, backgrounding and topicalization in Turkish. Finally, using examples from English and Turkish, we will discuss the necessity of a mechanism of *reconstruction* that optionally "undoes" or reverses the effects of movement to handle facts involving binding and scope.

References

Chomsky, N.A. *The Minimalist Program*, MIT Press 1995.

*Acknowledgments: The author wishes to thank Aysenur Birturk, David Lebeaux and Piroska Csuri for their invaluable linguistic input.

Pollock, J-Y. Verb movement, universal grammar, and the structure of IP, *Linguistic Inquiry* 20(3). 1989.
Pesetsky, D.M. *Zero Syntax: Experiencers and Cascades.* MIT Press 1995.

Parsing Technology and RNA Folding: a Promising Start

Fabrice Lefebvre
LIX, Ecole Polytechnique
91128 Palaiseau Cedex, FRANCE
lefebvre@lix.polytechnique.fr

Abstract

The determination of the secondary structure of RNAs is a problem which has been tackled by distantly related methods ranging from comparative analysis to thermodynamic energy optimization or stochastic context-free grammars (SCFGs). Because of its very nature (properly nested pairs of bases of a single stranded sequence) the secondary structure of RNAs is well modeled by context-free grammars (CFGs). This fact has been recognized several years ago by people who used context-free grammars as a tool to discover some combinatorial properties of secondary structures. More recently SCFGs were used by several teams (esp. David Haussler's team at UC Santa Cruz) as an effective tool to fold RNAs through Cocke-Younger-Kasami-like parsers. Until 1996, and in the context of RNA folding, CFGs and their derivatives where still considered theoretical tools, barely usable outside the computer scientist lab. The exception of SCFGs seemed promising, with all the hype around Hidden Markov Models and other stochastic methods, but it remained to be confirmed for RNAs longer than 200 bases.

The main obstacle to the use of context-free grammars and parsing technology for RNA folding and other closely related problems is the following : suitable grammars are exponentially ambiguous, and sentences to parse (i.e. RNA or DNA sequences) typically have more than 200 words, and sometimes more than 4000 words. These figures are rather unusual for ordinary parsers or parser generators, because they are mostly used in the context of natural language parsing, and thus do not have to face the same computation problems. Fact is, most people dealing with RNA folding problems were manually writing dynamic programming based tools. This was the case for folding models popularized by Michael Zuker, and based on free energy minimization. This was also the case for folding models based on SCFGs. This was in effect the case for just about every computer method available to fold or align sequences. Parsing sequences was not an issue because it simply seemed too slow, too memory hungry and even unrelated.

In 1995, I showed that S-attribute grammars were perfectly able to handle both the thermodynamic model and the stochastic model of RNA folding. I then introduced a parser generator which was able, given a proper S-attribute grammar, to automatically write an efficient parser based on suitable optimizations of Earley's parsing algorithm. All generated parsers turned out to be faster and less memory hungry than other available parsers for the same exponentially ambiguous grammars and the same sequences. More surprisingly, these parsers also turned out to be faster than hand-written programs based on dynamic programming equations. This was the first proof that improvements in parsing technology may certainly be put to good use in biocomputing problems, and that they shall lead to better algorithms and tools.

While trying to overcome some limitations of SCFGs, I generalized S-attribute grammars to multi-tape S-attribute grammars (MTSAGs). The automata theory counterpart of a MTSAG would be a non-deterministic push-down automaton with several one-way reading heads, instead of a single one-way reading head as it is the case for CFG. Given these MTSAGs, a generalization of the previous single-tape parser generator was the obvious way forward.

Thanks to this new parser generator, I was able to show that most biocomputing models previously based on dynamic programming equations were unified by MTSAGs, and that they were better handled by automatically generated parsers than by handwritten programs. It did not matter whether these models were trying to align sequences, fold RNAs, align folded RNAs, align folded and unfolded RNAs, simultaneously align and fold RNAs, etc. It also turned out that

the way SCFGs and HMMs are currently used may be better pictured, thanks to 2-tape MTSAGs, as the simultaneous alignment and folding of a first special tape, representing the target model, against a second tape, containing the actual sequence. This representation may lead to algorithms which will efficiently learn SCFGs from initially unaligned sequences.

While the current parser generator for MTSAGs is a usable proof of concept, which nevertheless required several months of work, I am quite convinced that there should be better ways than the current algorithm to parse several tapes. There should also exist other generalizations of CFGs which may reveal themselves fruitful. Current results are only promising starting points.

The irony of the story is that HMMs and SCFGs were borrowed by biocomputing people from other fields such as signal or speech analysis. It may very well be the time for these fields to retrofit their own models with current advances in biocomputing such as MTSAGs.

Intelligent Multimedia Information Access

Mark T. Maybury
Advanced Information Systems Center
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730, USA
maybury@mitre.org
http://www.mitre.org/resources/centers/advanced_info/

Abstract

The expansion of the information highway has generated requirements for more effective access to global and corporate information repositories. These repositories are increasingly multimedia, including text, audio (e.g., spoken language, music), graphics, imagery, and video. The advent of large, multimedia digital libraries has turned attention toward the problem of processing and managing multiple and heterogeneous media in a principled manner, including their creation, storage, indexing, browsing, search, visualization, and summarization.

Intelligent multimedia information access is a multidisciplinary area that lies at the intersection of artificial intelligence, information retrieval, human computer interaction, and multimedia computing. Intelligent multimedia information access includes those systems which go beyond traditional hypermedia or hypertext environments and analyze media, generate media, or support intelligent interaction with or via multiple media using knowledge of the user, discourse, domain, world, or the media itself.

Providing machines with the ability to interpret, generate, and support interaction with multimedia artifacts (e.g., documents, broadcasts, hypermedia) will be a valuable facility for a number of key applications such as videoteleconference archiving, custom on-line news, and briefing assistants. These media facilities, in turn, may support a variety of tasks ranging from training to information analysis to decision support.

In this talk I will describe our group's efforts to provide content based access to broadcast news sources, including our use of corpus-based processing techniques to the problems of video indexing, segmentation, and summarization. In addition to better access to content, we also need to concern ourselves with enabling more effective, efficient and natural human computer or computer mediated human-human interaction. This will require automated understanding and generation of multimedia and demand explicit representation of and reasoning about the user, discourse, task and context (Maybury 1993). To this end, I will describe our work in progress that aims to fully instrument the interface and build (automatically and semi-automatically) annotated corpora of human-machine interaction. We believe this will yield deeper and more comprehensive models of interaction which should ultimately enable more principled interface design.

References

Maybury, M. T. (ed.) 1993. *Intelligent Multimedia Interfaces.* Menlo Park: AAAI/MIT Press. (http://www.aaai.org/Publications/Press/Catalog/maybury.html)

Maybury, M. T. (ed.) 1997. *Intelligent Multimedia Information Retrieval.* Menlo Park: AAAI/MIT Press. (http://www.aaai.org:80/Press/Books/Maybury-2/)

Maybury, M., Merlino, A., and Morey, D. 1997. *Broadcast News Navigation using Story Segments*, ACM International Multimedia Conference, Seattle, WA, November 8-14.

Speaker Biography

Mark Maybury received his BA in Mathematics from the College of the Holy Cross in 1986 where he was valedictorian. As a Rotary Scholar at Cambridge University, England he received his M.Phil. in Computer Speech and Language Processing in 1987 and his Ph.D. in Artificial Intelligence in 1991 for his dissertation, "Generating Multisentential Text using Communicative Acts". Mark was awarded an MBA from RPI in 1989. Mark has published over fifty technical and tutorial articles in the area of language generation, multimedia presentation and intelligent multimedia information retrieval. He has given tutorials on intelligent multimedia interfaces at multiple international conferences, including CHI, AAAI, IJCAI, COLING, and ACM Multimedia. He chaired the AAAI-91 Workshop on Intelligent Multimedia Interfaces and the IJCAI-95 Workshop on Intelligent Multimedia Information Retrieval and edited collections of the same (AAAI/MIT Press, 1993 and 1997). Mark is Director of the Bedford Artificial Intelligence Center at the MITRE Corporation and Director of MITRE's Advanced Information Systems Center, where he leads a set of seven strategic technology sections which include the disciplines of intelligent information access, data and knowledge management, intelligent training systems, software understanding, high performance networking, collaborative environments, and distributed computing.

Making Use of Intonation in Interactive Dialogue Translation

Mark Steedman
University of Pennsylvania, USA
steedman@linc.cis.upenn.edu

Abstract

Intonational information is frequently discarded in speech recognition, and assigned by default heuristics in text-to-speech generation. However, in many applications involving dialogue and interactive discourse, intonation conveys significant information, and we ignore it at our peril. Translating telephones and personal assistants are an interesting test case, in which the salience of rapidly shifting discourse topics and the fact that sentences are machine-generated, rather than written by humans, combine to make the application particularly vulnerable to our poor theoretical grasp of intonation and its functions. I will discuss a number of approaches to the problem for such applications, ranging from cheap tricks to a combinatory grammar-based theory of the semantics involved and a syntax-phonology interface for building and generating from interpretations.

Disambiguating with Controlled Disjunctions

Philippe Blache
2LC - CNRS
1361 route des Lucioles
F-06560 Sophia Antipolis
pb@llaor.unice.fr

Abstract

In this paper, we propose a disambiguating technique called *controlled disjunctions*. This extension of the so-called named disjunctions relies on the relations existing between feature values (covariation, control, etc.). We show that controlled disjunctions can implement different kind of ambiguities in a consistent and homogeneous way. We describe the integration of controlled disjunctions into a HPSG feature structure representation. Finally, we present a direct implementation by means of delayed evaluation and we develop an example within the functionnal programming paradigm.

1 Introduction

Ambiguity can affect natural language processing at very different levels: it can be very local (e.g. limited to a feature value) or conversely affect entire syntactic structures. But the general disambiguating process remains the same and rely in particular on contextual information. Unfortunately, there exists very few solutions providing a general account of such a process with a direct and efficient implementation.

In this paper, we propose an approach allowing a general and homogeneous representation of ambiguity and disambiguation relations. This approach constitutes an extension of *named disjunctions* (cf. [Dörre90]) and allows a direct implementation of relations controlling the disambiguation.

This paper is threefold. In the first part, we approach the question of the representation and we situate our method among the most representative ones. We describe in particular the advantages and drawbacks of named disjunctions and show how different phenomena can be described within such a paradigm. In the second part, we propose an analysis of the disambiguating process itself and describe some of the control relations existing between the different parts of the linguistic structure. We integrate the representation of such relations to the named disjunction approach: we call this new technique *controlled disjunction*. The last section presents the implementation of controlled disjunction which uses different delayed evaluation techniques such as coroutining, constraint propagation or residuation.

2 Representing Ambiguity

Ambiguity is generally a problem for NLP, but it can also be conceived as a useful device for the representation of particular linguistic information such as homonymy, valency variations, lexical rules, etc. In this perspective, a general representation is very useful.

2.1 Different needs for representing ambiguity

Ambiguity can be more or less complex according to its scope and there is a general distinction between global and local ambiguities. We can also observe such a difference from a strictly structural point of view: ambiguity can affect the subcomponents of linguistic objects as well as entire structures (or, in a feature structure perspective, atomic as well as complex feature values). This is the reason why we distinguish two fundamental properties: (*i*) the interconnection between different subcomponents of an ambiguous structure and (*ii*) the redundancy of such structures.

	Unit	Language	Ambiguity	Relations between values
(1.a)	*les*	French	$\begin{bmatrix} det \vee pro \\ plur \\ masc \vee fem \end{bmatrix}$	None
(1.b)	*walks*	English	$\begin{bmatrix} noun \\ plur \end{bmatrix} \vee \begin{bmatrix} verb \\ sing \\ 3rd \end{bmatrix}$	Category and number
(1.c)	*mobile*	French	$\begin{bmatrix} noun \\ masc \\ sing \end{bmatrix} \vee \begin{bmatrix} adj \\ masc \vee fem \\ sing \end{bmatrix}$	Category and number
(1.d)	*die*	German	$\begin{bmatrix} det \\ nom \vee acc \\ plu \vee \begin{bmatrix} fem \\ sing \end{bmatrix} \end{bmatrix}$	Gender and number
(1.e)	*den*	German	$\begin{bmatrix} det \\ \begin{bmatrix} acc \\ masc \\ sing \end{bmatrix} \vee \begin{bmatrix} dat \\ plu \end{bmatrix} \end{bmatrix}$	Case, gender, number
(1.f)	suffix *_st*	German	$\begin{bmatrix} 3rd \\ sing \end{bmatrix} \vee \begin{bmatrix} 2nd \\ plu \end{bmatrix}$	Person and number

Figure 1: Examples of interconnection between feature values

Moreover, we can introduce a certain kind of dynamicity: ambiguity affects the objects differently depending on whether it is related to external values or not. More precisely, certain ambiguities have only a minimal effect on the structure to which they belong whereas some others can deeply affect it. A value can be poorly or strongly related to the context and the ambiguities are more or less dynamic according to the density of the entailed relations.

Figure (2.1) shows several ambiguities involving different kind of relations between feature values. In these examples, some ambiguities (e.g. (1.a)) have no effect on the other features whereas some features are strongly interconnected as in (1.c). Let us remark that the feature type has no consequence on such relations (see for example (1.d) and (1.e)).

The second problem concerns redundancy: when an ambiguity affects major feature values such as category, the result is a set of structures as for (1.b): the ambiguity between the verb and the noun involves in this example two completely different linguistic structures. But there are also ambiguities less interconnected with other parts of the structure (e.g. (1.d)). In these cases, a common subpart of the structure can be factorised. This is particularly useful for the representation and the implementation of some descriptive tools such as lexical rules (see [Bredekamp96]).

An efficient ambiguity representation must take into account the interconnection between the different subparts of the structure and allow a factorisation avoiding redundancy.

2.2 Different Representations

The disjunctive representation of ambiguity remains the most natural (see [Karttunen984], [Kay85] or [Kasper90]). However, this approach has several drawbacks. First of all, if an ambiguity affects more than one atomic value, then a classical disjunction can only represent variation between complete structures. In other words, such a representation doesn't allow the description of the relations existing between the features. In the same way, several approaches (see [Maxwell91] [Nakazawa88], [Ramsay90], [Dawar90] or [Johnson90]) propose to rewrite disjunctions as conjunctions (and negations). This method, in spite of the fact that it can allow efficient implementations of some ambiguities, presents the same drawback.

A partial solution, concerning in particular the redundancy problem, can be proposed with the use of *named disjunctions* (noted hereafter ND; also called *distributed disjunctions*). This approach has been described by [Dörre90] and used in several works (see [Krieger93], [Gerdemann95] or [Blache96]). The disjunction here only concerns the variable part of a structure (this allows the information factorisation).

A ND binds several disjunctive formulae with an index (the name of the disjunction). These formulae are ordered and have the same arity. The variation is controlled by a *covariancy* mechanism enforcing the

simultaneous variation of the ND values: when one disjunct in a ND is chosen (i.e. interpreted to true), all the disjuncts occurring at the same rank into the other ND formulae also have to be true. Several NDs can occur in the same structure. Figure (2) presents the lexical entry corresponding to example (1.d). In the following, the shaded indices represent the names of the disjunctions.

(2) $$den = \left[\begin{array}{l}\text{SPEC}\left[\begin{array}{l}\text{CASE}\left\{acc \vee_{\boxed{1}} dat\right\}\\ \text{INDEX}\left[\begin{array}{l}\text{GEN}\left\{masc \vee_{\boxed{1}} -\right\}\\ \text{NUM}\left\{sing \vee_{\boxed{1}} plu\right\}\end{array}\right]\end{array}\right]\end{array}\right]$$

This example shows a particular case of subvariation. Indeed, the *plural dative* determiner stipulates no constraints on the gender. We note this subvariation with an *anonymous* variable.

However, the ND technique also presents some drawbacks. In particular, covariancy, which allows a local representation of ambiguity, is the only way to represent relations between features. Such structures need redundant information as shown in example (3) (cf. [Krieger93]). Moreover, the semantic of the disjunction is lost: this is no longer a representation of a variation between values, but between different set of values.

(3) $$\left[\begin{array}{l}\text{MORPH}\left[\begin{array}{l}\text{STEMM}\\ \text{ENDING}\left\{e \vee_{\boxed{1}} st \vee_{\boxed{1}} t \vee_{\boxed{1}} en\right\}\end{array}\right]\\ \text{SYNSEM ... INDEX}\left[\begin{array}{l}\text{PER}\left\{1 \vee_{\boxed{1}} 2 \vee_{\boxed{1}} \left\{3 \vee_{\boxed{2}} 2\right\} \vee_{\boxed{1}} \left\{1 \vee_{\boxed{1}} 3\right\}\right\}\\ \text{NUM}\left\{sing \vee_{\boxed{1}} sing \vee_{\boxed{1}} \left\{sing \vee_{\boxed{2}} plu\right\} \vee_{\boxed{1}} plu\right\}\end{array}\right]\end{array}\right]$$

In the same perspective, covariancy forces a symmetrical relation between two feature values. In fact, there are relations expressing finer selection relations. This is the case in (1.c) in which no covariancy between the part-of-speech and gender features occurs: an *adjective* is either *masculine* or *feminine* whereas a *noun* is always *masculine*. A classical ND representation as in example (3) doesn't account for the fact that the noun selects a masculine value for the gender feature, but the reverse is not true.

3 Disambiguating

Generally, disjunctions are expanded into a disjunctive normal form and the disambiguation comes to a well-formedness verification. In case of incoherence, backtracking is applied and another model is elaborated (i.e. another value is choosen). This method can be improved by some techniques (cf. [Maxwell91]), but the basic mechanism remains a kind of generate-and-test device: the first feature instantiation leads to the choice of an entire structure which has to be validated during the parse.

3.1 Fundamental Needs

A natural solution consists in delaying the evaluation of the structure consistency. We highlight here two techniques used in our approach: selection constraints (cf. [Pulman96]) and coroutining (cf. [van Noord94], [Blache96]).

[Pulman96] represents ambiguities with fixed-arity lists. They are controlled by an agreement-like device between two categories, one being controlled by the other. These lists can be interpreted as disjunctions (not expanded into a normal form) and their evaluation requires contextual information. The problem is that this mechanism can only be represented by a phrase-structure rules, and not at a general level. This is problematic in two respects: it is a context-sensitive mechanism and it only works for formalisms using phrase-structure rules (unlike HPSG for example). Moreover, this approach involves the definition of a new kind of categorial relations (cf. in Pulman's paper the relation between prepositions and nouns) which have no linguistic motivation. Finally, the design of several control relations can rapidly become an exhaustive description of all the dependencies.

A coroutining approach applied to named disjunctions is described in [Blache96]. The idea is to propose the on-line use of NDs relying on (actual) delayed evaluation. The advantage is a direct interpretation of NDs: this representation of ambiguity doesn't use any pre-computation, there are no normal form expansion and the

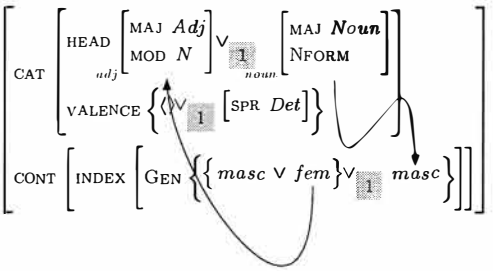
same mechanism is reused for all kind of ambiguities. In this approach, NDs are no longer simple "macros" as described in [Bredekamp96]: there is no interpretation of this formalisation into another one and the evaluation is direct. This is possible because the factorisation allows the construction of an underspecified structure which can be used during the parse. The problem however concerns the disambiguation propagation. More precisely, the only control is that of the covariancy which doesn't adress the problem of embedded NDs.

An efficient treatment of ambiguity would need a factorised representation allowing the use of underspecification together with a fine specification of disambiguating relations. These characteristics are fundamental for an implementation avoiding DNF expansion and delaying disambiguation as much as possible.

3.2 Controlled Disjunctions

The main interest with NDs is the relation between different disjunctive formulae implementing the disambiguation propagation. This propagation supposes an equivalence relation in the sense that any value belonging to a ND can be instantiated and propagates the information to the rest of the ND. But this cannot tackle the problem of directed disambiguation relations as shown in example (4). In this example, we can say that (*i*) if the word is a noun, then the gender is masculine and (*ii*) if the word is feminine, then it is an adjective. But the reverse is not true: we cannot deduce anything if the masculine value is instantiated.

(4)



We propose in the following the introduction of the notion of *controlled disjunctions*. They allow an integration of all the control relations relying on the distinction between *controlling* and *controlled* values within the ND framework.

The first problem being that of covariancy, we identify two kind of disjunctions:

- *Simple disjunctions*: their values can be controlling, controlled or both. They don't belong to a set of disjunctive formulae as for classical NDs but are represented in a separate ND which has its own name and contains this unique formula. This is the example case with the gender value of examples (1.c) or (1.e).
- *Covariant disjunctions*: their values are necessarily both controlling and controlled. These formulae are equivalent to classical NDs.

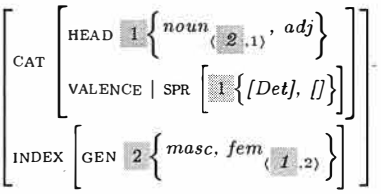
Insofar as covariancy is not the only relation between features, the second problem is the representation of control relations. In a covariant relation, all values occuring at the same rank of a formula are both controlling and controlled. Conversely, the description of non-covariant relations (involving simple disjunctions) relies on the distinction between controlling and controlled values. In example (1.c), the noun value controls (at least) the masculine, and the feminine controls the adjective.

A general solution consists of specifying explicitly all the non-covariant control relations. As said before, *controlled disjunctions* (hereafter CD) are a ND extension in the sense that they are named and their formulae are ordered. So, any value can be referenced by the name of the disjunction and its rank in the formula. In the case of a simple CD, these references specify a unique value whereas for a covariant CD, they specify a set of covariant values.

We note this control relation as: $value_{\langle i,j \rangle}$ where *value* is the name of the controlling value, i is the name of the CD and j is the rank of the controlled value in the formula.

Figure (5) is a representation in an HPSG notation of the example (1.c).

(5)



This structure contains two controlled disjunctions indexed by 1 and 2. Disjunction 1 is *covariant* and the instantiation of one of its value entails that of the other values of the same rank belonging to the same CD. Disjunction 2 is *simple* and specifies two possible values. The control relations are specified by the controlling values. These values can either belong to a covariant or a simple disjunction. In this example, the controlling values are $noun_{\langle 2,1\rangle}$ and $fem_{\langle 1,2\rangle}$, they control respectively the 1st value of the 2nd disjunction (i.e. *masc*) and the 2nd value of the 1st disjunction (i.e. *adj* and all its covariant values).

We can remark that, in the case of typed structures, the elements belonging to a control relation can either be feature values or types: this is the case of the first disjunction which concerns the head types *noun* and *adj*. This property can be very useful for the expression of high-level ambiguities.

The example of figure (6) shows a more complex case with embedded controlled disjunction. But we can observe that the mechanism is the same.

(6)
$$\left[\begin{array}{l} \text{MORPH} \left[\begin{array}{l} \text{STEMM} \\ \text{ENDING}_{\,1} \left\{ \begin{array}{l} e \left[\begin{array}{l} <2,1> \\ <5,1> \end{array}\right], \; st \left[\begin{array}{l} <2,1> \\ <5,1> \end{array}\right], \; t \left[\begin{array}{l} <2,4> \\ <5,3> \end{array}\right], \; en \left[\begin{array}{l} <2,5> \\ <5,2> \end{array}\right] \end{array} \right\} \end{array}\right] \\ \text{SYNSEM ... INDEX} \left[\begin{array}{l} \text{PER}_{\,2} \left\{ 1,\; 2,\; 3_{\,3} \left\{ 3,\; 2_{\,4} \left\{ 1,\; 3 \right\} \right\} \right\} \\ \text{NUM}_{\,5} \left\{ sing,\; sing_{\,3} \left\{ sing,\; plu \right\} \right\} \end{array}\right] \end{array}\right]$$

In another way, simple disjunctions, in addition to their controlling capacities, can be assimilate to finite domains contraints (in the constraint programming sense). Such constraint are very useful for several reasons: they constitute an efficient control on the unification process. Moreover, if no disambiguation can be applied, they propose an approximation of the solution.

4 Delayed Evaluation

Delayed evaluation techniques allow us to maintain the disjunctive structure until disambiguating information is found. This has several advantages. First of all, there are no arbitrary choices as with classical disjunctive approaches relying on normal forms: all instantiations are guided by contextual informations. Moreover, the disambiguation propagation of CD allows us to trigger the process at any level: in the case of HPSG, the disambiguation can come from principles, schemas or even at the lexical level with structure sharing.

We propose in this paper an approach using LIFE, a multi-paradigm programming language (cf. [Aït-Kaci94]) integrating in particular functionnal facilities to a logic programming framework.

A direct implementation of the CD presents two problems: the different kind of relations (covariancy and control) and the control of delaying devices. We propose here a method relying on functions which residuate if their arguments are insufficiently specified. As soon as these arguments are instantiated, their evaluation is fired. This mechanism is very similar to the delayed ones in Prolog (`freeze`, `block`, etc.) and correspond in some way to a constraint programming approach.

We distinguish for the implementation three kind of relations implemented by three different functions. These functions are associated with the values involved in the CD. Two functions implements the controlling and controlled relations and one describes the directed (non-covariant) selection.

The representation used for this relies on the following points:

- disjunctive formulae are represented with lists,
- the name of the CD is represented by an integer variable I,
- I also represents the rank of the instantiated disjunct in the list.

The residuation depends on (*i*) the feature value or (*ii*) the rank of the choosen value in the list. When a feature value can be disambiguated, its instantiation in the list entails the instantiation of its rank. The other values controlled by the same name (i.e. belonging to the same CD) are controlled by a function residuating on the integer variable representing this rank. So, when a value is choosen, all the values of the same rank in the disjunction are also instantiated. The mechanism is the same with directed selections. In this case, we

need a particular function binding two specific values which don't belong to the same CD. These relations are expressed again with the integer variable. To summarize (*i*) if a value belonging to a CD is instantiated, then its rank become known and (*ii*) if the rank is known, then the value can be instantiated. Let us describe more precisely these functions. In the following, the function `cond(a,b,c)` means that if `a` is true, then `b`, otherwise `c`.

- Function *controlling* :
```
controlling(I,A,L) →
      controlling_followed(1,I,A,L).
controlling_followed(J,I,A,[X|L]) →
      cond(A:==X, A | I=J, controlling_followed(J+1,I,A,L)).
```
`I` is an integer representing the rank, `A` the feature value to be instantiated and `L` the list of possible values. This function residuates on `A`. If `A` is known, its evaluation is fired, the function calculates the position of the corresponding value in the list `L` and returns the value itself.

- Function *controlled* :
```
controlled(I,[X|L]) →
      cond(I>0,
           cond(I=:=1,X,controlled(I-1,L)),
           false).
```
The two arguments are the rank (variable `I`) and the list of values (`[X|L]`). The function residuates on `I`. When `I` is known, then the function returns the value corresponding to this position in the list.

- Function *selection* :
```
selection (I,X,Y) →
      cond(I>0,
           cond(I=:=X,Y,true),
           false).
```
`I` represents the position of a controlling value (and the name of the corresponding CD), `X` represents the value that `I` must have to select the controlled value and `Y` represents the rank of the controlled value to be instantiated. This function residuates on `I`. It returns the position of the controlled value in case the controlling one has the right value.

Let us present now the implementation of the lexical entry of the figure (5).

(7)
```
word(mobile,A) :-
        A.phon=mobile,
        A.synsem.loc.cat.head= H ,          ——— Lists
        A.synsem.loc.cat.valence.spr= S ,   ——— Lists
        A.synsem.loc.content.index.gender= G ,   ——— Lists
        H=controlling(I,H,[adj, noun ]),
        S=controlling(I,S,[[],[Det]]),
        G=controlling(J,G,[masc, fem ]),    ——— Control features
        H=controlled(I,[adj,noun]),
        S=controlled(I,[[],[Det]]),
        G=controlled(J,[masc,fem]),
Disjunction
names   I =selection(J,2,1),
        J =selection(I,1,2).
```

This lexical entry bears two CDs: a covariant one controlled by `I` and a simple one controlled by `J`. In this example, all the disjunctive formulae contain values which can be controlled or controlling (this is necessarily the case for covariant CD, but not for the simple ones). So the three corresponding lists are in the scope of both the *controlling* and *controlled* funtions. Moreover, the particular directed relations between `noun/masc` and `fem/adj` are implemented by the *selection* function which indicates for example in its first occurrence that if the second value of the CD J is instantiated, then the CD `I` must instantiate the first value (i.e. if `J=2`, then `I=1`).

5 Conclusion

The *controlled disjunction* technique presented in this paper allows a concise representation of disjunctive information, the description of precise variation phenomena (not only covariancy as with ND) together with an efficient implementation relying on underspecification and delayed evaluation. It consitutes an important

extension of the named disjunction device: its representation of control phenoma between disjunctive values allow the implementation of all kind of ambiguities.

This approach can be improved in several respects and in particular with consistency tests: verification that disjunctions are exclusive, no cycles in the dependency net of relations between disjunctive values, etc.

The implementation of controlled disjunctions is direct using classical delayed evaluation techniques. The example proposed in this paper relies on functional programming, but other logic programming techniques can also be used.

References

[Aït-Kaci94] Hassan Aït-Kaci, Bruno Dumant, Richard Meyer, Andreas Podelski & Peter VanRoy. 1994. *The* Wild LIFE *Handbook*, PRL Research Report.

[Blache96] Philippe Blache. 1996. "Ambiguity, Disjunction and Constraints." In *Proceedings of the Second Conference on New Methods in Natural Language Processing – NeMLap-2*, Ankara, Turkey.

[Bredekamp96] Andrew Bredekamp, Stella Markantonatou & Louisa Sadler. 1996. "Lexical Rules: What are they?" In *Proceedings of COLING'96*, pages 163–168, Copenhaguen, Denmark.

[Dawar90] Anuj Dawar & K. Vijay-Shanker. 1990. "An Interpretation of Negation in Feature Structure Descriptions", in *Computational Linguistics*, 16:1.

[Dörre90] , Jochen Dörre & Andreas Eisele. 1990. " Feature Logic with Disjunctive Unification" in proceedings of *COLING'90.*

[Gerdemann95] Dale Gerdemann. 1995. "Term Encoding of Typed Feature Structures." In *Proceedings of the Fourth International Workshop on Parsing Technologies*, pp. 89–98.

[Johnson90] Mark Johnson. 1990. "Expressing Disjunctive and Negative Feature Constraints with Classical First-Order Logic" in proceedings of *ACL'90.*

[Kay85] Martin Kay. 1985. "Parsing in FUG" in Dowty, Karttunen & Zwicky (eds), *Natural Language Parsing*, Cambridge University Press.

[Karttunen984] Lauri Karttunen. 1984. "Features and Values" in proceedings of *COLING'84.*

[Kasper90] Robert Kasper & William Rounds 1990. "The Logic of Unification in Grammar" in *Linguistics and Philosophy*, 13:1.

[Krieger93] Hans-Ulrich Krieger & John Nerbonne. 1993. "Feature-Based Inheritance Networks for Computational Lexicons." In T. Briscoe, V. de Paiva and A. Copestake, editors, *Inheritance, Defaults and the Lexicon.* Cambridge University Press, Cambridge, USA.

[Maxwell91] John T. Maxwell III & Ronald M. Kaplan. 1991. "A Method for Disjunctive Constraints Satisfaction." In M. Tomita, editor, *Current Issues in Parsing Technology.* Kluwer Academic Publishers, Norwell, USA.

[Nakazawa88] Tsuneko Nakazawa, Laura Neher & Erhard Hinrichs. 1988. "Unification with Disjunctive and Negative Values for GPSG Grammars" in proceedings of *ECAI'88.*

[Pulman96] Stephen G. Pulman. 1996. "Unification Encodings of Grammatical Notations." *Computational Linguistics*, 22:3.

[Ramsay90] Allan Ramsay. 1990. "Disjunction without Tears" in *Computational Linguistics*, 16:3.

[van Noord94] Geert-Jan van Noord & Gosse Bouma. 1994 "Adjuncts and the Processing of Lexical Rules" in proceedings of *COLING'94.*

ENCODING FREQUENCY INFORMATION IN LEXICALIZED GRAMMARS

John Carroll **David Weir**
School of Cognitive and Computing Sciences
University of Sussex
Falmer, Brighton, BN1 9QH, UK

{johnca,davidw}@cogs.susx.ac.uk

Abstract

We address the issue of how to associate frequency information with lexicalized grammar formalisms, using Lexicalized Tree Adjoining Grammar as a representative framework. We consider systematically a number of alternative probabilistic frameworks, evaluating their adequacy from both a theoretical and empirical perspective using data from existing large treebanks. We also propose three orthogonal approaches for backing off probability estimates to cope with the large number of parameters involved.

1 Introduction

When performing a derivation with a grammar it is usually the case that, at certain points in the derivation process, the grammar licenses several alternative ways of continuing with the derivation. In the case of context-free grammar (CFG) such nondeterminism arises when there are several productions for the nonterminal that is being rewritten. Frequency information associated with the grammar may be used to assign a probability to each of the alternatives. In general, it must always be the case that at every point where a choice is available the probabilities of all the alternatives sum to 1. This frequency information provides a parser with a way of dealing with the problem of ambiguity: the parser can use the information either to preferentially explore possibilities that are more likely, or to assign probabilities to the alternative parses.

There can be many ways of associating frequency information with the components making up a grammar formalism. For example, just two of the options in the case of CFG are: (1) associating a single probability with each production that determines the probability of its use wherever it is applicable (i.e. Stochastic CFG; SCFG (Booth and Thompson, 1973)); or (2) associating different probabilities with a production depending on the particular nonterminal occurrence (on the RHS of a production) that is being rewritten (Chitrao and Grishman, 1990). In the latter case probabilities depend on the context (within a production) of the nonterminal being rewritten. In general, while there may be alternative ways of associating frequency information with grammars, the aim is always to provide a way of associating probabilities with alternatives that arise during derivations.

This paper is concerned with how the kind of frequency information that would be useful to a parser can be associated with lexicalized grammar formalisms. To properly ground the discussion we will use Lexicalized Tree Adjoining Grammar (LTAG) as a representative framework, although our remarks can be applied to lexicalized grammar formalisms more generally. We begin by considering the derivation process, and, in particular, the nature of derivation steps. At the heart of a TAG is a finite set of trees (the elementary trees of the grammar). In an LTAG these trees are 'anchored' with lexical items and the tree gives a possible context for its anchor by providing a structure into which its complements and modifiers can be attached. For example, Figure 1 shows four elementary trees—one *auxiliary* tree β and three *initial* trees α_1, α_2 and α_3. Nodes marked with asterisks and downarrows are foot and substitution nodes, respectively. In a derivation these trees are combined using the operations of substitution and adjunction to produce a derived tree for a complete sentence. Figure 2 shows a single derivation step in which α_2 and α_3 are substituted at frontier nodes (with addresses 1 and $2 \cdot 2$, respectively) of α_1 and β is adjoined at an internal node of α_1 (with address 2)[1].

[1] The root of a tree has the address ϵ. The ith daughter (where siblings are ordered from left to right) of a node with address a has address $a \cdot i$

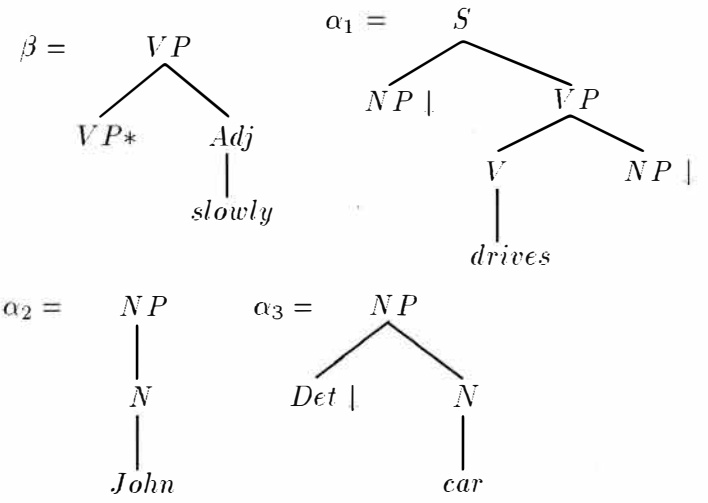


Figure 1: An Example Grammar

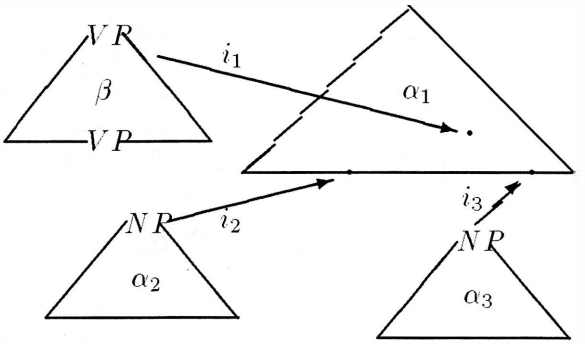


Figure 2: A Derivation Step

When formalizing LTAG derivations, a distinction must be made between the (object-level) trees that are derived in a derivation and the (meta-level) trees that are used to fully encode what happens in derivations. These trees are referred to as derived and derivation trees, respectively. A scheme for encoding TAG derivations was proposed by Vijay-Shanker (1987) and later modified by Schabes and Shieber (1994). Derivation trees show, in a very direct way, how the elementary trees are combined in derivations. Nodes of the derivation trees are labeled by the names of elementary trees, and edge labels identify tree addresses (i.e. node locations) in elementary trees. Figure 3 shows the derivation tree resulting from the derivation step in Figure 2. The nodes identified in the derivation tree encode that when the elementary tree α_1 was used, the elementary trees $\beta, \alpha_2, \alpha_3$ were chosen to fit into the various complement and modifier positions. These positions are identified by the tree addresses i_1, i_2, i_3 labeling the respective edges, where in this example $i_1 = 2$, $i_2 = 1$ and $i_3 = 2 \cdot 2^2$. In other words, this derivation tree indicates which choice was made as to how the node α_1 should be *expanded*. In general, there may have been many alternatives since modification is usually optional and different complements can be selected.

By identifying the nature of nondeterminism in LTAG derivations we have determined the role that frequency information plays. For each elementary tree of the grammar, frequency information must somehow determine how the probability mass is to be distributed among all the alternative ways of expanding that tree. In section 2 we consider a number of ways in which this frequency information can be associated with a grammar. We then go on to evaluate the degree to which each scheme can, in principle, distinguish the probability of certain kinds of derivational phenomena, using data from existing large treebanks (section 3). We discuss in section 4 how

[2] As Schabes and Shieber (1994) point out matters are somewhat more complex that this. What we describe here more closely follows the approach taken by Rambow, Vijay-Shanker, and Weir (1995) in connection with D-Tree Grammar.

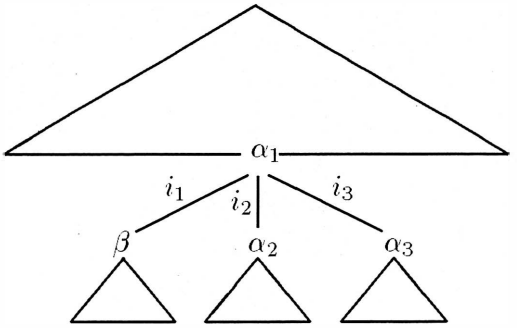


Figure 3: A Derivation Tree

to estimate the large number of probabilistic parameters involved, and propose three orthogonal approaches for smoothing the probability estimates obtained. The paper concludes (section 5) with comparisons with other related work.

2 Frequency Information in Lexicalized Grammars

In this section we consider four ways of associating frequency information with lexicalized grammars. Using the LTAG framework outlined in section 1 as a basis we define four Stochastic Lexicalized Grammar formalisms which we will refer to as SLG(1), SLG(2), SLG(3) and SLG(4). The differences between them lie in how fine-grained the frequency information is, which in turn determines the extent to which the resulting probabilities can be dependent on derivational context.

2.1 Context-Free Frequencies

The first approach we consider is the simplest and will be referred to as **SLG(1)**. A single probability is associated with each elementary tree. This is the probability that that tree is used in a derivation in preference to another tree with the same nonterminal at its root. A grammar is therefore well-formed if, for each nonterminal symbol that can be at the root of a substitutable (adjoinable) tree, the sum of probabilities associated with all substitutable (adjoinable) trees with the same root nonterminal is 1. When nondeterminism arises in a derivation nothing about the derivational context can influence the way that a tree is expanded, since the probability that the various possible trees are adjoined or substituted at each node depends only on the identity of the nonterminal at that node. As a result we say the frequency information in an SLG(1) is *context-free*.

2.2 Node-Dependent Frequencies

The second approach considered here, which we will call **SLG(2)**, has been described before by both Schabes (1992) and Resnik (1992). We describe the scheme of Schabes here, though the approach taken by Resnik is equivalent. In defining his scheme Schabes uses a stochastic version of a context-free-like grammar formalism called Linear Indexed Grammar (LIG). Based on the construction used to show the weak equivalence of TAG and LIG (Vijay-Shanker and Weir, 1994), a LIG is constructed from a given LTAG such that derivation trees of the LIG encode the derived trees of the associated LTAG. Compiling LTAG to LIG involves decomposing the elementary trees into single-level trees and introducing additional productions explicitly encoding every possible adjunction and substitution possibility[3]. It is the LIG productions encoding adjunction and substitution possibilities that are assigned probabilities[4]. The probabilities associated with all the productions that encode possible adjunctions (substitutions) at a node must sum to 1. The key feature of these probability-bearing LIG productions, in the context of the current discussion, is that they encode the adjunction or substitution of a specific elementary tree at a specific place in another elementary tree. This means that the frequency

[3] This scheme has proved useful in the study of LTAG parsing (Schabes, 1990; Vijay-Shanker and Weir, 1993; Boullier, 1996) since this pre-compilation process alleviates the need to do what amounts to the same decomposition process during parsing.

[4] The other productions (that decompose the tree structure) are assigned a probability of 1 since they are deterministic.

information can to some extent be dependent on context. In particular, when faced with nondeterminism in the way that some elementary tree is expanded during a derivation, the probability distribution associated with the alternative adjunctions or substitutions at a given node can depend on which elementary tree that node comes from. As a result we call the frequency information in SLG(2) **node-dependent**. This makes SLG(2) more expressive than SLG(1). As both Schabes and Resnik point out, by leveraging LTAG's extended domain of locality this approach allows probabilities to model both lexical and structural co-occurrence preferences.

The head automata of Alshawi (1996) also fit into the SLG(2) formalism since they involve a dependency parameter which gives the probability that a head has a given word as a particular dependent.

2.3 Locally-Dependent Frequencies

The third approach is **SLG(3)** which falls out quite naturally from consideration of the TAG derivation process. As we discussed in the introduction, LTAG derivations can be encoded with derivation trees in which nodes are labeled by the names of elementary trees and edges labeled by the addresses of substitution and adjunction nodes. The tree addresses can be omitted from derivation trees if a fixed linear order is established on *all* of the adjunction and substitution nodes in each elementary tree and this ordering is used to order siblings in the derivation tree. Given this possibility, Vijay-Shanker, Weir, and Joshi (1987) have shown that the set of derivation trees associated with a TAG forms a local set and can therefore be generated by a context-free grammar (CFG)[5]. The productions of this **meta-grammar** encode possible derivation steps of the grammar. In other words, each meta-production encodes one way of (fully) expanding an elementary tree[6]. In SLG(3) a probability is associated with each of these meta-productions. A SLG(3) is well-formed if for each elementary tree the sum of the probabilities associated with the meta-productions for that tree is 1.

In contrast to SLG(2)—which is limited to giving the probability that a tree anchored with a given lexical item is substituted or adjoined into a tree anchored with a second lexical item—SLG(3) specifies the probability that a particular *set* of lexical items is combined in a derivation step. It is the elementary trees of the underlying LTAG that determine the (extended local) domains over which these dependencies can be expressed since it is the structure of an elementary tree that determines the possible daughters in a meta-production. Although the types of elementary tree structures licensed are specific to a particular LTAG, it might be expected that a SLG(3) meta-grammar, for example, could encode the probability that a given verb takes a particular (type of) subject and combination of complements, including cases where the complements had been moved from their canonical positions, for example by extraction. A meta-grammar would also be likely to be able to differentiate the probabilities of particular co-occurrences of adverbial and prepositional phrase modifiers, and would moreover be able to distinguish between different orderings of the modifiers.

The approach described by Lafferty, Sleator, and Temperley (1992) of associating probabilities with Link Grammars—taken to its logical conclusion—corresponds to SLG(3), since in that approach separate probabilities are associated with each way of linking a word up with a combination of other words[7].

2.4 Globally-Dependent Frequencies

The fourth and final approach we consider is Bod's Data-Oriented Parsing (DOP) framework (Bod, 1995). In this paper we call it **SLG(4)** for uniformity and ease of reference. Bod formalizes DOP in terms of a *stochastic tree-substitution grammar*, which consists of a finite set of elementary trees, each with an associated probability such that the probabilities of all the trees with the same non-terminal symbol sum to 1, with an operation of substitution to combine the trees. In DOP, or SLG(4), the elementary trees are arbitrarily large subtrees anchored at terminal nodes by words/part-of-speech labels, and acquired automatically from pre-parsed training data. This is in contrast to SLG(3), in which the size of individual meta-productions is bounded, since the structure of the meta-productions is wholly determined by the form of the elementary trees in the grammar.

[5] In such context-free grammars, the terminal and nonterminal alphabets are not necessarily disjoint, and only the trees generated by the grammar (not their frontier strings) are of any interest.

[6] In the formulation of TAG derivations given by Schabes and Shieber (1994) an arbitrary number of modifications can take place at a single node. This means that there are an infinite number of productions in the meta-grammar, i.e., an infinite number of ways of expanding trees. This means that a pure version of SLG(3) is not possible. See Section 4.2 for ways to deal with this issue.

[7] Lafferty, Sleator, and Temperley (1992) appear to consider only cases where a word has at most one right and one left link, i.e., probabilities are associated with at most triples. However, the formalism as defined by Sleator and Temperley (1993) allows a more general case with multiple links in each direction, as would be required to deal with, for example, modifiers.

3 Empirical Evaluation

We have described four ways in which frequency information can be associated with a lexicalized grammar. Directly comparing the performance of the alternative schemes by training a wide-coverage grammar on an appropriate annotated corpus and then parsing further, unseen data using each scheme in turn would be a large undertaking outside the scope of this paper. However, each scheme varies in terms of the degree to which it can, in principle, distinguish the probability of certain kinds of derivational phenomena. This can be tested without the need to develop and run a parsing system, since each scheme can be seen as making verifiable predictions about the absence of certain dependencies in derivations of sentences in corpus data.

SLG(1), with only context-free frequency information, predicts that the relative frequency of use of the trees for a given nonterminal is not sensitive to where the trees are used in a derivation. For example, there should be no significant difference between the likelihood that a given NP tree is chosen for substitution at the subject position and the likelihood that it is chosen for the object position. SLG(2) (using so-called node-dependent frequency information) is able to cater for such differences but predicts that the likelihood of substituting or adjoining a tree at a given node in another tree is not dependent on what else is adjoined or substituted into that tree. With SLG(3) (which uses what we call locally-dependent frequency information) it is possible to encode such sensitivity, but more complex contextual dependencies cannot be expressed: for example, it is not possible for the probability associated with the substitution or adjunction of a tree γ into another tree γ' to be sensitive to where the tree γ' itself is adjoined or substituted. Only SLG(4) (in which frequency information can be globally-dependent) can do this.

In the remainder of this section we present a number of empirical phenomena that support or refute predictions made by each of the versions of SLG.

3.1 SLG(1) vs. SLG(2–4)

Magerman and Marcus (1991) report that, empirically, a noun phrase is more likely to be realized as a pronoun in subject position than elsewhere. To capture this fact it is necessary to have two different sets of probabilities associated with the different possible NP trees: one for substitution in subject position, and another for substitution in other positions. This cannot be done in SLG(1) since frequency information in SLG(1) is context-free. This phenomenon therefore violates the predictions of SLG(1), but it can be captured by the other SLG models.

Individual lexemes also exhibit these types of distributional irregularities. For example, in the Wall Street Journal (WSJ) portion of the Penn Treebank 2 (Marcus et al., 1994), around 38% of subjects of verbs used intransitively (i.e., without an object NP) in active, ungapped constructions are either pronouns or proper name phrases[8]. However, for the verbs *believe*, *agree*, and *understand*, there is a significantly higher proportion (in statistical terms) of proper name/pronoun subjects (in the case of *believe* 57%; χ^2, 40.53, 1 *df*, $p < 0.001$) [9]. This bias would, in semantic terms, be accounted for by a preference for subject types that can be coerced to *human*. SLG(2–4) can capture this distinction whereas SLG(1) cannot since it is not sensitive to where a given tree is used.

3.2 SLG(2) vs. SLG(3–4)

The Penn Treebank can also allow us to probe the differences between the predictions made by SLG(2) and SLG(3–4). From an analysis of verb phrases in active, ungapped constructions with only pronominal and/or proper name subjects and NP direct objects, it is the case that there is a (statistically) highly significant dependency between the type of the subject and the type of the object (χ^2, 29.79, 1 *df*, $p < 0.001$), the bias being towards the subject and direct object being either (a) both pronouns, or (b) both proper names. Thus the choice of which type of NP tree to fill subject position in a verbal tree can be dependent on the choice of NP type for object position. Assuming that the subject and object are substituted/adjoined into trees anchored by the verbs, this phenomenon violates the predictions of SLG(2)—hence also SLG(1)—but can still be modeled by SLG(3–4).

[8] Subjects were identified as the NP-SBJ immediately preceding a VP bracketing introduced by a verb labeled VBD/VBP/VBZ; pronouns, words labeled PRP/PRP$; and proper noun phrases, sequences of words all labeled NNP/NNPS.

[9] A value for p of 5 corresponds to statistical significance at the standard 95% confidence level; smaller values of p indicate higher confidence.

A similar sort of asymmetry occurs when considering the distribution of pronoun and proper name phrases against other NP types in subject and direct object positions. There is again a significant bias towards the subject and object either both being a pronoun/proper name phrase, or neither being of this type (χ^2, 8.77, 1 *df*, $p = 0.3$). This again violates the predictions of SLG(2), but not SLG(3–4).

Moving on now to modifiers, specifically prepositional phrase (PP) modifiers in verb phrases, the Penn Treebank distinguishes several kinds including PPs expressing manner (PP-MNR), time (PP-TMP), and purpose (PP-PRP). Where these occur in combination there is a significant ordering effect: PP-MNR modifiers tend to precede PP-TMP (χ^2, 4.12, 1 *df*, $p = 4.2$), and PP-TMP modifiers in their turn have a very strong tendency to precede PP-PRP ($p < 0.001$). Adopting Schabes and Shieber's (1994) formulation of the adjunction operation in TAG, multiple PP modifier trees would be adjoined independently at the same node in the parent VP tree, their surface order being reflected by their ordering in the derivation tree. Therefore, in SLG(3) multiple modifying PPs would appear within a single meta-production in the order in which they occurred, and the particular ordering would be assigned an appropriate probability by virtue of this. In contrast, SLG(2) treats multiple adjunctions separately and so would not be able to model the ordering preference.

Significant effects involving multiple modification of particular lexical items are also evident in the treebank. For example, the verb *rise* occurs 83 times with a single PP-TMP modifier—e.g. (1a)—and 12 times with two (1b), accounting in total for 6% of all PPs annotated in this way as temporal.

(1) a *Payouts on the S&P 500 stocks rose 10 % [PP-TMP in 1988] , according to Standard & Poor 's Corp. ...*
b *It rose largely [PP-TMP throughout the session] [PP-TMP after posting an intraday low of 2141.7 in the first 40 minutes of trading] .*

The proportion of instances of two PP-TMP modifiers with *rise* is significantly more than would be expected given the total number of instances occurring in the treebank (χ^2, 25.99, 1 *df*, $p < 0.001$). The verb *jump* follows the same pattern ($p = 1.0$), but other synonyms and antonyms of *rise* (e.g. *fall*) do not. This idiosyncratic behavior of *rise* and *jump* cannot be captured by SLG(2), since each adjunction is effectively considered to be a separate independent event. In SLG(3), though, the two-adjunction case would appear in a single meta-production associated with *rise/jump* and be accorded a higher probability than similar meta-productions associated with other lexical items.

There is another, more direct but somewhat less extensive, source of evidence that we can use to investigate the differences between SLG(2) and (3–4). B. Srinivas at the University of Pennsylvania has recently created a substantial parsed corpus[10] by analyzing text from the Penn Treebank using the XTAG system (Group, 1995). Some of the text has been manually disambiguated, although we focus here on the most substantial set—of some 9900 sentences from the WSJ portion—which has not been disambiguated, as yet. For each sentence we extracted the set of meta-level productions that would generate the XTAG derivation. To obtain reliable data from ambiguous sentences, we retained only the (approximately 37500) productions that were common across all derivations. In this set of productions we have found that with the elementary tree licensing subject–transitive-verb–object constructions, the likelihood that the object NP is expanded with a tree anchored in *shares* is much higher if the subject is expanded with with a tree anchored in *volume*, corresponding to sentences such as (2a) and (2b).

(2) a *Volume totaled 14,890,000 shares .*
b *Overall Nasdaq volume was 151,197,400 shares .*

Indeed, in all 11 cases where *volume* is the anchor of the subject, an NP anchored in *shares* is analyzed as the object, whereas more generally *shares* is object in only 18 of the 1079 applications of the tree. This difference in proportions is statistically highly significant ($p < 0.001$). Correlation between each of *volume* and *shares* and the verbs that appear is much weaker. There is of course potential for bias in the frequencies since this data is based purely on unambiguous productions. We therefore computed the same proportions from productions derived from all sentences in the XTAG WSJ data; this also resulted in a highly significant difference. SLG(2) models the substitution of the subject and of the object as two independent events, whereas the data show that they can exhibit a strong inter-dependency.

[10] We wish to thank B. Srinivas for giving us access to this resource.

3.3 SLG(3) vs. SLG(4)

Bod (1995) observes that there can be significant inter-dependencies between two or more linguistic units, for example words or phrases, that cut across the standard structural organization of a grammar. For example, in the Air Travel Information System (ATIS) corpus (Hemphill, Godfrey, and Doddington, 1990) the generic noun phrase (NP) *flights from X to Y* (as in sentences like *Show me flights from Dallas to Atlanta*) occurs very frequently. In this domain the dependencies between the words in the NP—but without X and Y filled in—are so strong that in ambiguity resolution it should arguably form a single statistical unit. Bod argues that Resnik and Schabes' schemes (i.e. SLG(2)) cannot model this; however it appears that SLG(3) can since the NP would give rise to a single meta-production (under the reasonable assumption that the *from* and *to* PPs would be adjoined into the NP tree anchored by *flights*).

An example given by Bod that does demonstrate the difference between SLG(3) and SLG(4) concerns sentences like *the emaciated man starved*. Bod argues that there is a strong (semantic) dependence between *emaciated* and *starved*, which would be captured in DOP—or SLG(4)—in the form of a single elementary tree in which *emaciated* and *starved* were the only lexical items. This dependence cannot be captured by SLG(3) since *emaciated* and *starved* would anchor separate elementary trees, and the associations made would merely be between (1) the S tree anchored by *starved* and the substitution of the NP anchored by *man* in subject position, and (2) the modification of *man* by *emaciated*.

3.4 Discussion

The empirical phenomena discussed above mainly concern interdependencies within specific constructions between the types or heads of either complements or modifiers. The phenomena fall clearly into two groups:

- ones relating to distributional biases that are independent of particular lexical items, and
- others that are associated with specific open class vocabulary.

Token frequencies—with respect to treebank data—of phenomena in the former group are relatively high, partly because they are not keyed off the presence of a particular lexical item: for example in the case study into the complement distributions of pronoun/proper name phrases versus other NP types (section 3.2) there are 13800 data items (averaging one for every four treebank sentences). However, there appears to be a tendency for the phenomena in this group to exhibit smaller statistical biases than are evident in the latter, lexically-dependent group (although all biases reported here are significant at least to the 95% confidence level). In the latter group, although token frequencies for each lexical item are not large (for example, the forms of *rise* under consideration make up only 1% of comparable verbs in the treebank), the biases are in general very strong, in what are otherwise an unremarkable set of verbs and nouns (*believe*, *agree*, *understand*, *rise*, *jump*, *volume*, and *shares*). We might therefore infer that although individually token frequencies are not great, *type* frequencies are (i.e. there are a large number of lexical items that display idiosyncratic behavior of some form or other), and so lexicalized interdependencies are as widespread as non-lexical ones.

4 Parameter Estimation

4.1 Training Regime

Schabes (1992) describes an iterative re-estimation procedure (based on the Inside-Ouside Algorithm (Baker, 1979)) for refining the parameters of an SLG(2) grammar given a corpus of in-coverage sentences; the algorithm is also able to simultaneously acquire the grammar itself. The aim of the algorithm is to distribute the probability mass within the grammar in such as way that the *probability of the training corpus* is maximized, i.e. model as closely as possible the language in that corpus. However, when the goal is to return as accurately as possible the *correct analysis* for each sentence using a pre-existing grammar, estimating grammar probabilities directly from normalized frequency counts derived from a pre-parsed training corpus can result in accuracy that is comparable or better to that obtained using re-estimation (Charniak, 1996). Direct estimation would mesh well with the SLG formalisms described in this paper.

4.2 Smoothing

The huge number of parameters required for a wide-coverage SLG(2) (and even more so for SLG(3–4)) means that not only would the amount of frequency information be unmanageable, but data sparseness would make useful probabilities hard to obtain. We briefly present three (essentially orthogonal and independent) backing-off techniques that could be used to address this problem.

Unanchored Trees

It is the size of a wide-coverage lexicon that makes pure SLG(2–4) unmanageable. However, without lexical anchors a wide-coverage SLG would have only a few hundred trees (Group, 1995). Backup frequency values could therefore be associated with unanchored trees and used when data for the anchored case was absent.

Lexical Rules

In a lexicalized grammar, elementary trees may be grouped into families which are related by lexical rules—such as *wh* extraction, and passivization. (For example, the XTAG grammar contains of the order of 500 rules grouped into around 20 families). In the absence of specific frequency values, approximate (backup) values could be obtained from a tree that was related by some lexical rule.

SLG(i) to SLG($i-1$)

Section 3 indicated informally how, when moving from SLG(1) through to SLG(4), the statistical model becomes successively more fine-grained, with each SLG(i) model subsuming the previous ones, in the sense that SLG(i) is able to differentiate probabilistically all structures that previous ones can. Thus, when there is insufficient training data, sub-parts of a finer-grained SLG model could be backed off to a model that is less detailed. For example, within a SLG(3) model, in cases where a particular set of meta-productions all with the same mother had a low collective probability, the set could be reduced to a single meta-production with unspecified daughters (i.e. giving the effect of SLG(1)).

5 Comparison with Other Work

The treatment of stochastic lexicalized grammar in this paper has much in common with recent approaches to statistical language modeling outside the TAG tradition. Firstly, SLG integrates statistical preferences acquired from training data with an underlying wide-coverage grammar, following an established line of research, for example (Chitrao and Grishman, 1990; Charniak and Carroll, 1994; Briscoe and Carroll, 1995). The paper discusses techniques for making preferences sensitive to context to avoid known shortcomings of the context-independent probabilities of SCFG (see e.g. Briscoe and Carroll (1993)).

Secondly, SLG is *lexical*, since elementary trees specify lexical anchors. Considering the anchor of each elementary tree as the head of the construction analyzed, successive daughters for example of a single SLG(3) meta-grammar production can in many cases correspond to a combination of Magerman's (1995) mother/daughter and daughter/daughter head statistics (although it would appear that Collins' (1996) head-modifier configuration statistics are equivalent only to SLG(2) in power). However, due to its extended domain of locality, SLG(3) is not limited to modeling local dependencies such as these, and it can express dependencies between heads separated by other, intervening material. For example, it can deal directly and naturally with dependencies between subject and any verbal complement without requiring mediation via the verb itself: c.f. the example of section 3.2.

Thirdly, the SLG family has the ability to model explicitly syntactic *structural* phenomena, in the sense that the atomic structures to which statistical measures are attached can span multiple levels of derived parse tree structure, thus relating constituents that are widely-separated—structurally as well as sequentially—in a sentence. Bod's DOP model (Bod, 1995) shares this characteristic, and indeed (as discussed in section 2.4) it fits naturally into this family, as what we have called SLG(4).

Srinivas et al. (1996) (see also Joshi and Srinivas (1994)) have recently described a novel approach to parsing with LTAG, in which each word in a sentence is first assigned the most probable elementary tree—or 'supertag'—given the context in which the word appears, according to an trigram model of supertags. The rest of the parsing

process then reduces to finding a way of combining the supertags to form a complete analysis. In this approach statistical information is associated simply with linear sub-sequences of elementary trees, rather than with trees within derivational contexts as in SLG(2–4). Although Srinivas' approach is in principle efficient, mistaggings mean that it is not guaranteed to return an analysis for every in-coverage sentence, in contrast to SLG. Also, its relatively impoverished probabilistic model would not be able to capture many of the phenomena reported in section 3.

Acknowledgement

This work was supported by UK EPSRC project GR/K97400 'Analysis of Naturally-occurring English Text with Stochastic Lexicalized Grammars' (<http://www.cogs.susx.ac.uk/lab/nlp/dtg/details.html>), and by an EPSRC Advanced Fellowship to the first author. We would like to thank Nicolas Nicolov and Miles Osborne for useful comments on previous drafts.

References

Alshawi, Hiyan. 1996. Head automata and bilingual tilings: Translation with minimal representations. In *34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 167–176.

Baker, J. 1979. Trainable grammars for speech recognition. In D. Klatt and J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*. MIT, Cambridge, MA, pages 547–550.

Bod, R. 1995. *Enriching Linguistics with Statistics: Performance Models of Natural Language*. Ph.D. thesis, University of Amsterdam, ILLC dissertation series 95-14.

Booth, T. and R. Thompson. 1973. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450.

Boullier, P. 1996. Another facet of LIG parsing. In *34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 87–94.

Briscoe, E. and J. Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–60.

Briscoe, E. and J. Carroll. 1995. Developing and evaluating a probabilistic LR parser of part-of-speech and punctuation labels. In *4th International Workshop on Parsing Technologies (IWPT'4)*, pages 48–58.

Charniak, E. 1996. Tree-bank grammars. Technical Report CS-96-02, Brown University, Department of Computer Science.

Charniak, E. and G. Carroll. 1994. Context-sensitive statistics for improved grammatical language models. In *12th National Conference on Artificial Intelligence (AAAI'94)*, pages 728–733.

Chitrao, M. and R. Grishman. 1990. Statistical parsing of messages. In *DARPA Speech and Natural Language Workshop*, pages 263–266.

Collins, M. 1996. A new statistical parser based on bigram lexical dependencies. In *34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 184–191.

Group, The XTAG Research. 1995. A lexicalized tree adjoining grammar for English. Technical Report IRCS 95-03, The Institute for Research in Cognitive Science, University of Pennsylvania.

Hemphill, C., J. Godfrey, and G. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*.

Joshi, A. and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): almost parsing. In *15th International Conference on Computational Linguistics (COLING'94)*.

Lafferty, J., D. Sleator, and D. Temperley. 1992. Grammatical trigrams: a probabilistic model of link grammar. In *AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.

Magerman, D. 1995. Statistical decision-tree models for parsing. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 276–283.

Magerman, D. and M. Marcus. 1991. Pearl: a probabilistic chart parser. In *2nd International Workshop on Parsing Technologies (IWPT'2)*, pages 193–199.

Marcus, M., G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: annotating predicate argument structure. In *Proceedings of the ARPA Human Language Technology Workshop*.

Rambow, O., K. Vijay-Shanker, and D. Weir. 1995. D-Tree Grammars. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 151–158.

Resnik, P. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *14th International Conference on Computational Linguistics (COLING'92)*, pages 418–424.

Schabes, Y. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Schabes, Y. 1992. Stochastic lexicalized tree-adjoining grammars. In *14th International Conference on Computational Linguistics (COLING'92)*, pages 426–432.

Schabes, Y. and S. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Sleator, D. and D. Temperley. 1993. Parsing English with a link grammar. In *3rd International Workshop on Parsing Technologies (IWPT'3)*, pages 277–292.

Srinivas, B., C. Doran, B. Hockey, and A. Joshi. 1996. An approach to robust partial parsing and evaluation metrics. In J. Carroll, editor, *Proceedings of the Workshop on Robust Parsing*. 8th European Summer School in Logic, Language and Information, pages 70–82.

Vijay-Shanker, K. 1987. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

Vijay-Shanker, K. and D. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

Vijay-Shanker, K. and D. Weir. 1994. The equivalence of four extensions of context-free grammars. *Math. Syst. Theory*, 27:511–546.

Vijay-Shanker, K., D. Weir, and A. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Meeting of the Association for Computational Linguistics (ACL'87)*, pages 104–111.

Towards a Reduced Commitment, D-Theory Style TAG Parser

John Chen*
K. Vijay-Shanker
Department of Computer and Information Sciences, University of Delaware
Newark, DE 19716, USA
Email: {jchen,vijay}@cis.udel.edu

Abstract

Many traditional TAG parsers handle ambiguity by considering all of the possible choices as they unfold during parsing. In contrast, D-theory parsers cope with ambiguity by using underspecified descriptions of trees. This paper introduces a novel approach to parsing TAG, namely one that explores how D-theoretic notions may be applied to TAG parsing.

Combining the D-theoretic approach to TAG parsing as we do here raises new issues and problems. D-theoretic underspecification is used as a novel approach in the context of TAG parsing for delaying attachment decisions. Conversely, the use of TAG reveals the need for additional types of underspecification that have not been considered so far in the D-theoretic framework. These include combining sets of trees into their underspecified equivalents as well as underspecifying combinations of trees.

In this paper, we examine various issues that arise in this new approach to TAG parsing and present solutions to some of the problems. We also describe other issues which need to be resolved for this method of parsing to be implemented.

1 Introduction

Ambiguity is a central problem in the parsing of natural languages. Within the framework of TAG, a number of approaches have been adopted in trying to deal with this issue. Traditional approaches share the characteristic of exploring all of the choices in the face of ambiguity. To make such a scheme effective, techniques such as structure sharing and dynamic programming are commonly used. In contrast, we explore a different style of parsing that copes with ambiguity through the underspecification of tree structures.

This style of parsing deals with descriptions of trees instead of trees themselves. Notable examples of such parsers include [Marcus, Hindle, and Fleck, 1983], [Gorrell, 1995], and [Sturt and Crocker, 1996]. We refer to these as D-theory parsers. Following them, we construct an underspecified representation that captures a set of parses for the input seen so far.

The primitive of dominance is used in the descriptions that our parser manipulates. Its use in parsing has been pioneered by [Marcus, Hindle, and Fleck, 1983], and further investigated by the others such as those listed above. It has also been used within the confines of TAG elementary trees, as stated by [Vijay-Shanker, 1992]. Nevertheless, in our context of using domination to handle ambiguity in TAG parsing, we differ from both of these bodies of work. We differ from the latter because our use of domination is not limited to use within TAG elementary trees; it is crucially used here to underspecify ambiguity in the attachment of elementary trees. Thus the domination statements that our parser makes are domination statements between nodes in different elementary trees. We differ from prior work in D-theory parsing because our parser incorporates a specific grammatical formalism and grammar in order to specify syntactically well-formed structures. Moreover, it uses new kinds of representations that have not been considered in D-theory parsing in order to specify ambiguity.

This paper discusses several novel issues and presents our solutions to many of the issues that arise in adopting the use of dominance to parsing TAGs. For example, there is considerable bookkeeping that is necessary to ensure whether a particular underspecified representation conforms to a possible legal TAG derivation. Furthermore, TAG's extended domain of locality presents a problem because it hinders the degree of underspecification

*Supported by NSF grants #GER-9354869 and #SBR-9710411.

that is necessary in order to delay disambiguation decisions. We therefore argue that sets of elementary trees need to be underspecified into single representations. Another problem is that the notion of standard referent, used in D-theory parsing as the default model of a given underspecified representation, needs to be reconceptualized in order to accommodate the novel use of domination that is proposed here.

This paper is organized as follows. We first consider some broad distinguishing characteristics of our parser in Section 2. Subsequently, we define some terminology in Section 3 as a preparatory for the following sections, which delve into various aspects of of our parser. Section 4 gives a high level outline of our parser. Section 5 explicates the mechanism whereby descriptions of trees are combined. Section 6 deals with computing the standard referent. Section 7 discusses the issue of combining a set elementary trees into a single representation. Finally, concluding remarks are given in Section 8.

2 Parsing Framework

Our parser draws on typical characteristics of D-theoretic parsers, such as [Marcus, Hindle, and Fleck, 1983] and [Gorrell, 1995]. Such characteristics, as well as their relation to our parser, are discussed in this section. A basic feature of D-theoretic parsers is that the input sentence is parsed strictly from left to right and a description is constructed that captures a set of parse trees for the input seen.

One of our goals, which is shared by many other D-theoretic parsers, is that of incremental interpretation. This means at each point in the parse, a partial semantic representation is constructed from the sentence prefix that has been input so far, as delineated by the underspecified representation.

The notion of standard referent, as discussed in [Marcus, Hindle, and Fleck, 1983] and [Rogers and Vijay-Shanker, 1996], is included in our parser. Recall that an underspecified representation specifies a set of parse trees, namely those trees which are consistent with every assertion that is embodied by the underspecified representation. The standard referent is that parse tree which is taken to be the default choice from the set of trees that are consistent with a particular underspecified representation. The need for a standard referent would arise from a requirement of incremental interpretation. However, here it is mainly used in determining how structures may be combined.

Our parser subscribes to the notion of informational monotonicity. This means that, during parsing, the assertions that the parser makes are considered to be indelible. This is important because, at any point during parsing, we find that not all of the attachment possibilities that may be expressed by a dominance based description language can be incorporated into the underspecified representation that our parser constructs. Certain ambiguities, for example, may not be expressible in our underspecified representation. Therefore, there is the possibility that one of the assertions that the parser makes will turn out to be incompatible with evidence provided by later input. In that situation, the parser is forced to retract that assertion, leading to backtracking. In the literature on D-theory parsers, it has been commonly conjectured that such backtracking correlates with garden path effects.

3 Terminology

Our terminology borrows from TAG and D-Tree Grammar literature. From TAG, the grammar consists of a set of lexicalized trees. The frontier of each elementary tree is composed of a *lexical anchor*; the other nodes on the frontier will either be *substitution* or *foot* nodes. An interior node is commonly considered to be constituted of a top and bottom feature structure. In contrast, we use the terminology introduced by [Vijay-Shanker, 1992] in that an interior node is considered as a pair of *quasi-nodes*. A domination link connects the *top* and *bottom* quasi-nodes. The path from the root to the lexical anchor of an elementary tree will be called the *trunk*.

We now define the terms component and lowering node. The notion of component is borrowed from D-Tree Grammars, as defined in [Rambow, Vijay-Shanker, and Weir, 1995]. A *component* is a maximal subpart of an underspecified representation that has the property that all of its nodes are connected by immediate domination links. Consider α_0 in Figure 1 for example. It consists of two components: the first contains the nodes S, NP_0, and the top VP quasi-node, while the second contains the bottom VP quasi-node (VP_0), the V node, and the NP_1 node. A *lowering node* is the root of a component. It can either be the root of an elementary tree or a bottom quasi-node. We call such nodes lowering nodes because in D-theory, adjunction or substitution at these

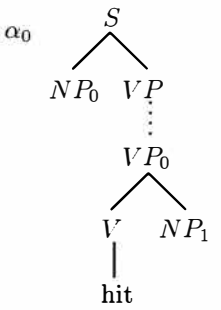


Figure 1: A lexicalized TAG tree with two components and two lowering nodes.

nodes would mean that these nodes would be lowered, i.e., later input causes some structure to be placed above them. The elementary tree α_0 contains two lowering nodes, S and VP_0.

4 Parsing Algorithm

With the terminology so defined, the disquisition of our parsing algorithm can now proceed. At any given time during parsing, we assume that there is a single *underspecified representation* R corresponding to the sentence prefix which has been seen so far. It describes a set of trees that are obtained by combinations of structures anchored by the input encountered. For now, we assume that R is treelike except that some of the nodes may be connected by domination links instead of immediate domination links. The immediate domination links arise only from the immediate domination links that appear in the composed TAG trees. The domination links arise from either those that appear in elementary TAG trees or between nodes in different trees. At the start of parsing, R consists of a single substitution node S.

A provisional description of the parsing process is outlined as follows.

- The next word is input. There will usually be more than one elementary tree corresponding to this word.
- Those elementary trees that are *structurally* incompatible with the current representation R are removed from consideration. This is addressed in Sections 5.1 and 5.2.
- The remaining elementary trees must be incorporated into the current representation R to produce the new underspecified representation. If only one elementary tree corresponding to the current input word remained after the second step's filter, then this would be straightforward. On the other hand, there may be cases where more than one tree remains. We propose to address the latter situation by grouping such sets of elementary trees into one underspecified representation. This would allow us to delay the choice of elementary tree until later while still being consistent with informational monotonicity. Nevertheless, it may still be the case that the remaining set of elementary trees corresponding to the current word cannot be grouped together. In this case, we would have to make a choice. We propose that other sources of information would need to be recruited in order to further constrain the set of viable elementary representations, as well as their attachment sites. These sources may include semantics, or lexical frequencies as determined from a corpus.

In what follows, we begin a more detailed discussion of certain essential aspects of our parsing algorithm. We start by considering what structural possibilities the parser needs to consider when it tries to combine the current underspecified representation R with an elementary tree η that is anchored by the next input word.

5 Control Strategy

This section describes how representations may be combined. Associated with each representation are expectation lists that encode how it expects to fit into a complete sentence structure. We first describe the nature of expectation lists. Next, we develop the control strategy that is used to check if two representations may be combined, through the use of expectation lists. We also consider what actions may take place if two expectation lists are incompatible.

5.1 Expectations

At any given point in the parsing of the input sentence, subsequent input may clearly not be inserted into the part of R that is to the left of that path running from the root of R to the leaf l. Extending the notion of trunk, which has hitherto only been defined for elementary trees, we state that the *trunk* of an elementary representation R is the path from the root to the last input node. Thus, when inserting new structure into R, only the nodes and domination links on and to the right of the trunk need to be considered. These nodes are contained in what we call the *right expectation list* of R.

Analogous observations hold about the elementary TAG trees. Namely, at each point in parsing, an elementary tree η, representing the next input word, is combined with R. When considering how η may be combined with R, it suffices to consider only those nodes and domination links that appear on or to the left of that path from the root to the anchor. These nodes are contained in the *left expectation list* of η.

Now we will consider how to compute the left expectation list of R and the right expectation list of η. In the relevant subparts of R and η, we need to examine nodes that can be involved in combinations: lowering nodes, substitution nodes, and foot nodes. In our expectation lists, we consider expectations of two types: substitution expectations or lowering expectations. A substitution expectation corresponds to a substitution node or a foot node. Conversely, a lowering expectation corresponds to a lowering node.

Each expectation is characterized as optional or obligatory. Substitution expectations are always obligatory. Lowering expectations may either be optional or obligatory. Suppose there exists a domination link that states that node a dominates node b. The lowering expectation corresponding to node b is obligatory if the labels of a and b do not match.

Expectation lists are *ordered* lists of expectations. The order of expectations for one representation corresponds to the sequence in which the parser will try to assemble it with the other representation. The algorithm that is presented in Figure 2 returns an ordered list of expectations when given an underspecified representation or elementary tree and a direction d that specifies whether a left or right expectation list should be found. It first examines the anchor and then makes its way up the trunk of the representation, one node at a time. At each such node n, the algorithm sees if n corresponds to an expectation. If so, this is recorded. It also checks if n has any siblings in the dth direction. If so, each such sibling is scanned in a depth first fashion for any expectations.

Consider the expectations in the right expectation list for α_0 in Figure 1. The algorithm starts at the anchor "hit" and then makes its way up to the V node. The V node has a sibling, NP_1, which is examined in a depth first fashion for any expectations. This yields the first expectation, $NP_{1(oblig,subst)}$. The algorithm continues up the trunk to the bottom VP quasi-node (VP_0), which is associated with another expectation $VP_{(opt,low)}$, which is duly recorded. Continuing further up the trunk, the algorithm finds one more expectation corresponding to the root S, which is $S_{(opt,low)}$. Similarly, the left expectation list of α_0 would be $< VP_{(opt,low)}, NP_{0(oblig,subst)}, S_{(opt,low)} >$.

5.2 Combining Representations

As the parser proceeds through the input sentence from left to right, the parser tries to combine the underspecified representation R with an elementary tree η corresponding to the current input word. Expectation lists encode how a representation to the left may be combined with an elementary tree to the right. We say that the underspecified representation R can be combined with the elementary tree η when the right expectation list of R is *compatible* with the left expectation list of η.

An outline of a proposed procedure to determine the compatibility of two expectation lists is delineated in this section. This procedure has two steps. First, one expectation is taken from the left expectation list of η and the right expectation list of R is searched for a matching expectation. Second, given that the two representations partially combine according to how these first two expectations match, it is seen whether the two representations can be fully combined.

[1] If d is left (right), then *not* d is right (left).

[2] Node p is a *parent* of a node c if p is connected to c by a domination or immediate domination link. For example, a top quasi-node is the parent of the corresponding bottom quasi-node.

[3] The *child* relation is defined analogously to the parent relation in the previous footnote. For example, a bottom quasi-node is the child of its corresponding top quasi-node.

Order_expect(n:node, d:left or right)
  Case n lies on trunk
    If n is a lowering node
      Record n as a lowering expectation
    Loop through each *sibling* i of n that appears to the d of n from *not* d[1] to d
      Order_expect(i, d)
    If n has a parent p[2]
      Order_expect(p, d)
  Case n does *not* lie on trunk
    If n is a substitution node
      Record n as a substitution expectation
    Else (n is an interior node)
      If n is a lowering node
        Record n as a lowering expectation
      Loop through each *child*[3] i of n that appears to the d of n from *not* d to d
        Order_expect(i, d)
      If n is an adjoining node
        Record n as a lowering expectation

Figure 2: Algorithm that returns an ordered list of left or right expectations.

Finding the First Pair of Matching Expectations

We take the first expectation e_1 from the left expectation list of η. e_1 may either be a substitution expectation or a lowering expectation. If it is the former, we look for a matching lowering expectation in R. Conversely, if it is the latter, we look for a matching substitution expectation.

It may occur that, although e_1 is an obligatory expectation, there is no expectation in R that matches. We need not consider this particular situation further, because the two representations are obviously incompatible. Alternatively, although there may be no matching expectation for e_1, e_1 is an optional expectation. In this case, we can skip to the next expectation on the left expectation list of the elementary tree.

Matching the Rest of the Two Expectation Lists

Suppose that a matching expectation e_m (in R) for e_1 (in η) is found. Now there are two tasks that must be accomplished. First it must be ensured that this matching expectation can indeed lead to a valid combination of the two representations that are under consideration. This is accomplished as follows. Suppose that n_m and n_1 are the nodes corresponding to the expectations e_m and e_1, respectively. Now let p_m be the path from the root of the underspecified representation R to n_m. Conversely, let p_1 be the path from the root of the elementary tree η to n_1. The task is to find out if the components along the path p_m can be interspersed with the components along the path p_1. This is done by iteratively matching the expectations corresponding to the nodes that lie on path p_m with those that lie on the path p_1, starting from those expectations corresponding to nodes that lie closest to e_m and e_1. Second, any remaining unmatched obligatory expectations in η should be matched, in order of their appearance on the expectation list, by recursively calling the aforementioned procedure.

Representing Ambiguity of Attachment

Suppose e_1 is a substitution expectation and we find a matching lowering expectation e_m in R. That e_m must be the corresponding attachment site for e_1 does not immediately hold. In Figure 3(b), for example, the auxiliary tree could be adjoined at either of the VP's in R. To capture such attachment ambiguity, rather than equating n_1 and n_m (the node corresponding to the first expectation in the expectation list for R that matches e_1), we will say n_1 dominates n_m. On the other hand, in Figure 3(a), there is no ambiguity in attachment of the new elementary tree. In this case we can make a stronger statement by equating n_1 and n_m.

In general, the type of attachment ambiguity that we underspecify is where e_1 is a substitution expectation that matches more than one lowering expectation and no obligatory expectations occur between the first and

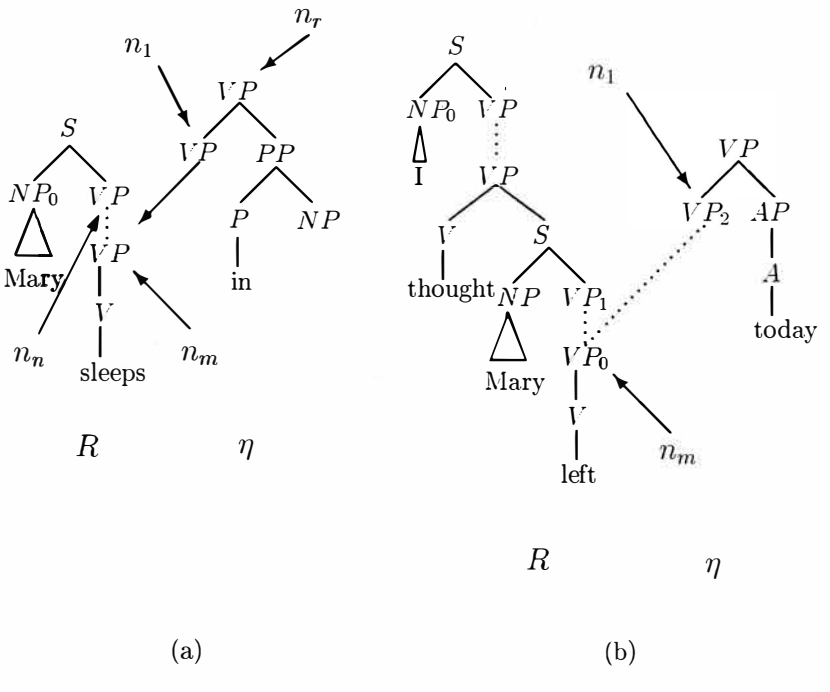


Figure 3: Example of case 1 of the algorithm to match expectation lists. (a) is the subcase of unambiguous attachment. (b) is the subcase of ambiguous attachment.

the last matching lowering expectations on the left expectation list of η, inclusively.

Whether such an ambiguity exists is determined as follows. Suppose e_1 is a substitution expectation. Furthermore suppose its combination with e_m, the first matching lowering expectation, does in fact lead to a valid combination of representations. We may find another valid combination by scanning further in the right expectation list of the underspecified representation R, starting from e_m, to find another matching expectation $e_{m'}$ (before encountering an obligatory expectation that does not match e_1) and verifying that it leads to another valid combination of structures, using the same method that we performed for the first matching expectation e_m.

Our underspecified representation captures this ambiguity as follows. Notice that no matter where the elementary tree η is adjoined, it is the case that the node n_1 will always dominate the lowest attachment site n_m. Therefore, in cases of such ambiguity, we stipulate such a domination as shown in Figure 3(b). Notice that while the example in Figure 3(b) corresponds to adjoining, a similar situation arises even if n_1 is a substitution node.

Representing the ambiguity in this way has various repercussions. For example, if the elementary tree that is being ambiguously added is an auxiliary tree, then we should make sure that if disambiguation should subsequently occur, an additional assertion should be made in order to guarantee a proper TAG adjunction. In detail, suppose that γ is the auxiliary tree, and n_1 is the foot node of γ. When disambiguation occurs, n_1 is equated with some bottom quasi-node n_b. In that case, an additional stipulation should be made that the top quasi-node n_t that corresponds to n_b dominates the root of γ. Another consequence is that an underspecified representation R that encodes such ambiguities is no longer the treelike structure that was assumed earlier. This necessitates alterations in our algorithm for computing expectation lists, as will be discussed in Section 6.

Details Behind Matching the First Two Expectations

As noted previously, matching one expectation list against another entails finding a sequence of pairs of corresponding substitution and lowering expectations. Whether two expectations match depends upon various factors, such as whether the substitution expectation occurs in R or η, and whether the nodes corresponding to the expectations are substitution nodes, foot nodes, or quasi-nodes.

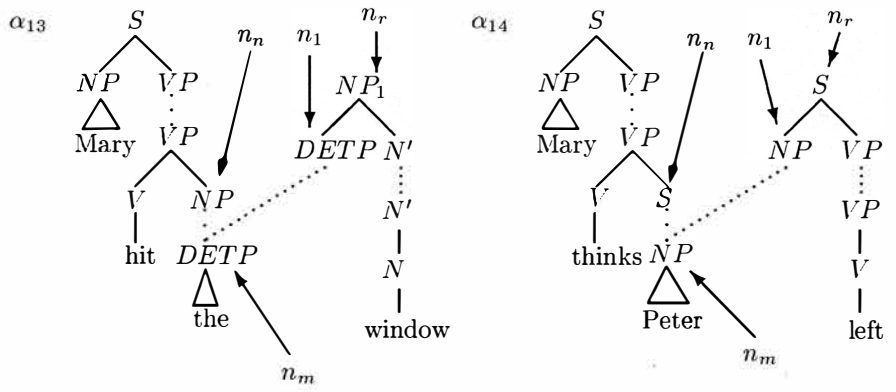


Figure 4: Case where stating that NP_0 dominates NP_1 is justified.

Let e_1 be the first expectation from the left expectation list of the elementary tree η. The expectation e_1 may either be a substitution expectation (corresponding to a substitution node or a foot node) or a lowering expectation (corresponding to a lowering node). We handle the tests for satisfaction of these two kinds of expectation as case 1 and case 2, respectively. In the following we take e_m to be the first expectation in the right expectation list of R that matches e_1. Furthermore, let n_m and n_1 be the nodes corresponding to the expectations e_m and e_1, respectively.

Case 1: e_1 is a Substitution Expectation. As stated previously, nodes n_1 and n_m are equated if there is no ambiguity in the placement of n_1. Otherwise, the ambiguity is encoded into the representation with node n_1 dominating n_m. Now consider the situations where n_1 is either a substitution node (Case 1a) or a foot node (Case 1b).

Case 1a: n_1 is a Substitution Node. If n_1 is a substitution node, then n_m must be the root of some elementary tree. This follows because the TAG operation of substitution requires that the root of an elementary tree be equated with a substitution node and that entire tree must have already been integrated into R. Node n_m may be dominated by another node n_n in R. n_n may be a substitution node (Case 1a-i) or a foot node (Case 1a-ii). We proceed to consider the cases where n_1 is to be equated with n_m.

Case 1a-i: n_n is a Substitution Node. Let n_r be the root of η. We can say that it is dominated by n_n if $n_r \in LC(n_n)$ [4]. See α_{13} of Figure 4 for an example. Otherwise, our original assertion of equality between n_1 and n_n was in error. The use of domination of n_r by n_n is to allow for further material to the right to be inserted between these two.

Case 1a-ii: n_n is a Foot Node. Suppose n_n is a foot node. If $n_r \notin LC(n_n)$ then n_n does not dominate n_r. Otherwise, n_n may or may not dominate n_r; the rest of the expectation lists would have to be checked in order to find out if there is the possibility of ambiguity in the placement of n_r. See α_{14} of Figure 4.

Case 1b: n_1 is a Foot Node. If n_1 is a foot node, then n_m must be a bottom quasi-node. Analogous to Case 1a, this follows because the TAG operation of adjoining requires that the foot node be equated with a bottom quasi-node. Again, where there is no ambiguity in the placement of n_1, we consider whether n_n is a foot node (Case 1b-i) or a top quasi-node (Case 1b-ii). Note that node n_n cannot be a substitution node because a substitution node cannot occur along the path between a top and bottom quasi-node.

Case 1b-i: n_n is a Foot Node. This case is similar to Case 1a-ii.

Case 1b-ii: n_n is a Top Quasi-node. If n_n is a top quasi-node, then it dominates n_r. This is to ensure that a valid TAG adjunction is performed. See Figure 3(a) for an example.

Case 2: e_1 is a Lowering Expectation. Again suppose that n_m and n_1 are the nodes corresponding to the expectations e_m and e_1, respectively. This time e_m must be a substitution expectation. Note that the possible attachment of n_m to more than one node in η does not occur. Because n_1 is a lowering node, it can only be a bottom quasi-node (Case 2a) or a root of an elementary tree (Case 2b).

Case 2a: n_1 is a Bottom Quasi-node. If n_1 is a bottom quasi-node, then n_m must be a foot node. It cannot be a substitution node because that would mean that a substitution node exists along the path between

[4] $LC(\nu)$ stands for left corner, by which we include anything that can appear in the left periphery of a tree whose root is ν.

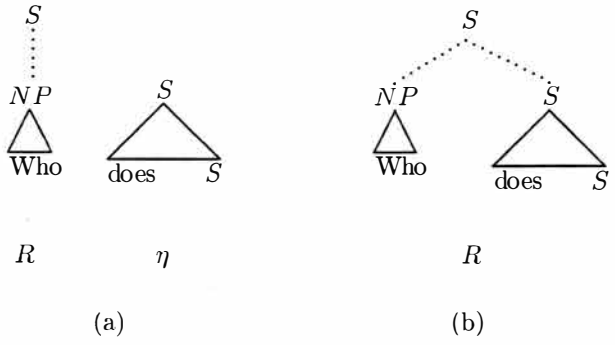


Figure 5: (a) Representations that are selected when the input "Who does..." is encountered. (b) The same representations combined with domination.

a top and bottom quasi-node. Node n_1 is dominated by another node n_k in elementary tree η, which can only be a top quasi-node. Because n_k is a top quasi-node, it dominates the root of the auxiliary tree that has n_m as a foot node. This follows from the fact that we want to preserve TAG adjoining.

Case 2b: n_1 is the Root of an Elementary Tree. If n_1 is the root of the elementary tree η, then n_m can either be a substitution node or a foot node. In this situation, node n_m is made to dominate n_1. Equality is not asserted because it may be possible for material from the right to be inserted between n_m and n_1.

When the Relationship between Two Representations is Locally Unknown

It is also possible that there is no expectation e_1 in the left expectation list of the elementary tree η which might suggest how to combine it with the underspecified representation R to its left. Furthermore, the expectations in that list may have all been optional. In this situation, although no assertion of domination can combine R and η, it is possible that subsequent input may be able to combine them. Call this Case 3. It occurs for example in the context of the sentence "Who does John annoy?" After the prefix "Who does..." has been seen, the parser considers combining the two representations that are shown in Figure 5(a), only to find that it is impossible.

As a general strategy for case 3, we will assume that the disparate representations are combined using dominance where both of the representations are dominated by the lowest node which we know for certain will dominate both. An example is shown in Figure 5(b). This strategy would entail complications to the procedure of building expectation lists and checking for the compatibility of expectations. Notice that this strategy is akin to buffering the uncombinable items. There is therefore the problem of determining what, if anything, is the proper buffer size.

6 Computing the Standard Referent

In D-theory, the notion of standard referent plays an important role. The standard referent is a minimal model (tree) of a description. It is usually obtained by minimizing the path lengths of all of the domination links. Such a tree would be useful in incremental interpretation. For us the method for obtaining the standard referent is also useful in determining the order in which the components of the current underspecified representation are to be combined with new additional input. Recall that the underspecified representation that our parser produces allows for non-treelike forms in order to describe some forms of attachment ambiguity. On the other hand, the algorithm that was given to compute expectation lists requires a treelike description as input.

Consider the description α_8 in Figure 6. The problem here is that obtaining the expectation list for α_8 is not possible because an expectation list requires a unique ordering of domination. In particular, whether XP_3 dominates XP_4 depends on whether or not XP_3 is placed above XP_4. If the standard referent is implicitly assumed to be the default structure, the question arises as to what the standard referent of α_8 is. Such cases have never arisen in previous work on D-theory. [5] In defining the notion of standard referent here, we have

[5] See [Rogers and Vijay-Shanker, 1996] for discussion on the determinism hypothesis in D-theory parsing and the uniqueness of the standard referent.

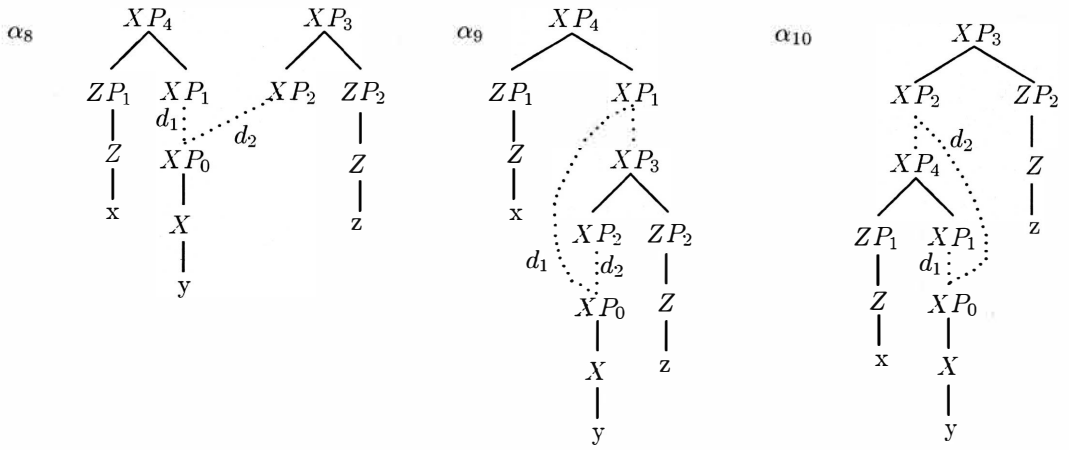


Figure 6: An example of our parser's underspecified representation (α_8), its standard referent (α_9), and another representation compatible with α_8 (α_{10}).

to consider disambiguating the attachment ambiguity that is expressed in structures such as α_8. Because the standard referent is usually taken to be the default structure, its definition could also be used to give an ordering for the expectation list.

The basic strategy that we would like to use in order to determine the standard referent is to minimize the lengths of the domination links. This strategy, however, does not always yield a single standard referent, as seen in α_8 of Figure 6. In this case, there is no unique standard referent because the sum of the lengths of the domination links d_1 and d_2 does not vary with the placement of the auxiliary tree that is rooted by node XP_3. [6] The difficulty arises when two domination links reach the same node. We call such pairs of domination links *ambiguous*. In α_8 the pair of domination links d_1 and d_2 are ambiguous because they both dominate the same node XP_3.

In order to guarantee that a single standard referent is obtained, the process of determining a standard referent is divided into two stages. First, all of the ambiguous domination links are disambiguated by minimizing their lengths sequentially according to a predetermined order, yielding an unambiguous representation R'. Second, a single standard referent is determined from R' by minimizing the sums of the lengths of the domination links.

Suppose d_1 and d_2 are any ambiguous pair of domination links in R. d_1 and d_2 state that certain nodes U and V dominate a node X. Clearly, U and V must come from different elementary trees, say β_U and β_V. There must have been an order in which β_U and β_V were introduced into R. This is the order in which the ambiguous domination links d_1 and d_2 are ordered.

Each domination link is assigned a value that specifies the time at which the elementary tree in which the link's parent node resides was introduced into R. In defining the order for an expectation list, the domination link with the higher value is ordered before one with a lower value. Consider for example the underspecified representation α_8 of Figure 6. There are two ambiguous domination links d_1 and d_2 with $value(d_1) < value(d_2)$. Consequently, d_2's length is minimized before d_1's length. The resulting underspecified representation is α_9.

There are two motivations behind our method. First, computing the expectation list of the result of computing the first step of the standard referent yields a sequence of expectations that does not omit any possible attachment points. For example, the expectation list of α_9 encodes the possibility of material being inserted between XP_1 and XP_3, something that would not happen had XP_3 been postulated to dominate XP_4, as in α_{10}. Second, this strategy typically corresponds to right association, as described in [Frazier, 1995]. Specifically, minimizing more recently created domination links first means that more recently added subtrees will be attached as low as possible. For example, the standard referent for the representation in Figure 3(b) has the adverb "today" modifying the lower verb "left." Insofar as right association has been postulated as a

[6] Another possible standard referent of α_8 would be found by equating the two pairs of nodes XP_1 and XP_2, and XP_3 and XP_4, which would result in a parse tree where, for example, ZP_1 and ZP_2 are siblings. Nevertheless, it does not result in a tree obtained by any sequence of TAG operations, assuming that the component that is rooted at XP_4 and the component that is rooted at XP_3 come from different elementary structures. We avoid this undesirable result by never attempting, in computing the standard referent, to equate two nodes simply because they both happen to be asserted as parents of a single node.

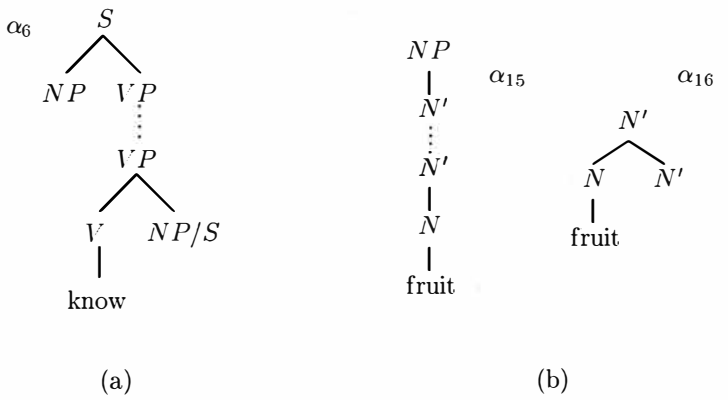


Figure 7: (a) Elementary representation corresponding to two elementary TAG trees. (b) Elementary trees that require underspecification in order for parser to avoid backtracking.

general strategy for human parsing, modulated perhaps by semantic and discourse factors, it corresponds with this strategy of computing the standard referent, which may be used as the baseline structure for semantic interpretation.

7 Elementary Representations

Underspecification plays an important role in reduced commitment parsing. Too great an underspecification means that few constraints are placed on what the final structure of the sentence will look like. It diminishes the predictive ability of the parser and enlarges the range of potential structures that are considered at any given time. Conversely, too little an underspecification leads the parser into making incorrect predictions and reduces the number of sentences that can be parsed without backtracking.

The enlarged domain of locality that TAG provides causes the latter sort of problem in this context. Let us select the strategy of choosing only one elementary tree at each step in the parse. Consider the problems it raises when trying to parse the following pair of sentences: "I know the answer" and "I know the answer is wrong." Crucially, after having seen the word "know" the parser cannot predict which elementary tree to select because the extended domain of locality of TAG requires two different elementary trees for "know," each corresponding to a different subcategorization.

In order to alleviate this problem, it is useful to coalesce different elementary trees into a single representation because at this point, there is no way to tell which is the correct tree to choose. We merge the two elementary trees for the verb "know" by having the substitution node corresponding to its object ambiguously specified as NP or S. See Figure 7(a). We call a single set of assertions with the added proviso of node label underspecification (which represent a set of elementary TAG trees) an *elementary representation.*

The expedient of node label underspecification is also useful in avoiding backtracking in several other situations. Another example is the ambiguity that is associated with PP attachment. Suppose that a PP may either attach at the N' level or the VP level. The problem is that there is a separate elementary tree for each case. With node label underspecification, these separate elementary trees may be coalesced into one elementary representation. Using such elementary representations, we can delay the PP attachment decision until all of the necessary disambiguating information is encountered, as in Figure 8.

Other situations require underspecification of structure in addition to node label underspecification. In general, such cases occur when there is an optional argument or an optional modifier. For example, the verb "wash" optionally subcategorizes for an NP. Another instance occurs with the sentence prefix "The wedding marked..." There is ambiguity in whether the reduced relative or the main verb tree for "marked" ought to be chosen. Notice that parsing free word order languages also introduces this problem of tree selection. The ambiguity in selection of tree structure may even cross part of speech boundaries. This is exhibited in the sentence pairs "Fruit flies like an orange" and "Fruit flies through the air." The relevant trees for the word "fruit" are shown in Figure 7(b).

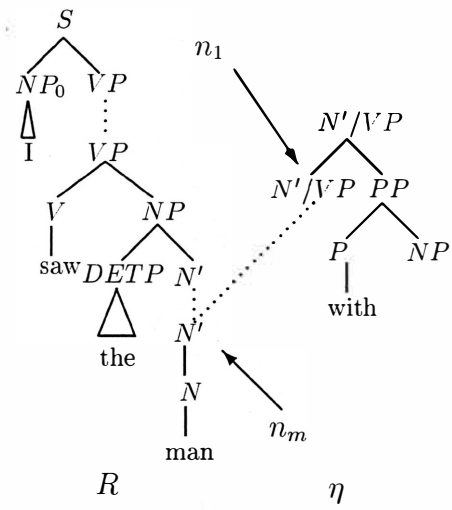


Figure 8: Node label underspecification allows us to represent PP attachment ambiguity.

We are exploring various possibilities for representing this structural ambiguity. One possibility is to license only those parts of an elementary tree up to a certain projection. Take the trees in Figure 7(b) for example. Recall that in the case of seeing the word "know," we underspecified the argument label because it was unavailable. Similarly, upon encountering the word "fruit" we underspecify the set of two elementary trees because only projection up to the N level is licensed. The method suggested by [Evans and Weir, 1997] can perhaps be used to coalesce these structures.

8 Conclusions and Future Work

In this work, we consider parsing TAG using underspecified descriptions of trees featuring dominance and underspecified node labels. The prediction component of an Earley parser corresponds to our assertion of domination. The completion component of an Earley parser does not have an analogue in our parser; it is a manifestation of how an Earley parser considers all possible parses while our parser leaves the situation underspecified. Specifically, in order to underspecify the attachment, we find that in most of the cases in Section 5.2, our parser does not immediately perform TAG operations of substitution and adjoining. That would require equating nodes from two different trees. Instead, our parser asserts domination that leaves us uncommitted to a particular attachment site. Nevertheless, when our parser recovers a parse tree from the underspecified representation, it is necessary for our parser to ensure that the resulting tree corresponds to one derived using substitution and adjoining.

In contrast with traditional work in TAG parsing, we envision our parser to be one that delays attachment decisions. However, we would not like our parser to leave the situation completely underspecified, but to reduce the search space through the judicious use of extra information. This is our notion of reduced commitment, the idea that the underspecified representation that the parser constructs encodes only those relations that the parser strongly believes must hold for the entire sentence. This extra information may include semantics, information about thematic roles, selectional restrictions, or frequency data that are collected from a parsed corpus. It is a subject for future work.

While our parser draws elements from D-theory, this work also extends beyond current work on D-theory. Current work on D-theory parsers do not use any explicit grammar formalism or grammar. Our choice of TAG was motivated by several factors. First, TAGs in general have the desirable properties of factoring recursion and localizing dependencies, which provide argument and structural requirements that are needed by D-theory style parsers for making its assertions regarding structures. Second, TAGs encode linguistic knowledge in a declarative format, making it easier to develop and modify than a more procedural design. Third, extensive TAG grammars have already been developed which could be quickly adapted to the deterministic parser that is proposed here.

Our use of TAG notwithstanding, we use dominance in a different manner than is customary with D-theory parsers. D-theory parsers usually construct a treelike description. On the other hand, some of the descriptions that our parser constructs are not necessarily treelike. This was necessary to capture attachment site ambiguity. In these cases, a statement is made that the tree which may be ambiguously placed dominates the lowest possible attachment site. Consequently, this site would be dominated by two structures, each of which is ambiguously ordered with respect to domination.

This ambiguous representation had a direct impact on our computation of the standard referent, another area where we deviate from normal D-theory parsers. We have provided a means of determining a standard referent under these exceptional circumstances. Our solution appears to produce a structure that conforms to right association, which is not the case for all other D-theory parsers. Not only may our standard referent aid in incremental interpretation but in our case it also aids in deciding how representations may be combined.

We have also discussed how representations may be combined through the use of expectation lists. We have provided a taxonomy of expectations. These help define a number of cases that illustrate when domination may be postulated between components of different elementary trees. We have enumerated many of these cases and given examples of their application.

Finally, we have argued that in incorporating TAG into D-theory style parsing, the assumptions that are standardly made about TAG trees can interfere with the goal of delaying attachment decisions. To alleviate this situation, we have suggested how several TAG trees should be coalesced into one elementary representation. An elementary representation represents elementary TAG trees that differ only in their node labels. This underspecification essentially allows the parser to delay the choice of the specific elementary TAG tree until further information can be obtained. We have also enumerated other cases where node label underspecification is insufficient. We have postulated a solution based on limiting how much of a projection of an elementary tree that we would license in a given situation. This is another subject for further investigation.

References

[Evans and Weir, 1997] Evans, R. and Weir, D. (1997) Automaton-Based Parsing for Lexicalized Tree Adjoining Grammars. In *Proceedings of the Fifth International Workshop on Parsing Technologies.* Cambridge, Massachusetts.

[Frazier, 1995] Frazier, L. (1995) Issues of Representation in Psycholinguistics. In J. L. Miller and P. D. Eimas (Eds.), *Speech, Language, and Communication* (pp. 1-27). Academic Press, New York, New York.

[Gorrell, 1995] Gorrell, P. (1995) *Syntax and Parsing.* Cambridge University Press, United Kingdom.

[Marcus, Hindle, and Fleck, 1983] Marcus, M., Hindle, D., and Fleck, M. (1983) D-Theory: Talking about talking about trees. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics* Association for Computational Linguistics, Cambridge, Massachusetts.

[Rambow, Vijay-Shanker, and Weir, 1995] Rambow, O., Vijay-Shanker, K., and Weir, D. (1995) D-Tree Grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics.* Association for Computational Linguistics, Cambridge, Massachusetts.

[Rogers and Vijay-Shanker, 1996] Rogers, J. and Vijay-Shanker, K. (1996) Towards a Formal Understanding of the Determinism Hypothesis in D-Theory. In H. Bunt and M. Tomita (Eds.), *Recent Advances in Parsing Technology* (pp. 59-78). Paper presented at IWPT '93. Kluwer Academic Publishers, Boston, Massachusetts.

[Sturt and Crocker, 1996] Sturt, P. and Crocker, M. (1996) Monotonic Syntactic Processing: A Cross-Linguistic Study of Attachment and Reanalysis. *Language and Cognitive Processes.*

[Vijay-Shanker, 1992] Vijay-Shanker, K. (1992) Using Descriptions of Trees in a Tree-Adjoining Grammar. *Computational Linguistics*, 18(4), 481-517.

Controlling Bottom-Up Chart Parsers through Text Chunking

Fabio Ciravegna, Alberto Lavelli
Istituto per la Ricerca Scientifica e Tecnologica
Loc. Panté di Povo, I-38050 Trento, Italy
Email:{cirave|lavelli}@irst.itc.it

Abstract

In this paper we propose to use text chunking for controlling a bottom-up parser. As it is well known, during analysis such parsers produce many constituents not contributing to the final solution(s). Most of these constituents are introduced due to the parser inability of checking the input context around them. Preliminary text chunking allows to focus directly on the constituents that seem more likely and to prune the search space in the case some satisfactory solutions are found. Preliminary experiments show that a CYK-like parser controlled through chunking is definitely more efficient than a traditional parser without significantly losing in correctness. Moreover the quality of possible partial results produced by the controlled parser is high. The strategy is particularly suited for tasks like Information Extraction from text (IE) where sentences are often long and complex and it is very difficult to have a complete coverage. Hence, there is a strong necessity of focusing on the most likely solutions; furthermore, in IE the quality of partial results is important.

1 Introduction

Bottom-up parsers spend large amounts of time in building constituents not contributing to the final solution(s). Many of such constituents are produced due to the inability of the parser to take into account the surrounding context. For example, in the sentence `John Smith works in the field of language processing` a standard bottom-up parser builds an S constituent spanning just `John Smith works in the field of language`, a constituent not useful for the final correct parse tree (so as many of the constituents subsumed by it). The impact of such constituents on parser performances is relevant when long sentences are analyzed.

One possible alternative solution is that of adopting strategies that integrate the advantages of both bottom-up and top-down strategies (see [Wiren, 1987] for a comparison of different chart-based parsing strategies). Such strategies, however, do not show to work very efficiently or robustly on long and complex texts, especially when the grammar coverage is not expected to be complete (as is often the case in applicative domains such as Information Extraction from text, henceforth IE).

Chunk Parsing [Abney, 1995] is a technique that mostly allows to overcome the problem of effectively taking into consideration the surrounding context. Chunks are portions of text close to the structures highlighted by psycholinguistic and phonological studies. The chunk parser is designed so to analyze first of all these chunks by using a limited grammar (recognizing constituents such as NPs, PPs, etc.). In a second step it builds clause structures composing chunks by using theta-roles, in a dependency-like fashion. Such strategies render a text largely unambiguous. We think that there are some drawbacks in the original Chunk Parsing proposal:

- it requires the definition of a specifically suited parsing strategy (i.e., a strategy that partially builds constituents and partially dependency structures);
- it requires the use of a non standard grammar, as - for the same reason as above - some of the operations are performed on constituents in a standard way, some others are performed in a non standard way; this obviously prevents the re-use of already existing grammars;
- a deterministic behavior is provided for constituent attachment (e.g. PP attachment) [Abney, 1991]; unfortunately, in analyzing complex sentences effective heuristics for attachment are difficult to be found; very often true ambiguity arises and it is then necessary to deal with it in a proper way.

The idea proposed in this paper is that of providing an agenda-based bottom-up chart parser with the ability of performing its analysis in a chunk based fashion, while maintaining the general characteristics of chart parsing (i.e. the standard algorithm, the ability of coping with ambiguity and the possibility of re-using existing grammars). The proposal fits within a well-established experience of chart-based parsing techniques at IRST [Satta and Stock, 1994] and is a first response to the challenges put to a unification-based approach by the necessity of analyzing real texts.

The parser performs its analysis in consecutive steps and at each step it behaves as follows:

- partial constituents are built by using only the constituents produced during the previous step;
- only some of the possible tasks are activated; the parser chooses among the most promising ones, delaying or pruning the others.

This strategy allows to consider at each time only a limited set of paths in the search space, so to reduce the ambiguity seen by the parser at a given time.

In order to select the constituents to be built (and to be used in the following analysis), the parser control module uses the information provided by an automaton which performs preliminary text chunking. Aim of such preliminary chunking is that of identifying the input areas where certain types of constituents (such as NPs, PPs, etc.) are likely to be found and to divide the input in sub-chunks that are likely to be clauses[1]. The parser attempts first of all to build a (set of) solution(s) that fulfills the expectations selected by the chunking algorithm: at three different steps NPs, clauses and sentences are recognized; if the expectations are fulfilled (e.g. some complete parse trees are found that cover the sentence under analysis), the paths in the search space that are not likely to provide equivalent solutions are pruned out and the found solution(s) are returned.

For example in the case of the sentence `John Smith works in the field of language processing` the idea is that of temporarily hiding the availability of the NP [`language`] while promoting the NP [`language processing`]. During the first step the NPs [`John Smith`], [`field`] and [`language processing`] are recognized. During the following step these NPs are promoted to the prejudice of others such as [`John`], [`Smith`] and [`language`]. Then PPs are recognized and promoted during the clause recognition[2] so that [`in the field`] and [`of language processing`] are promoted. Since just one clause is present in the sentence, the third step (sentence analysis) is not necessary. Given that a complete parse tree is found, at the end of clause analysis the tasks that require the use of constituents spanning [`John`], [`Smith`], [`language`] and [`the field`] are pruned from the agenda.

If a solution has not been found after visiting the paths in the search space recommended by the chunker, different possibilities are envisaged:

- all the remaining paths are visited;
- only a limited number of paths are visited choosing among the most promising ones not yet pursued (i.e. those that are not too far from the chunking expectation);
- the best partial solutions are chosen among those already found by the parser and no additional steps are performed.

The best strategy to be adopted depends on both the characteristics of the corpus under analysis and the requirements set for the parser. For example when the grammar is likely to be more or less complete for the corpus and completeness of the analysis is preferred to efficiency, all alternative solutions can be searched; on the other hand, when the lack of coverage is likely to be frequently experienced and/or efficiency is one of the requirements for the parser (as it is often the case in IE), one of the other two strategies can be selected.

The aim of this paper is to analyze how the three step analysis schema can be introduced in an agenda-based bottom-up chart parser without:

- imposing requirements on the grammar structure or content (so to allow the re-use of preexisting resources);

[1] Throughout the paper we will use the term *clause* not in the usual linguistic sense, but in an informal way that will become clear in the following.

[2] DP and PP recognition is where mostly our approach differs from standard chunk parsing; in the following we will clarify the difference.

- requiring modifications in the general parsing algorithm, except in the agenda manager - a quite limited piece of software - (so to make it applicable to any agenda-based bottom-up chart parsers).

In the following we will:

- formalize the definition of each chunk level (Section 2);
- analyze the impact on the parsing algorithm (Section 3);
- present some preliminary experimental evidence in terms of efficiency and effectiveness of the three step strategy on real world texts (Section 4);
- present some conclusions and related research (Section 5).

2 The Four Levels of Chunks

Abney, starting from a cognitive point of view, identifies three levels of chunks, broadly corresponding to:

- **level 0** words;
- **level 1** non recursive NPs, DPs, PPs and Verb Groups (VGs);
- **level 2** clauses.

We have modified Abney's original proposal, giving a more grammatically oriented definition of chunks; in particular we split the definition of level 1 in two because, as we will see in the following, in a standard parser elements such as DPs and PPs are to be treated differently from NPs and VGs:

- **level 0** *Word-Chunks*: words (ignored in this paper);
- **level 1a** *Head-Chunks*: non recursive NPs and VGs;
- **level 1b** *Mod-Chunks*: non recursive PPs, DPs and Complex VGs (CVGs);
- **level 2** *Clause-Chunks*: clauses.

In the following subsections we will provide a more precise definition of each level for Italian.

2.1 Head-Chunks: NPs and VGs

Definition[3]:

$$\begin{aligned} \text{NP} &\rightarrow (\text{adv}^* \ \text{adj}^+)^* \ \text{noun}^+ \ (\text{adv}^* \ \text{adj}^+)^* \\ \text{VG} &\rightarrow (\text{adv})^* \ \text{verb} \ (\text{adv})^* \end{aligned}$$

Very often constituents built by separately analyzing Head-Chunks are found as they are inside the final parse tree. Consider the following example:

Example 1 `di una società italiana del gruppo XYZ` [4]

In Figure 1 the parse tree for Example 1 is shown: a complete constituent (the NP spanning `società italiana`) recognizes the Head-Chunk highlighted. For this reason Abney's proposal of recognizing such chunks separately still holds also when using a standard parser, because - if chunking is done correctly - the resulting constituents will contribute to the final parse tree.

[3] The regular expressions used in the definitions are just meant to be representative of the set of rules used by the chunker. In the regular expressions the following operators are employed: * (Kleene star), + (with the usual meaning), and ? (for optionality).

[4] Literally, `of a company Italian of-the group XYZ`.

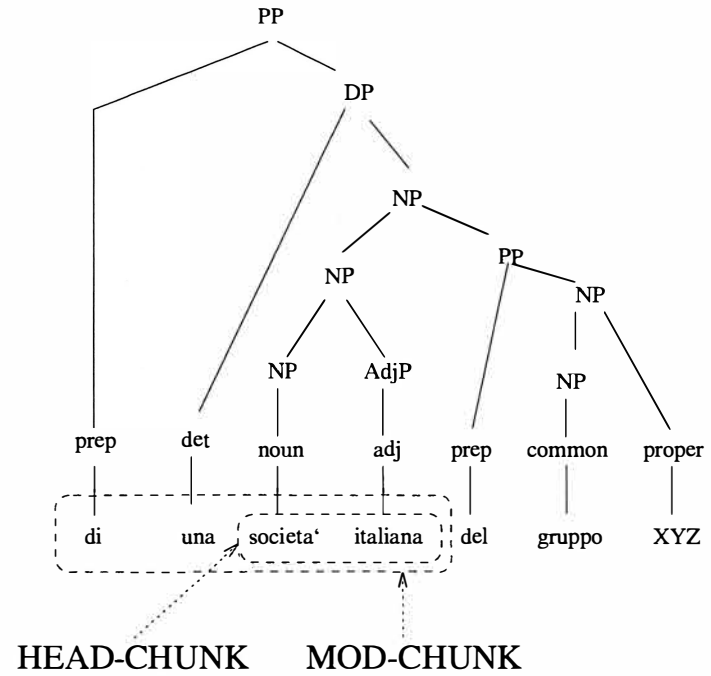


Figure 1: Parse tree (with two chunks highlighted) for Example 1.

2.2 Mod-Chunks: DPs, PPs and Complex Verb Groups

Mod-Chunks contain Head-Chunks:

$$\begin{array}{lcl} \text{DP} & \rightarrow & \text{Det}^{?}\ \text{NP} \\ \text{PP} & \rightarrow & \text{Prep Det}^{?}\ \text{NP} \\ \text{CVG} & \rightarrow & (\text{adv})^{*}\ (\text{auxiliary})^{*}\ \text{VG} \end{array}$$

Often constituents spanning exactly a Mod-Chunk are not found as they are within the final parse tree. In Figure 1 there is no PP covering exactly the Mod-Chunk shown. The PP spanning it also spans other parts of the input not within the same chunk (i.e. the PP spanning `del gruppo XYZ`). What is provided by Mod-Chunks is the coverage of DPs and PPs when they do not include any modifier of the same level. The consequence is that it is not possible to recognize Mod-Chunks separately because the resulting constituents may not contribute to the final tree.

2.3 Clause-Chunks

Clause-Chunks identify the structures of both main and subordinate clauses. Most of the problems in constituent attachments arise within Clause-Chunks. Chunking points are mainly commas, relative pronouns, the Italian complementizer `che` (more or less equivalent to the English complementizer `that`), some types of conjunctions, and some domain-specific cue words. For example, in the sentence:

Example 2 `Corporacion Banesto, holding industriale e di servizi del Banco Espanol de Credito, avrebbe perso nel '93 103,4 miliardi di peseta, circa 1.200 miliardi di lire, dopo l'utile di 5,6 miliardi del '92.` [5]

[5] Literally. `Corporacion Banesto, holding industrial and of services of Banco Espanol de Credito, would-have lost in '93 103.4 billion of pesetas, about 1,200 billion of lire, after the profit of 5.6 billion of '92.`

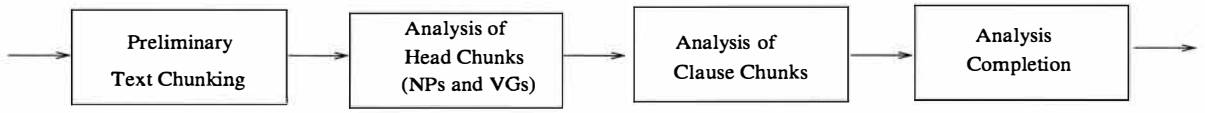


Figure 2: Outline of the overall approach.

chunking after commas produces the following result:

Chunk1: `Corporacion Banesto,`
Chunk2: `holding industriale e di servizi del Banco Espanol de Credito,`
Chunk3: `avrebbe perso nel '93 103,4 miliardi di peseta,`
Chunk4: `circa 1.200 miliardi di lire,`
Chunk5: `dopo l'utile di 5,6 miliardi del '92.`

Chunk2 and Chunk4 identify appositive clauses, whereas Chunk1 and Chunk3 are two parts of the main clause; Chunk5 identifies a PP modifying the main sentence.

2.4 The Chunking Process

The chunking process is performed via a finite state automaton. In case of uncertainty between closing a chunk or extending it with the next word, a greedy strategy is used, i.e. the next element is included in the current chunk. Currently the rules are manually encoded. The chunker relies on the output of a Part of Speech Tagger to solve lexical ambiguity.

3 Parsing, Chunks and Agenda Control

As said in the introduction, our aim is that of defining a control algorithm that allows a bottom-up chart parser to visit the search space so that:

1. *satisfactory solutions* are pursued as firsts;
2. non satisfactory solutions can be pruned out from the search space;
3. the quality of partial solutions is maximized (when a complete solution is not available).

We propose text chunking as a way for providing an estimation of the satisfactoriness of solutions. The analysis is divided in three steps; at each stage satisfactory solutions (when found) are promoted during the following step to the prejudice of other solutions. A recovery action is provided for cases where such solutions are not found. An outline of the three step analysis (plus preliminary chunking) is shown in Figure 2.

Before running the parser, preliminary text chunking is performed via a finite state automaton. The output of this step is a collection of chunking points, i.e. the boundaries for Head-Chunks, Mod-Chunks and Clause-Chunks. Such output is used by the control mechanism of an agenda-based bottom-up chart parser. In agenda-based chart parsers the control relies on activating/delaying tasks in the agenda. A task is the combination of two adjacent edges; each pair of edges represents two (groups of) adjacent elements of the right hand side of a rule. One of the two edges will contain the head of the rule and is defined here as the head edge; the other edge is defined here as modifier edge.

The control algorithm requires the parser make the following steps:

- Step 1: Analysis of Head-Chunks;
- Step 2: Analysis of Clause-Chunks;

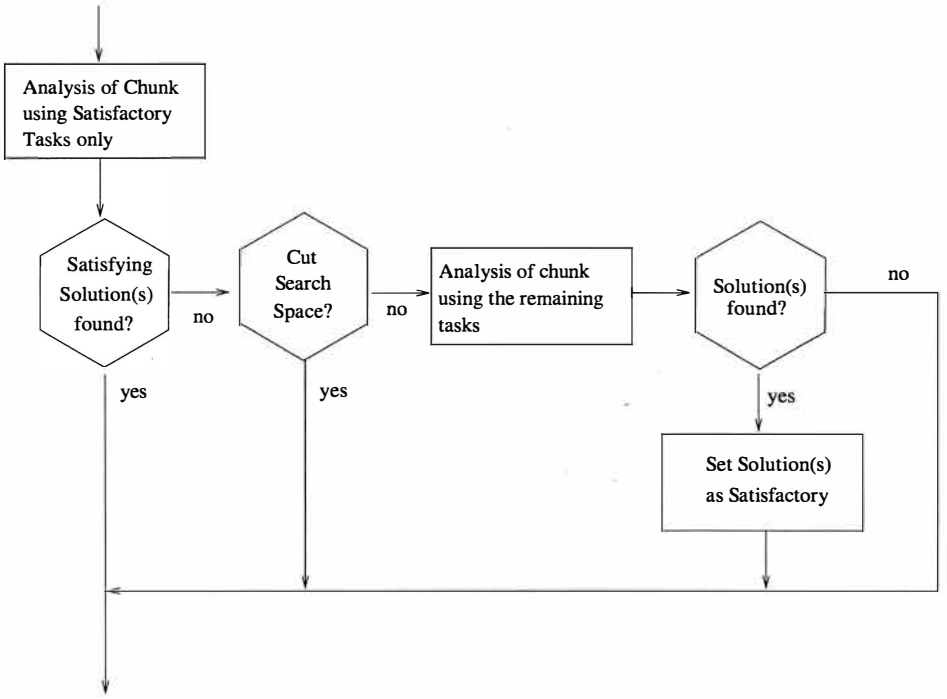


Figure 3: Outline of the algorithm for the analysis of a chunk (of any type).

- Step 3: Text Analysis Completion.

During *Step 1* the parser analyzes each Head-Chunk separately. Tasks that require to go beyond the chunk limits are delayed. If chunking is done correctly, this step produces complete constituents spanning the entire chunk that will be found as they are in the final parse tree. A constituent spanning exactly a Head-Chunk is defined here as *HEAD-CONST*.

During *Step 2* each Clause-Chunk is analyzed separately. The following two rules are used:

RULE0:

head edge is a generic edge (e.g. a preposition, an article, etc.) not spanned by any HEAD-CONST;

modifier edge is at least a HEAD-CONST;

both the edges are within the same Mod-Chunk.

and

RULE1:

head edge is a HEAD-CONST;

modifier edge is MOD-CONST.

We define a MOD-CONST as an edge either built by following RULE0 and spanning a whole Mod-Chunk, or built by following RULE1.

Tasks trying to build MOD-CONSTs are favored in Clause-Chunk recognition; a MOD-CONST spanning a complete Clause-Chunk is defined as a CLAUSE-CONST.

During *Step 3* the completion of text analysis is executed; in this phase the parser executes all the tasks delayed in the previous steps, i.e. it executes those tasks that require to overcome the constraints imposed by any levels of chunks. But the agenda is ordered so as to favor those tasks following:

RULE2:

the head is at least a HEAD-CONST and the modifier is a CLAUSE-CONST.

A constituent built by following RULE2 is also a CLAUSE-CONST. The analysis stops when the agenda is empty.

3.1 Some Considerations

Head-Chunks identify input portions where constituents contributing to the final parse tree can be recognized. After being recognized during Step 1, for the rest of the analysis these constituents are preferred to those subsumed by them. This strategy allows to declare portions of input as non separable units. For example, in:

Example 3 `La Dhl International del network DHL consegue un utile lordo` [6]

a Head-Chunk spans `utile lordo`. Favoring the NP that covers it exactly means favoring the construction of the S constituent spanning the entire sentence, while delaying the production of an S constituent spanning only `La Dhl International del network DHL consegue un utile`.

Mod-Chunks pick the minimum area in which modifiers are found. Minimum area means that chunks identify the portion of the sentence spanned by such constituents if they do not have any modifiers either. This allows delaying the use of constituents spanning sub-chunks of Mod-Chunks as modifiers. For example it delays the use of NPs internal to a pre-verbal PP as verbs subjects. In Example 3 the combination

$$(network\ Dhl)^{modif} + (consegue\ un\ utile)^{head}$$

is delayed because `network DHL` is a sub-chunk of the Mod-Chunk spanning `del network DHL`.

Clause-Chunks help in the complex problem of connecting constituents. In defining Clause-Chunks we start from an intuition about some cue words: let's consider for example commas. In most languages commas do not follow precise rules, nevertheless they are largely used in texts for introducing intonation pauses; very often commas delimit portions within sentences too long to be read as a whole. Instead of trying to capture in the grammar the information brought by commas (quite a difficult task for many languages), we give them a status of preference feature to be used in constituent attachment; i.e. we prefer attachments of CLAUSE-CONSTs rather than attachments of subconstituents of CLAUSE-CONSTs.

In the text about Banesto (Example 2) the use of constituents that entirely span chunks 2, 4 and 5 (i.e. appositives for 2 and 4 and a PP for 5) is favored. As a matter of fact it seems obvious to a normal reader that none of the PPs inside Chunk5 (except the first one) can modify anything outside such chunk: for example `di 5,6 miliardi` cannot modify `perso`. That seems to happen because commas separate the sentence in well defined subparts.

Clause-Chunks are used to capture such separation. By using the limits imposed by Clause-Chunks, most attachments are done without the need of a comparison with other possibilities involving their subconstituents, greatly reducing the problem of attachment when the complexity of the sentence grows.

3.2 A Refinement: the Direction of Analysis

Building Clause-Chunks separately means not taking into account that the constituents internal to the Clause-Chunk under consideration may have as modifiers some of the other CLAUSE-CONSTs. In the text about Banesto (Example 2) Chunk4 modifies the NP spanning `103,4 miliardi` in Chunk3. Building Chunk3 without considering the rest of the sentence means completely recognizing such NP without attaching one of its modifiers. In general this means regularly sending the parser down a garden path when the text is composed by many Clause-Chunks.

In Italian (the language we are working on) modifiers appear mainly to the right of the head; hence we build CLAUSE-CONSTs starting from the rightmost. In this way when considering the constituents internal to a Clause-Chunk, any CLAUSE-CONST to its right is available for attachments. For this reason tasks composing CLAUSE-CONSTs external to the current Clause-Chunk with edges internal to the current Clause-Chunk are also considered during Step 2. In Example 2 Chunk3 is recognized after Chunk4 and Chunk5 and the composition of parts of Chunk3 with Chunk4 and Chunk5 are considered during Step 2.

[6] Literally, `The Dhl International of-the network DHL achieves a profit gross.`

3.3 Further Details

As mentioned, the aim of the control algorithm is first of all that of rapidly focusing on *satisfactory solutions*. Satisfactory solutions are defined as follows:

- for a word: a lexical edge or its projections;
- for a Head-Chunk: a HEAD-CONST;
- for a Clause-Chunk: a CLAUSE-CONST;
- for a sentence: a CLAUSE-CONST spanning the whole sentence.

To focus on satisfactory solutions, during the three steps of analysis tasks are classified as[7]:

- *Exec-Curr-Step*: a task with immediate evaluation; it involves the composition of edges declared as satisfactory solutions for some sub-chunks; the edge that results from the task execution will still span within the current chunk;
- *Delay-Curr-Step*: a task delayed in the current step; a composition involving non satisfactory solutions for some sub-chunks; resulting edge still spans within the current chunk;
- *Exec-Next-Step*: a task to be favored in the next step; composition involving satisfactory solutions for the current chunk or some of its sub-chunks; resulting edge no longer spans within the current chunk;
- *Delay-Next-Step*: a task delayed in the next step; composition involving non satisfactory solutions; resulting edge no longer spans within the chunk.

Table 1 summarizes the task classification algorithm for the three steps of analysis.

Class/Step	Step 1	Step 2	Step 3
Exec-Curr-Step	INT	INT RULE0-1 or INT-EXT RULE2	INT RULE2
Delay-Curr-Step	none	other INT	other tasks
Exec-Next-Step	INT-EXT RULE0-1	EXT RULE2	none
Delay-Next-Step	other INT-EXT	other tasks	none

Table 1: Task classification during analysis: INT tasks involve edges only spanning within the current chunk; INT-EXT tasks compose an edge of the current chunk with one external to it. EXT tasks involve an edge (generated by the execution of INT-EXT tasks) spanning more than the current chunk and an external edge. RULE0-1 tasks respect the conditions imposed by RULE0 or RULE1.

3.4 Agenda Pruning

Until now we have shown how pursuing the satisfactory solutions as firsts. In this paragraph we will see how to reduce the search space. This is done either when at least one satisfactory solution is found or when we are sure that no satisfactory solution can be found for a specific chunk. The general idea is that of pruning the non satisfactory solutions for the chunks for which a satisfactory solution is available. For the other chunks only a limited number of tasks are executed after the satisfactory solution has demonstrated to be not available; the number of tasks activated is a function of the complexity of the current chunk (e.g. the length of the spanned input).

At the end of the analysis of each chunk (of any type) all the Exec-Curr-Step tasks have been activated. The other tasks are then reclassified according to the algorithm shown in Tables 2 (for cases when a satisfactory solution is found) and 3 (for the other cases).

The final algorithm for task classification (inclusive of agenda pruning) is shown in Figure 4.

[7] Such classification is performed when tasks are generated and inserted into the agenda.

	Step 1	Step 2	Step 3
A: Exec-Curr-Step	activated	activated	activated
B: Delay-Curr-Step	removed	removed	removed
C: Exec-Next-Step	→ 2A	→ 3A	none
D: Delay-Next-Step	removed	removed	none

Table 2: Task reclassification at the end of each step when a satisfactory solution was found for the current chunk; → 2A means that before starting Step 2 the tasks contained in that cell are reclassified as Exec-Curr-Step.

	Step 1	Step 2	Step 3
A: Exec-Curr-Step	activated	activated	activated
B: Delay-Curr-Step	activated	activated	activated
C: Exec-Next-Step	none	none	none
D: Delay-Next-Step	→ 2A	→ 3A	none

Table 3: Task reclassification at the end of each step when NO satisfactory solution was found for the current chunk. → 2A means that the current tasks are reclassified during Step 2 as Exec-Curr-Step.

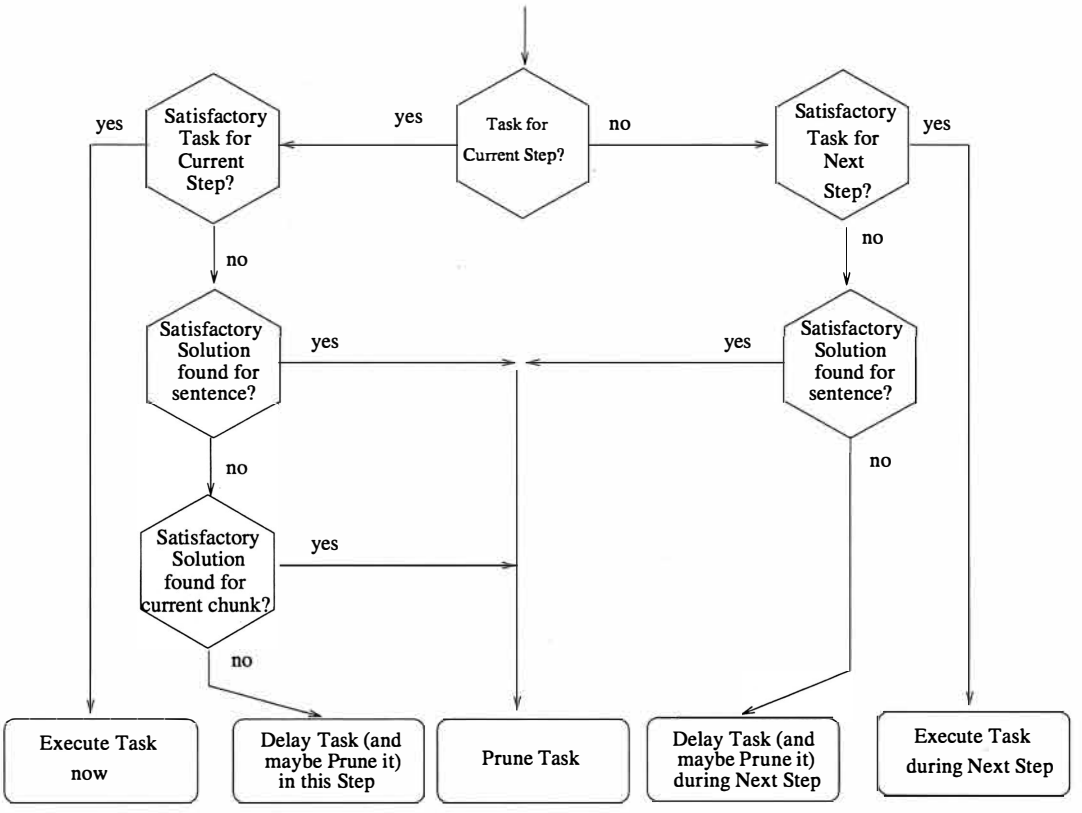


Figure 4: Task Classification Algorithm.

Parsing Strategy	Total edges	average/text	% produced
NON controlled	350,644	2,874	100%
Controlled	110,511	905	32%

Table 4: Edges produced in analyzing 122 sentences.

Parsing Strategy	Global Time	average/text	% Time
NON controlled	06:54:18	00:03:26	100%
Controlled	01:32:25	00:00:45	22%

Table 5: Time consumed in analyzing 122 sentences.

4 Preliminary Experiments

We have carried on some experiments on the effectiveness of the strategy proposed in this paper. Given the lack of corpora for Italian annotated such as in TreeBanks, the amount of data used in such experiments is quite limited, and therefore the results are only preliminary. We plan in the near future to carry on more extensive experiments in order to verify the effectiveness of the approach.

The chunk-based control strategy was added to a CYK-like parser; the parser uses a grammar based on a Typed Features Logic (TFL) [Carpenter, 1992] oriented formalism; the control strategy effectiveness was compared with that of the standard control mechanism in analyzing 122 sentences taken from articles of the Italian financial newspaper "IL SOLE 24 Ore" and news provided by ANSA. The total number of words in the corpus is 2769, an average of 23 words for sentence. Table 4 and 5 show the results of the comparison.

As it may be seen, there is a reduction of about 68% of constituents (edges) generated and of 78% of time consumed. Note that both parsers were constrained to produce less than 10,000 edges per sentence; the non-controlled version reached 21 times such limit, against 4 of the controlled; this means that a completely uncontrolled parser could generate more edges and spend more time than showed and that the effectiveness of the strategy could increase in the tables.

Concerning correctness of the results, this is the most difficult part to be tested given the lack of annotated corpora for Italian. We have manually checked the correctness of the results on 31 sentences among the most complex of our corpus. The controlled version of the parser was able to find 29 correct solutions missing 2. The two errors were due to the pruning strategy based on Clause-Chunks. In both cases the parser missed to produce a complete parse tree and produced 2 meaningful partial parses. The non-controlled parser produced 27 correct solutions out of 31; missing 4 solutions when the upper limit of 10,000 edges was reached. In such cases the quality of the partial parses was kind of random, due to the LIFO strategy in activating the tasks in the agenda: some parts of the input were fully analyzed, other were completely unanalyzed; with a FIFO strategy the partial results could be better, but the number of the missing solutions could increase as the edge limit could be reached for other sentences. The inability of the non-controlled parser to find in some cases the correct solutions due to the upper limit of 10,000 edges is an interesting point. Some other experiments show that reducing the limit to 9,000 the number of missing solutions grows for the non controlled parser; on the contrary, for making the controlled parser losing further solutions it is necessary to bring that limit under 4,000. Further experiments on the 122 sentences show that, establishing for the controlled parser an upper limit of 4,000 edges (instead of 10,000), only 25% of the edges were produced and 12% of the time was consumed, if compared with the non-controlled one (with the limit of 10,000 edges). We are currently investigating the impact of the 4,000 limit on the correctness of the results on all the 122 sentences. Anyway, considering that we use the controlled parser in an application of Information Extraction from text (where there are demanding time and space constraints and complete solutions are not necessarily required) and considering that the analysis of only 10 out of 122 sentences exceeded the limit of 4,000 edges, we are confident that such limit can provide a suitable trade-off between parsing efficiency and effectiveness.

As for the relationship between sentence length and effectiveness of the proposed approach, some qualitative considerations can be drawn. As said above, the sentences were taken from two different sources, the short news of a financial newspaper and a news agency. In the former case, sentences are short (the length is usually less than 25 words), the text grammatical structure is plain and the linguistic phenomena used are very often completely dealt with by the grammar; the improvement introduced by our strategy on these texts is relevant.

As for the latter case, agency news are usually longer (35 words on average) and much more complex from a linguistic point of view: nested clauses (such as nested incidentals) and long PP sequences are frequent; the grammar very often does not completely recognize the sentences. In this case the strategy seems to be insufficient in the part relying on the clause level chunking, as - especially when confronted with nested clauses - the current strategy is too straightforward and needs to be refined. Anyway the cut in the search space is always relevant, especially when the grammar does not cover the sentence.

5 Final Remarks

This paper shows how it is possible to modify Abney's definition of Chunk Parsing so to allow its use within a framework such as standard chart parsing. The proposed approach maintains the advantages of Chunk Parsing (mainly the reduction of ambiguity) avoiding most of its problems such as the use of a mixed constituency/dependency parsing and the complexity of constituent attachment at the clause level. The performances of the chart parser based on preliminary chunking outperformed those of one adopting a traditional LIFO strategy for ordering tasks in the agenda, without significantly losing in correctness. Introducing the control strategy in a pre-existing parser required a limited effort (about one person/month); the chunking grammar accounting for three levels of chunking is composed by about 20 rules in all. The controlled parser is now used within a system for Information Extraction (IE) from economical news: more precisely it is used in FACILE [FACILE-Team, 1996], a 3-year project funded by the European Union within the framework of Language Engineering. The goal of the project is to build a set of tools for categorization of financial texts and information extraction from them. The general aim of an application derived from such tools is message dispatching and routing for financial institutions (banks, trading companies, rating companies). Relevant texts include agency news and newspaper articles.

We think that the chunk-based strategy is particularly suited for Information Extraction from text where there is a strong interest in focusing on the most likely phenomena [Lehnert, 1994] even if it can cost in terms of completeness of the analysis. When the parser focuses on the solution proposed by chunking, actually promotes constituents that are supposed to be likely, given the current sentence. The (possible) analysis of parts of the search space outside the chunking hints is an attempt to look also for alternative solutions when they are needed; limiting the number of steps done when looking for alternative solutions is a way for pursuing such solutions only if they are not too distant from the expectation (i.e. if they are not too unlikely). Moreover the experiments show that the partial results produced by the controlled parser are interesting from a qualitative point of view (and the quality of partial results is another requirement for IE).

The use of chunking for controlling the parser is coherent with the current trends in IE towards the use of finite state models of language for helping in analyzing texts: these models have been used for preprocessing (e.g. in proper name recognition [Vilain and Day, 1996]) or even for the whole IE process [Appelt et al., 1993]. In our approach finite state devices are used for chunking input texts: the results are later on used by the control mechanism of the parser in order to decide how to visit the search space.

Other authors use techniques similar to preliminary chunking for information extraction. McDonald [McDonald, 1992] pre-segments the input that is then analyzed by a bidirectional chart parser adopting a semantic grammar. McDonald's segments broadly correspond to our Mod-Chunks and are used in a similar manner. The problem of the complexity of attachment in long sentences (even when using semantic knowledge for solving attachment problems) is not taken into consideration.

Terminal Substring Parsing is used in TACITUS [Hobbs et al., 1992] for avoiding the complete chart-based analysis of very long sentences; sentences are segmented around commas and other function words (i.e. in a way similar to our Clause-Chunk) and segments are analyzed from right to left. Terminal Substring Parsing is adopted only for very long sentences and not for all the sentences as we do; moreover within the segments the overgeneration of false constituents is not controlled as we do by using the information on Head-Chunks and Mod-Chunks.

References

[Abney, 1991] Abney, S. P. (1991). Parsing by chunks. In Berwick, A. and Tenny, editors, *Principle-Based Parsing*. Kluwer.

[Abney, 1995] Abney, S. P. (1995). Chunks and dependencies: Bringing processing evidence to bear syntax. In *Computational Linguistics and the Foundation of Linguistic Theory*. CSLI.

[Appelt et al., 1993] Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., and Tyson, M. (1993). FASTUS: A finite-state processor for information extraction from real-world text. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France.

[Carpenter, 1992] Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, Massachusetts.

[FACILE-Team, 1996] FACILE-Team (1996). Facile project summary. description available at http://www2.echo.lu/langeng/en/le1/facile/facile.html.

[Hobbs et al., 1992] Hobbs, J. R., Appelt, D. E., and Tyson, M. (1992). Robust processing of real-world natural-language texts. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Trento, Italy.

[Lehnert, 1994] Lehnert, W. (1994). Cognition, computer and car bombs: How Yale prepared me for the 90's. In Schank and Langer, editors, *Beliefs, Reasoning and Decision Making: Psycho-logic in Honor of Bob Abelson*, pages 143–173. Lawrence Erlbaum Associates.

[McDonald, 1992] McDonald, D. D. (1992). Robust partial-parsing through incremental, multi-algorithm processing. In Jacobs, P. S., editor, *Text-Based Intelligent Systems*. Lawrence Erlbaum Associates.

[Satta and Stock, 1994] Satta, G. and Stock, O. (1994). Bi-Directional Context-Free Grammar Parsing for Natural Language Processing. *Artificial Intelligence*, 69(1-2):123–164.

[Vilain and Day, 1996] Vilain, M. B. and Day, D. (1996). Finite-state phrase parsing by rule sequences. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 274–279, Copenhagen, Denmark.

[Wiren, 1987] Wiren, M. (1987). A comparison of rule invocation strategies in context-free chart parsing. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark.

PRUNING SEARCH SPACE FOR PARSING FREE COORDINATION IN CATEGORIAL GRAMMAR

Crit Cremers
Maarten Hijzelendoorn
Department of General Linguistics, Leiden University, The Netherlands
{cremers,hijzelendrn}@rullet.leidenuniv.nl

Abstract

The standard resource sensitive invariants of categorial grammar are not suited to prune search space in the presence of coordination. We propose a weaker variant of count invariancy in order to prune the search space for parsing coordinated sentences at a stage prior to proper parsing. This Coordinative Count Invariant is argued to be the strongest possible instrument to prune search space for parsing coordination in categorial grammar. Its mode of operation is explained, and its effect at pruning search space is exemplified.[1]

1 Lexical Ambiguity and Natural Language Parsing

Lexical ambiguity is known to be a major threat to efficient parsing of natural language. It is easy to see why. Let G_{NL} be a grammar for a language NL, and L a lexicon with initial assignment A of nonterminals to the words of NL. Let $S = w_1 \ldots w_n$ be a sentence over L. Under a parsing-as-deduction-approach, then, the parsing problem for S is whether for some sequence $C = c_1 \ldots c_n$, with $c_i \in A(w_i)$, C is derivable under G_{NL}. The solution to the problem may require checking the derivability of many such Cs. Basically, the number of sequences of which the derivability must be checked for a certain S is $\prod_1^n |A(w_i)|$, exponentially dependent on *n*. $\prod_1^n |A(w_i)|$ measures the search space for parsing S.
There is a trade-off between lexical ambiguity and properties of the grammar. In particular, a grammar may assign nonterminals to certain lexically assigned nonterminals, by having monadic rules or theorems of type $c \rightarrow c'$. In that case, the search space for parsing S is partially erected by the grammar itself. Again, the question whether G_{NL} derives S, explodes to a multiplicity of queries as to whether G_{NL} derives a certain C, thus simulating the effect of structural variation in parsing phrase structure grammar. But categorial grammar tends to spell out (or compile) structural variation in the lexicon.
A certain degree of lexical ambiguity, or rather: polymorphism *per* lexical atom, seems inevitable and inherent to the expressive power of natural language. In categorial grammar, for example, differences in subcategorization (a verb may select an infinitival as well as a tensed complement), word order (a finite verb may have its complements to the left or to the right) or functionality (a word may be a preposition or a particle) must lead, in some stage of the parsing process, to branching possibilities of assignments and an increase of search space.

To a large extent, the art of parsing consists in finding secure means to restrict the number of possible assignments. Given the function $\prod_1^n |A(w_i)|$ over strings of words, efficiency requires serious pruning

[1] We are greatly indebted to three anonymous referees for their valuable comments on an earlier version of this paper, and to Michael Redford and Jeroen van de Weijer.

of the search space. Optimally, the means for pruning are anchored in the grammar that is to be applied. Occurrence-sensitive or linear categorial grammar offers some options for pre-checking assignments. In this article, we will discuss a particular instrument that overcomes a flaw in the effectiveness of the standard pre-checks in the presence of coordination. This instrument is developed and tested as a part of a parsing system for Dutch - Delilah - in which the lexicon is the main generator of search space.[2] Some test results are presented in section 4.

2 Categorial Count Invariance and Parsing Coordination

Certain categorial calculi exhibit a property that is known as *count invariance* [Van Benthem 1986]. For these logics, it is true that if a proposition $Y \rightarrow z$ is derivable, the string to the left of $\rightarrow$ and the type to its right share the results of a particular way of counting occurrences of basic types. This count protocol discriminates between positive and negative occurrences of basic types; for each basic type x and for each string of types S it yields an integer representing the occurrences of x in S. Here and below, slash categories are invariantly written in the format *(head)-direction-argument*, where *direction* is indicated by the slash; the bracketing of the head is often suppressed.

(1) **Count Protocol**
for each basic type x,
$count_x(x) = 1$
$count_x(y) = 0$ if y is a basic type and $y \neq x$
$count_x(y/z) = count_x(y\backslash z) = count_x(y) - count_x(z)$
$count_x(y_1, ..., y_n) = count_x(y_1) + ... + count_x(y_n)$.

Because of this way of counting, a complex type x/y will be said to contain a positive occurrence x of type x and a negative occurrence $/y$ of type y. The main application of type count is stated in (2):

(2) **Count Invariance for grammar *G***
if $Y \rightarrow z$ is derivable in some resource sensitive categorial grammar G, then for all basic types x, $count_x(Y) = count_x(z)$. By consequence, if z is a basic type, then for all basic types $x \neq z$, $count_x(Y) = 0$.

Count Invariance can be proved for resource sensitive calculi like Lambek, Lambek+Permutation and Ajdukiewicz/Bar-Hillel, but does, of course, not hold in systems that perform Contraction or Expansion - with $y\ y \rightarrow y$ - and/or Monotonicity - if $X \rightarrow y$ then $X\ x \rightarrow y$; see [Van Benthem 1991]. Count Invariance underlies the notion of balance in proof nets [Roorda 1992]: for a sequent $Y \rightarrow z$ (in Roorda's notation: $y_1...y_n\ z$) to be balanced implies that for every basic type x, $count_x(Y) - count_x(z) = 0$. Consequently, if for some (sub)sequent S $count_x(S) \neq 0$, S is unbalanced in x.
Count Invariance can be used to delimit the search space unfolded by lexical ambiguity [Moortgat 1988]. By contraposition of (2), a proposition $Y \rightarrow z$ cannot be proved in a count invariant system G if for some basic type x, $count_x(Y) \neq count_x(z)$.

[2] This parsing system is available at http://fonetiek-6.leidenuniv.nl/hijzlndr/delilah.html. Among the phenomena it deals with are unbounded coordination, verb clustering including cross-serial dependencies, wh-movement, topicalization and adjunction.

Almost by definition, coordination in natural language involves the multiplication of types: a certain subsequence of types to the left of the coordination point is doubled or mirrored at the right of the coordination point. The relative unrestrictedness of coordination is reflected in the proposal to categorize coordinating elements like *and* as *(X\X)/X*, i.e. by means of essential variables over types [Wittenburg 1986, Moortgat 1988, Steedman 1990, Emms 1993]. Thus they are assigned type frames (polymorphic types) rather than types. We may assume that these frames do not occur negatively in any other type, as coordinations - or free coordinations, at least - are not selected by other elements in a natural language [Grootveld 1993].

This categorization as a frame of type variables also accounts for the doubling of types induced by coordination. This doubling would interfere with Count Invariance: unless the repeated sequences are themselves fully balanced, the count of certain types will be unbalanced by the repetition of subsequences. The variables in the coordinator's type frame *(X\X)/X* are supposed to be instantiated by a type to which both the repeated sequences can be reduced. Since, according to the Count Protocol, the type frame for *and* halves the counts for whatever type substitutes *X*, the effect of doubling is neutralized and Count Invariance is still a property of a coordinated sentence.

Unfortunately, this is not sufficient to keep Count Invariance available as a method to prune the search space for coordination. In fact, Count Invariance (2) needs more information than is available prior to proper parsing, to do its pruning job. It is not difficult to see what prevents it from being effective in the presence of coordination. Consider the following sequent as one of the possible hypotheses emenating from a lexicon concerning the assignment of categories to the words of the coordinated sentence w_1 ... *and* ... w_n, according to which the category c_i is assigned to w_i.

(3) $\quad c_1 \ldots c_i \ldots c_k\ (X\backslash X)/X\ c_{k+2} \ldots c_m \ldots c_n \rightarrow s$

In order to know whether it is sensible to look for a possible proof of this sequent, we would like to check Count Invariance, i.e. the property that for every basic type x, $count_x(c_1 \ldots c_n) = count_x(s)$. Since the sequent contains a coordinator, we know that for the sequent to be derivable, for some i and some m the segment $c_i \ldots c_k$ and the segment $c_{k+2} \ldots c_m$ must have identical count characteristics, which accumulate in the overall count. According to one's theory of coordination and the scope of the intended grammar, one can relax the conditions on the two count characteristics, but the categorial structure of the two coordinates will always be related in one way or another, and will always be dependent on the non-coordinated context. In the spirit of [Sag *et al.* 1985], one could try to define a more general or liberal relation between the categorial characteristics of the coordinates. This would only complicate but not alter the problem of finding the coordinates. For ease of exposure, however, we choose here the most restrictive form of coordination, and require the coordinates to be categorially equal.

The doubling of type occurrences disturbs the count balances. As a matter of fact, the relevant instances of Count Invariance are $count_x(c_1 \ldots c_n) - count_x(c_{k+2} \ldots c_m) = count_x(s)$; here the part of the string counted twice is substracted to restore the balance. In order to check these instances at (3), however, we have to know which is the proper value of m, and of i, for that matter. Since we have not yet parsed the sequent - we are just testing whether it should be parsed - we do not have information as to the borders of the coordinates. It is well known that neither coordinates nor their borders are locally marked - which is a problem for parsing itself anyway [Cremers 1993]. Consequently, we can only guess i and m without having any clue, and apply Count Invariance with respect to each pair. This is easily recognized as a procedure which is as complex as parsing: for every assignment of lexical categories to the sentence, we have to check every possible pair $\langle i, m \rangle$, and if we find a pair for which Count Invariance can be established, this pair marks the borders of the coordination. But this implies that we

(partially) parse that particular assignment while we are pre-checking its derivability. It follows that we cannot use Count Invariance to prune search space, since applying Count Invariance presumes the exploration of search space. In the presence of coordination, Count Invariance is a blunt knife.
It is worth noting that replacing the type frame for coordinators by a set of constant types does not solve the problem. It would just lead to the additional hypothesis that the coordinator type which was (randomly) selected in a certain assignment, complies with the counts for c_i .. c_k and c_{k+2} .. c_m. Again, by lack of a proper parse we can only guess the values i and m. In either case, therefore, Count Invariance (2) faces a comparison of two unknown elements.

This indeterminacy of coordination at the level where Count Invariance is supposed to come in, is a property of natural language coordination as such. It is not caused or provoked by a particular way of categorizing a language. Since almost everything can almost always be coordinated and coordinations are generally neither selected nor selecting, a string of words or categories can contain no immediately accessible hints as to what is coordinated in that particular case.
In the presence of coordination, Count Invariance (2) thus turns out to be of no help at selecting viable sequences prior to parsing. Therefore, we developed a weaker alternative which can be effectively exploited in delimiting the search space for coordinated sentences. It operates at least on the basis of a nonassociative, context-sensitive and bracket-free categorial grammar, as designed in [Cremers 1993]. Its basic properties, however, are definable for any grammar that is sensitive to directionality and occurrence.

3 A Count Invariant for Coordinated Sequences

Coordination can be constructed so as to imply that certain elements outside the scope of the coordination have a double task with respect to elements inside the scope of the coordination: they have to serve elements in both coordinates. Under this approach a coordinator is treated syncategorematically, and thus as combinatorially inactive. For example, in a sequent

(4) $\quad a/b \; b \; [\&] \; b \; d\backslash a \rightarrow d$

the single negative occurrence $/b$ in a/b has to deal with the two positive occurrences of b to the left and the right of the coordinator & which itself does not contribute to balancing the sequent. In fact, we see that b is coordinated in that string, and being so it overcharges, in terms of count balance, its non-coordinated context. By contrast, the negative occurrence $\backslash a$ in $d\backslash a$ is cancelled on a one-to-one basis by the positive occurrence a in a/b; neither a nor $\backslash a$ occurs in the scope of coordination, and they are balanced. This phenomenon is general: in a well-formed coordinated string, being in the scope of coordination determines whether or not the occurrences of a type are balanced (for some relevant lemmas in this respect, see [Cremers 1993, ch. 3.3]. Types occurring in a coordinate and not balanced inside that coordinate, will require certain other types to take care of more than one of them. The outside types, like $/b$ in (4), can be said to have double functions.
By just counting the positive and negative occurrences of primitive types to the left and the right of the coordinator, we can check whether enough suitable double function categories are available. Their presence in the sequent is a necessary condition for grammaticality. For example, if we change (4) into (5) by adding a positive occurrence b to the left of the coordinator which cannot be part of the coordination, the intended double function type $/b$ is no longer available for the coordinated b's, and the sequent must be rejected.

(5) $a/b\ b\ b\ [\&]\ b\ d\backslash a \rightarrow d$

In order to bring about this rejection in an early stage, we only have to know the count values of the intended coordinated substring, since these values will specify the nature of the need for double function primitives. The question is, then, how to determine these 'coordinated' count values prior to proving the corresponding propositions, i.e. prior to deriving the proper coordinates.

For that purpose, we need a particular counting device which keeps track of detailed information on the categorial architecture of an assignment. Suppose we have a string of words of the form $w_1 \ldots w_i$ *and* $w_{i+2} \ldots w_n$. Let L be an assignment of types $c_1 \ldots c_i$ to $w_1 \ldots w_i$; let R be an assignment of types $c_{i+2} \ldots c_n$ to $w_{i+2} \ldots w_n$. Assume that a negative and a positive occurrence of a certain type cancel each other on a one-to-one basis. In a sequent $\ldots x \ldots x \ldots y\backslash x \ldots$ exactly one of the x-occurrences and the $\backslash x$-occurrence cancel each other; the other x is free if there is no other occurrence of $\backslash x$ or $/x$ in the sequent. In this stage of parsing we cannot and do not need to decide which particular occurrence is cancelled against which other occurrence. We are just interested in the numbers of cancellation candidates.
Accordingly, with L can be associated a register reg_L of quadruples of integers <*bound_head, bound_arg, free_head, free_arg*>, such that for each primitive type x there is an x-quadruple specifying:

bound_head^x_L: the number of (positive) occurrences of x in L that are cancelled by (negative) occurrences of x under a \-slash ($\backslash x$) in L;

bound_arg^x_L: the number of (negative) occurrences of x under a /-slash ($/x$) in L that are cancelled by (positive occurrences) of x in L;

free_head^x_L: the number of (positive) occurrences of x in L that are not cancelled by (negative) occurrences of $\backslash x$ or $/x$ in L;

free_arg^x_L: the number of (negative) occurrences of $/x$ in L that are not cancelled by (positive) occurrences of x in L.

Each occurrence (positive or negative) of a type is counted once in its register. The procedure for this way of counting differs from the count protocol (1) underlying Count Invariance (2) in being sensitive to the direction of the negative occurrences. A head (or positive occurrence of) x is counted as free in L if there is no $/x$ to its left or $\backslash x$ to its right in L by which it could be cancelled. Similarly, an argument (or negative occurrence of) $/x$ is free in L if no head to its right can possibly match it. There is no need to count free occurrences of $\backslash x$. Arguments $\backslash x$ cannot be free in L if L is to be the prefix of a derivable sequent, by the directionality invariant of [Steedman 1990]: no rule can change the directionality of an argument type; if an occurrence of $\backslash x$ is free in a prefix, it can never be cancelled by a positive occurrence to its right, and thus it is doomed to remain free. A free argument $\backslash x$ will obstruct any derivation of a sequent in which it occurs. As soon as a free argument $\backslash x$ shows up in L, L can be rejected as a prefix of the sequence of assignments to the sentence.
For R, a similar register reg_R is supposed to be available, though directional parameters are reversed.

Here is an example of a full register reg_L for basic types a, b, c, and of the register for the first three types in L.

(6) L = $a/c\ b/c/a\ b\backslash b\ c\backslash b\ a\ a$
reg_L = { *a:<0,1,2,0>, b:<1,0,0,0>, c:<0,1,0,1>* }.
L_3 = $a/c\ b/c/a\ b\backslash b$ (a prefix of L)
reg_{L3} = { *a:<0,0,1,1>, b:<1,0,1,0>, c:<0,0,0,2>* }

Computing the register is a linear task. It is computed and updated incrementally and accumulatively whenever a new type is added to the prefix. In (6), reg_L reflects the changes in reg_{L3} after adding $c\backslash b$, a and a to L_3.

What can the registers reg_L and reg_R tell us about combining L and R into a hypothesis $L\ [\&]\ R \rightarrow s$? The values for $free\_head^x$ and $free\_arg^x$ in each quadruple provide the number of positive and negative occurrences, respectively, that are not cancelled at their side of the coordinator. These occurrences may or may not be in the domain of coordination. We can determine the occurrence patterns that are characteristic for grammatical, i.e. derivable sequents as follows (for some more formal elaboration on this topic, see [Cremers 1993, ch. 3]. Suppose a free (positive) occurrence a in L is in the scope of coordination. By the nature of coordination, this occurrence has a matching occurrence in R that is necessarily cancelled at its side. For a free occurrence of x in L to be inside the scope of coordination, a potentially cancelling type must be available in the balanced part of R -this type will have a double function. By the same line of reasoning, if the occurrence of x which is free in L, is to be outside the scope of coordination, it has to cancel against some free occurrence in R. We can illustrate the range of possibilities with a simple example; it is assumed that the substrings V, W, X, Y and Z do not contain occurrences of type a.

(7) $X\ a\ Y\ [\&]\ W\ a\ V\ b\backslash a\ Z \rightarrow s$
L = $X\ a\ Y$
R = $W\ a\ V\ b\backslash a\ Z$
a-quadruple in reg_L = <0, 0, 1, 0>
a-quadruple in reg_R = <0, 1, 0, 0>

The a-quadruple in reg_L tells us that some positive occurrence of a is free in L: $free\_head^a{}_L = 1$. This occurrence may be part of the coordinated substring. If it is, there must be some negative occurrence in R that takes care of both a in L and its matching counterpart in R. This negative occurrence, however, is cancelled in R, and must be accounted for in the number of bound arguments $\backslash a$ in reg_R, $bound\_arg^a{}_R$; this number happens to be 1, due to the composition of R . Now suppose the positive free occurrence of a in L is not inside the scope of coordination. Then there must be a negative occurrence $\backslash a$ free in R. This negative occurrence should be counted in $free\_arg^a{}_R$. Since, in example (7), $free\_arg^a{}_R$ has the value 0, the latter assumption is rejected; all the occurrences counted in $free\_head^a{}_L$ must be in the scope of coordination.

This type of reasoning can be generalized in order to decide, under the hypothesis that the coordinated sequent is derivable, how many balanced negative and positive occurrences of basic types in L and R must be in the scope of coordination. Here is the argument.

Let λ_x be the difference $free\_head^x{}_L - free\_arg^x{}_R$, for some basic type x. Clearly, if $\lambda_x > 0$, there are λ_x positive occurrences of x in L which are not cancelled by negative occurrences $\backslash x$ free in R. These λ_x occurrences must therefore be in the scope of coordination and be co-covered by already bound negative occurrences $\backslash x$ in R. In that case, the value of $bound\_arg^x{}_R$ must be at least as large as λ_x. On the other hand, if $\lambda_x < 0$, there must be negative occurrences $\backslash x$ in R that, for the string to be grammatical, must be dealt with by already cancelled occurrences x in L, accounted for in the number $bound\_head^x{}_L$; this number must be large enough to accommodate the $|\lambda_x|$ negative occurrences $\backslash x$ in R. If $\lambda_x = 0$, all or none of the $free\_head^x{}_L$ positive occurrences and $free\_arg^x{}_R$ negative occurrences are in the scope of coordination, depending on other parameters. An equivalent reasoning can be built around the value ρ_x, this being the difference $free\_head^x{}_R - free\_arg^x{}_L$. (As a corollary, $\lambda_x + \rho_x = count_x(C)$ for that proper affix C of L and of R that happens to be in coordination.)

The above reasoning gives us the following inequalities for two assignments L and R:

(8) **Coordinative Count Invariant**

if $L\ [\&]\ R \rightarrow s$ is derivable, then

for every basic type $x \neq s$ such that

$<bound\_head^x_L,\ bound\_arg^x_L,\ free\_head^x_L,\ free\_arg^x_L>$ is in reg_L and

$<bound\_head^x_R,\ bound\_arg^x_R,\ free\_head^x_R,\ free\_arg^x_R>$ is in reg_R and

$\lambda_x = free\_head^x_L - free\_arg^x_R$ and

$\rho_x = free\_head^x_R - free\_arg^x_L$,

it is true that

$\lambda_x \leq bound\_arg^x_R$ and

$\rho_x \leq bound\_arg^x_L$ and

$-\lambda_x \leq bound\_head^x_L$ and

$-\rho_x \leq bound\_head^x_R$.

By contraposition, a sequence of types with a coordinator is not reducible to s if for some primitive type x one or more of the inequalities in (8) does not hold. This justifies the following statement:

(9) **Conjoinability**

An assignment L of types to the words to the left of a coordinator and an assignment R of types to the words to its right are *conjoinable with respect to a basic type x* iff the quadruples for x in reg_L and reg_R satisfy the inequalities of (8).

Two strings of types L and R are *conjoinable* as $L\ [\&]\ R$ iff L and R are conjoinable with respect to every basic type x, $x \neq s$.

This is the invariant that can do the job of selecting coordinated type sequences prior to proper parsing. It requires neither information nor guesses as to the nature and the extent of the coordination in the sentence which is to be parsed.

Given a sentence W_l *and* W_r, a set LL of assignments to W_l and a set RR of assignments to W_r, with registers associated to each member of each set, it is straightforward to select those pairs <L, R> in LL × RR that are conjoinable according to (9). Only these pairs have to be checked for derivability. The pairs rejected as not conjoinable cannot represent a well-formed coordination, by *modus tollens* applied to (8).

Here is a simple example involving just two basic types where both LL and RR contain only two assignments; the designated type s is neglected, as its count deserves special treatment. (10) specifies the four subsequences of assignments to a coordinated sentence of length 7, including the coordinator, and the related registers. (11) - (14) specify the relevant data for a particular member of the product LL × RR, according to the registers in (10), and apply the Coordinative Count Invariant (8) in a *modus tollens* mode. If one of the queries fails, the hypothesis that that particular member of LL × RR gives rise to a derivable sequent, is rejected, by Conjoinability (9).

(10)

LL = {L_1, L_2}	
L_1 = a b/b a\b	reg_{L1} = {a:<0,0,2,0>, b:<1,0,0,1>}
L_2 = a b a\b	reg_{L2} = {a:<0,0,2,0>, b:<1,0,0,0>}
RR = {R_1, R_2}	
R_1 = a\b s\a\a	reg_{R1} = {a:<0,1,0,1>, b:<0,0,0,1>}
R_2 = a\b s\b\a	reg_{R2} = {a:<0,1,0,0>, b:<0,0,0,2>}

(11) hypothesis 1: L_1 *[&]* $R_1 \rightarrow s$ (L_1 and R_1 are conjoinable)

$\lambda_a = 1$; $\rho_a = 0$; $\lambda_b = -1$; $\rho_b = -1$; $\text{bound\_arg}^a_L = 0$; $\text{bound\_arg}^a_R = 1$; $\text{bound\_head}^a_L = 0$; $\text{bound\_head}^a_R = 0$; $\text{bound\_arg}^b_L = 0$; $\text{bound\_arg}^b_R = 0$; $\text{bound\_head}^b_L = 1$; $\text{bound\_head}^b_R = 0$

$\lambda_a \leq \text{bound\_arg}^a_R$? *true* ; $\rho_a \leq \text{bound\_arg}^a_L$? *true*; $-\lambda_a \leq \text{bound\_head}^a_L$? *true*; $-\rho_a \leq \text{bound\_head}^a_R$? *true*; $\lambda_b \leq \text{bound\_arg}^b_R$? *true*; $\rho_b \leq \text{bound\_arg}^b_L$? *true* ; $-\lambda_b \leq \text{bound\_head}^b_L$? *true*; $-\rho_b \leq \text{bound\_head}^b_R$? *false*

▲ hypothesis 1 rejected by (9)

(12) hypothesis 2: L_1 *[&]* $R_2 \rightarrow s$ (L_1 and R_2 are conjoinable)

$\lambda_a = 2$; $\rho_a = 0$; $\lambda_b = -2$; $\rho_b = -1$; $\text{bound\_arg}^a_L = 0$; $\text{bound\_arg}^a_R = 1$; $\text{bound\_head}^a_L = 0$; $\text{bound\_head}^a_R = 0$; $\text{bound\_arg}^b_L = 0$; $\text{bound\_arg}^b_R = 0$; $\text{bound\_head}^b_L = 1$; $\text{bound\_head}^b_R = 0$

$\lambda_a \leq \text{bound\_arg}^a_R$? *false*

▲ hypothesis 2 rejected by (9)

(13) hypothesis 3: L_2 *[&]* $R_1 \rightarrow s$ (L_2 and R_1 are conjoinable)

$\lambda_a = 1$; $\rho_a = 0$; $\lambda_b = -1$; $\rho_b = 0$; $\text{bound\_arg}^a_L = 0$; $\text{bound\_arg}^a_R = 1$; $\text{bound\_head}^a_L = 0$; $\text{bound\_head}^a_R = 0$; $\text{bound\_arg}^b_L = 0$; $\text{bound\_arg}^b_R = 0$; $\text{bound\_head}^b_L = 1$; $\text{bound\_head}^b_R = 0$

$\lambda_a \leq \text{bound\_arg}^a_R$? *true*; $\rho_a \leq \text{bound\_arg}^a_L$? *true* ; $-\lambda_a \leq \text{bound\_head}^a_L$? *true*; $-\rho_a \leq \text{bound\_head}^a_R$? *true*; $\lambda_b \leq \text{bound\_arg}^b_R$? *true*; $\rho_b \leq \text{bound\_arg}^b_L$? *true* ; $-\lambda_b \leq \text{bound\_head}^b_L$? *true*; $-\rho_b \leq \text{bound\_head}^a_R$? *true*

▲ hypothesis 3 can not be rejected by (9).

(14) hypothesis 4: L_2 *[&]* $R_2 \rightarrow s$ (L_2 and R_2 are conjoinable)

$\lambda_a = 2$; $\rho_a = 0$; $\lambda_b = -2$; $\rho_b = 0$; $\text{bound\_arg}^a_L = 0$; $\text{bound\_arg}^a_R = 1$; $\text{bound\_head}^a_L = 0$; $\text{bound\_head}^a_R = 0$; $\text{bound\_arg}^b_L = 0$; $\text{bound\_arg}^b_R = 0$; $\text{bound\_head}^b_L = 1$; $\text{bound\_head}^b_R = 0$

$\lambda_a \leq \text{bound\_arg}^a_R$? *false*

▲ hypothesis 4 is rejected by (9).

Only one of the four possible hypotheses concerning the derivability of the sentence underlying the assignments in LL × RR is submitted to the proper parsing procedure. One can easily check that this sequent, *a b a\b [&] a\b s\a\a* → *s*, is the only one that induces a proper coordination. The type *a\b* is coordinated. If one occurrence of this type and the inert coordinator type *&* are neglected, the resulting sequent *a b a\b s\a\a* → *s* complies with the general Count Invariance (2) and will turn out to be derivable under standard assumptions of categorial grammar. The other three hypotheses can be rejected on the basis of the checklist defined in the consequent of (8); these checks are performed in constant time, not dependent on the length of the sentence cq. the sequents of types. This rejection is correct. None of the hypothesis checked in (11), (12) and (14) looks fit for derivability. In particular, they do not appear to contain two coordinates in such a fashion that the non-coordinated context with one of the coordinates added makes sense as a derivable sequent. For example, the hypothesis tested in (12), *a b/b a\b [&] a\b s\a\a* → *s*, may seem to contain a coordination *a\b [&] a\b*, but the 'de-coordinated' sequent *a b/b a\b s\a\a* → *s* from which the coordination is removed, does not even comply with (2) and is thus inderivable. Therefore, the check in (12) correctly rejects the hypothesis.

4 Effectiveness of Conjoinability

Conjoinability (9) has been implemented in a categorial parsing system called Delilah, which embodies among other things the algorithm for parsing unbounded coordination of [Cremers 1993]. In this system the selection of conjoinable pairs of strings is completely deterministic. During the construction of an assignment, a register is maintained and associated with it. If the assignment passes some other occurrence tests, it is admitted to LL or RR; its register is fixed. The procedure for comparing assignments in the product LL × RR checks pairs of quadruples from the registers in the spirit of the contraposition to the Coordinative Count Invariant (8). This requires only a fixed number of steps per pair of quadruples and per pair of registers. Furthermore, the check based on the Coordinative Count Invariant (8) is applied in addition to and in accordance with occurrence checks living on the more general Count Invariance (2); for these checks, see [Cremers 1989]. Here is an example of the effect of conjoinability in Delilah.

Sentence (15) contains 51 words from a restricted but highly polymorphic lexicon of Dutch. In this lexicon, all kinds of combinatorial options for each word are spelled out as categorial types. Because of this lexical polymorphism, the number of possible different assignments of lexical categories to the sentence is 5.3e16. The coordinator *en* is handled syncategorematically.

(15) Omdat ik niet had willen zeggen dat ik door de man met
Because I not had want say that I by the man with
de auto werd gedwongen te proberen Henk met de pop **en**
the car was forced to try Henk with the doll and
Agnes met een boek te laten spelen, werd de man door de
Agnes with a book to let play, was the man by the
vrouw gedwongen te zeggen dat hij mij niet met de pop
woman forced to tell that he me not with the doll
wilde proberen te laten spelen.
wanted try to let play.

First, some general occurrence checks that are not related to coordination keep the number of 5.6e16 possibilities just virtual by dynamically reducing the set of viable assignments of categories to this string to 2.9e6, or 6e–9 % of the search space. This number is the product of 136 assignments to the left of the coordinator - the set LL - and 21576 assignments to the right of the coordinator - the set RR. Then, applying Conjoinability to LL × RR leaves 652 combinations of a left and a right assignment as viable, i.e. 0,02 % of |LL × RR|. Only these 652 assignments are analyzed by the parser, which in this case finds a derivation for exactly one of them.
By the joined forces of independent occurrence checks and Conjoinability the number of assignments admitted to full parsing, is thus reduced to 1.2e–12 % of the original 5.3e16.

It is very difficult, if not impossible, to find a general metric for the effect of Conjoinability (9) on the set of admitted sequences: its effect - the ratio of pairs of left and right assignments that are rejected - is intrinsically dependent on type instantiations. Therefore, we can only give some more data to illustrate its effect. Table (16) contains, for some grammatical coordinated sentences over the same lexicon as was used for sentence (15), their length in words L, the number of possible assignments PA ($= \prod_1^n |A(w_i)|$), the product CP = |LL × RR|, the ratio CP/PA as a percentage, the number AA of assignments admitted to parsing, and the ratio AA/CP as a percentage. The latter ratio measures the effectiveness of applying Conjoinability. The ratio AA/PA gives the percentage of the total space of

possibilities that is transmitted to the parsing module; it stands for the effectiveness of the count preselection as a whole.
A comparable overview for some ungrammatical sentences is given in (17). The zero values mean that the system, in a stage prior to proper parsing, could not find any potentially derivable sequence; in the case of ungrammatical sentences this is, of course, a desirable result.

(16) *Pruning search space for grammatical coordinated sentences*

L	PA	CP	CP/PA %	AA	AA/CP %	AA/PA %
16	6.0e3	2e1	3.3e-1	2	1.0e1	3.3e−2
22	7.7e5	9.6e1	1.2e-2	4	4.2e0	5.2e−4
33	5.0e7	1.2e3	2.3e-3	8	6.9e−1	1.6e−5
39	7.4e7	5.4e3	7.3e-3	72	1.3e0	9.7e−5
44	4.4e8	2.2e4	5.1e-3	2	8.9e−3	4.5e−7

(17) *Pruning search space for ungrammatical coordinated sentences*

L	PA	CP	CP/PA %	AA	AA/CP %	AA/PA %
15	2.0e3	2.4e1	1.2e0	0	0	0
22	9.6e2	2.2e2	2.3e1	4	1.9e0	4.2e−1
33	1.5e6	2.1e3	1.4e−1	0	0	0
38	3.7e7	2.7e3	7.3e−3	24	8.8e−1	6.5e−5
47	2.6e8	6.7e4	2.6e−2	242	3.6e−1	9.2e−5

L: sentence length in words, including the coordinator. *PA*: the number of possible different assignments of lexical categories to the sentence of length L. *CP*: the cardinality of the cartesian product over the set of pre-checked assignments to the left hand side of the coordinator and the set of pre-checked assignments to the right of the coordinator, previously referred to as LL × RR. *AA*: the number of sequents admitted to the proper parsing procedure

From these figures one can conclude that the fraction AA/PA, which measures the effectiveness of the complete battery of count checks prior to parsing, including Conjoinability (9), tends to *decrease* as the search space PA, i.e. the number of lexically possible assignments, *increases*. This is also the tendency of AA/CP in figure (16): as CP increases - with PA - this ratio gets smaller, though not monotonically. The flaws in the tendency may be due to the ratio's being dependent on each and every detail in the type string. Nevertheless, we feel the tendency suggests that applying Conjoinability (9) is not only effective but also efficient: the time needed to bring about the reduction of CP to AA is only linearly dependent on CP, and the time for checking (8) and (9) at an individual sequent - demonstrated in (11) to (14) - is constant and given with the fixed number of basic types in a grammar.

5 Inevitability of Inequalities in Coordinative Count

In many cases, Conjoinability admits more assignments to the parser module than is necessary or desirable from a parsing point of view. The remaining redundancy is due to the fact that the Coordinative Count Invariant (8) is stated in terms of inequalities, rather than in terms of equalities. Count invariants in terms of inequalities, however, are the best we can get for a pruning instrument prior to parsing in the presence of coordination. To see why, consider the family of conjunctions

(18) Henk zei dat ik Agnes een boek **en**
Henk said that I Agnes a book and
(a) een tijdschrift had gegeven
a magazine had given
(b) de vrouw een tijdschrift had gegeven
the woman a magazine had given
(c) jij de vrouw een tijdschrift had gegeven
you the woman a magazine had given

Each of the right hand sides (18)a - c is a legitimate continuation of the given left hand side. Now take some assignment L to the left environment of the coordinator, i.e. to *Henk zei dat ik Agnes een boek.* From L alone one cannot make any predictions as to the nature of assignments R that are conjoinable. The only grammaticality condition imposed by L on R is that some proper prefix of R should reflect some proper suffix of L. Because coordination does not express itself functionally to the left nor to the right of the coordinator, we cannot tell which suffixes of L are available. In general, then, there will be more than one conceivable way of grammatically extending the string to the right of the coordinator. Consequently, several essentially different sequences R for (18)a - c have to be conjoinable with L. Nothing in L or its register imposes *a priori* occurrence conditions on assignments to possible extensions of the string to the right of a coordinator. The fact is that an assignment L with a fixed register reg_L may be conjoinable with many different Rs. Since these Rs all bear different registers, it is impossible to derive an nontrivial equality-condition on reg_R from reg_L - nor can it be done the other way around. One would have to look for a two-place function f such that $f(l,r)$ is constant for some l while r varies: these functions will hardly be dependent on r in an interesting way. Thus, the Coordinative Count Invariant defines the margins for the candidate Rs as narrowly as possible, but has to leave room for variation caused by the functional indeterminacy of coordination.

6 Discussion

The above result does not offer a general method of reducing parsing complexity in the presence of coordination. Rather it shows that in parsing certain types of lexicon-driven categorial grammars the harmful mix of natural language's intrinsic ambiguity and the intrinsic indeterminacy of coordination can be tamed. Although every grammar will confront its parsers with this mix in one way or another, it is by no means clear that there is a general strategy to handle this source of computational complexity. This paper argues that there is an approach for certain categorial grammars, namely those for which the parameters of the Coordinative Count Invariant (8) can be set in a meaningful way. These grammars are certainly not structurally complete in the sense of [Buszkowski 1988], but may have mildly context-sensitive capacity [Joshi *et al.* 1991]. To the categorial grammars in the scope of the present approach belong at least those dubbed 'parenthesis-free' in [Friedman *et al.* 1986]. For other

(categorial) grammars, there may be no, or a completely different, solution to the explosion of search space in the presence of coordination.
Moreover, up to now the Coordinative Count Invariant (8) is only consolidated for sentences containing a single coordinator. Handling multiple coordination in the same spirit will be even more tedious, but there is no reason to believe that (8) could not be generalized. The discussion in 4 suggests that the set of conditions resulting from generalizing (8) will be weaker than the present set of inequalities.

References

[Van Benthem 1986] Van Benthem, J. (1986), *Essays in Logical Semantics*, Reidel Pub Cy

[Van Benthem 1991] Van Benthem, J. (1991), *Language in Action*, SLFM 130, North-Holland

[Buszkowski 1988] Buszkowski, W. (1988), 'Generative Power of Categorial Grammars', in R. Oehrle, E. Bach and D. Wheeler (eds), *Categorial Grammars and Natural Language Structures*, Reidel, p. 69-94

[Cremers 1989] Cremers, C. (1989), 'Over een lineaire kategoriale ontleder', *TABU 19:2*, p. 76-86

[Cremers 1993] Cremers, C. (1993), *On Parsing Coordination Categorially*, HIL diss 5, Leiden University, also available as http://fonetiek-4.leidenuniv.nl/pub/cremers/dissi.ps

[Emms 1993] Emms, M. (1993), 'Parsing with Polymorphism', *Proceedings Sixth Conference of the European Chapter of the ACL*, ACL, p. 120 - 129

[Grootveld 1994] Grootveld, M. (1994), *Parsing Coordination Generatively*, HIL diss 7, Leiden University

[Friedman *et al.* 1986] Friedman, J., D. Dai, W. Wang (1986), 'The weak generative capacity of parenthesis-free categorial grammars', *Proceedings of Coling 86*, ACL, pp. 199 - 201

[Joshi *et al.* 1991] Joshi, A., K. Vijai-Shanker and D. Weir (1991), 'The Convergence of Mildly Context-Sensitive Grammar Formalisms', in P. Sells, S. Shieber and T. Wasow (eds), *Foundational Issues in Natural Language Processing*, MIT Press, p. 31-82

[Moortgat 1988] Moortgat, M. (1988), *Categorial Investigations*, GRASS 9, Foris

[Roorda 1992] Roorda, D. (1992), 'Proof Nets for Lambek calculus', *in*: A. Lecomte (ed), *Word Order in Categorial Grammar*, Ed. Adosa, p. 149-171

[Sag *et al.* 1985] Sag, I.A., G. Gazdar, T. Wasow and S. Weichler (1985), 'Coordination and How to Distinguish Categories', *Natural Language and Linguistic Theory 3:2*, p. 117-171

[Steedman 1990] Steedman, M. (1990), 'Gapping as Constituent Coordination', *Linguistics and Philosophy 13:2*, p. 207-263

[Wittenburg 1986] Wittenburg, K.B. (1986), *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification Based Formalism*, Ph.D. diss., University of Texas at Austin

BILEXICAL GRAMMARS AND A CUBIC-TIME PROBABILISTIC PARSER*

Jason Eisner

Dept. of Computer and Information Science, Univ. of Pennsylvania
200 South 33rd St., Philadelphia, PA 19103-6389 USA

Email: jeisner@linc.cis.upenn.edu

1 Introduction

Computational linguistics has a long tradition of *lexicalized* grammars, in which each grammatical rule is specialized for some individual word. The earliest lexicalized rules were word-specific subcategorization frames. It is now common to find fully lexicalized versions of many grammatical formalisms, such as context-free and tree-adjoining grammars [Schabes *et al.* 1988]. Other formalisms, such as dependency grammar [Mel'čuk 1988] and head-driven phrase-structure grammar [Pollard & Sag 1994], are explicitly lexical from the start.

Lexicalized grammars have two well-known advantages. Where syntactic acceptability is sensitive to the quirks of individual words, lexicalized rules are necessary for linguistic description. Lexicalized rules are also computationally cheap for parsing written text: a parser may ignore those rules that do not mention any input words.

More recently, a third advantage of lexicalized grammars has emerged. Even when syntactic *acceptability* is not sensitive to the particular words chosen, syntactic *distribution* may be [Resnik 1993]. Certain words may be able but highly unlikely to modify certain other words. Such facts can be captured by a *probabilistic* lexicalized grammar, where they may be used to resolve ambiguity in favor of the most probable analysis, and also to speed parsing by avoiding ("pruning") unlikely search paths. Accuracy and efficiency can therefore both benefit.

Recent work along these lines includes [Charniak 1995, Collins 1996, Eisner 1996b, Collins 1997], who reported state-of-the-art parsing accuracy. Related models are proposed without evaluation in [Lafferty *et al.* 1992, Alshawi 1996].

This recent flurry of probabilistic lexicalized parsers has focused on what one might call **bilexical grammars**, in which each grammatical rule is specialized for not one but *two* individual words.[1] The central insight is that specific words subcategorize to some degree for other specific words: *tax* is a good object for the verb *raise*. Accordingly, these models estimate, for example, the probability that (a phrase headed by) word y modifies word x, for any two words x, y in the vocabulary V.

At first blush, probabilistic bilexical grammars appear to carry a substantial computational penalty. Chart parsers derived directly from CKY or Earley's algorithm take time $O(n^3 \min(n, |V|)^2)$, which amounts to $O(n^5)$ in practice. Such algorithms implicitly or explicitly regard the grammar as a context-free grammar in which a noun phrase headed by *tiger* bears the special nonterminal $\mathrm{NP}_{\mathrm{tiger}}$. Such $\approx O(n^5)$ algorithms are explicitly used by [Alshawi 1996, Collins 1996], and almost certainly by [Charniak 1995] as well.

The present paper formalizes an inclusive notion of bilexical grammars, and shows that they can be parsed in time only $O(n^3 g^3 t^2 m) \approx O(n^3)$, where g, t, and m are bounded by the grammar and are typically small. (g is the maximum number of senses per input word, t measures the degree of lexical interdependence that the grammar allows *among* the several children of a word, and m bounds the number of modifier relations that the parser need distinguish for a given pair of words.) The new algorithm also reduces space requirements to $O(n^2 g^2 t) \approx O(n^2)$, from the $\approx O(n^3)$ required by CKY-style approaches to bilexical grammar. The parsing algorithm finds the highest-scoring analysis or analyses generated by the grammar, under a probabilistic or other measure. Non-probabilistic grammars may be treated as a special case.

*I am grateful to Mike Collins and Joshua Goodman for useful discussions of this work.

[1] Actually, [Lafferty *et al.* 1992] is formulated as a *trilexical* model, though the influence of the third word could be ignored.

The new $\approx O(n^3)$-time algorithm has been implemented, and was used in the experimental work of [Eisner 1996a, Eisner 1996b], which compared various bilexical probability models. The algorithm also applies to the head-automaton models of [Alshawi 1996] and the phrase-structure models of [Collins 1996, Collins 1997], allowing $\approx O(n^3)$-time rather than $\approx O(n^5)$-time parsing, granted the (linguistically sensible) restrictions that the number of distinct X-bar levels is bounded and that left and right adjuncts are independent of each other.

2 Formal Definition of Bilexical Grammars

2.1 Unweighted Bilexical Grammars

A bilexical grammar consists of the following elements:

- A set V of words, called the **vocabulary**, which contains a distinguished symbol ROOT.

  The elements of V may be used to represent word senses: so V can contain separate elements $bank_1, bank_2$, and $bank_3$ to represent the various meanings of *bank*. For parsing to be efficient, the maximum number of senses per word, g, should be small.

- A set M of one or more **modifier roles**.

  M does not affect the set of sentences generated by the grammar, but it affects the structures assigned to them. In these structures, elements of M will be used to annotate syntactic or semantic relations among words. For example, *John* might not merely modify *loves*, but modify it as SUBJECT or OBJECT, or as AGENT or PATIENT; these are roles in M.

  For parsing to be efficient, M should be small. (While a semantic theory may posit a great many modifier roles, it need not be the parser's job to make the finer distinctions.) More precisely, what should be small is m, the maximum number of modifier roles available for connecting two *given* words, such as *John* and *loves*.

- For each word w, a pair of deterministic finite-state automata ℓ_w and r_w. Each automaton accepts some set of strings over the alphabet $V \times M$.

  ℓ_w specifies the possible sequences of left dependents (arguments and adjuncts) for w. r_w specifies the possible sequences of right dependents for w. By convention, the first element in such a sequence is closest to w in the surface string. Thus, the possible dependent sequences are specified by $\mathcal{L}(\ell_w)^R$ and $\mathcal{L}(r_w)$ respectively.

  Note that the collection of automata in a grammar may be implemented as a pair of functions ℓ and r, such that $\ell(w, s, w', \mu)$ returns the destination state of the (w', μ)-labeled arc that leaves state s of automaton ℓ_w, and similarly for $r(w, s, w', \mu)$. These functions may be computed in any convenient way.

  For parsing to be efficient, the maximum number of states per automaton, t, should be small. However, without penalty there may be arbitrarily many distinct automata (one per word in V), and each automaton state may have arbitrarily many *arcs* (one per possible dependent in V), so $|V|$ does not affect performance.

We must now define the language generated by the grammar, and the structures that the grammar assigns to sentences of this language.

Let a **dependency tree** be a rooted tree whose nodes (internal and external) are labeled with words from V, and whose edges are labeled with modifier roles from M. The children ("dependents") of a node are ordered with respect to each other and the node itself, so that the node has both **left children** that precede it and **right children** that follow it. Figure 1a illustrates a dependency tree all of whose edges are labeled with the empty string.

Given a bilexical grammar, a leaf x of a dependency tree may be **expanded** to modify the tree as follows. Let w be the word that labels x, α be any string of left dependents accepted by ℓ_w, and β be any string of right dependents accepted by r_w. These strings are in $(V \times M)^*$. Add $|\alpha|$ left children and $|\beta|$ right children to the leaf x. Label the left children and their attached edges with the symbol pairs in α (from right to left); similarly label the right children and their attached edges with the symbol pairs in β (from left to right).

A **dependency parse tree** is a dependency tree generated from the grammar by starting with a single node, labeled with the special symbol ROOT $\in V$, and repeatedly expanding leaves until every node has been expanded exactly once. Figure 1a may be so obtained: one typical expansion step gives the leaf *plan* the child sequences $\alpha = the\epsilon$, $\beta = of, raise$. Another such step expands *the*, choosing to give it no children at all ($\alpha = \beta = \epsilon$).

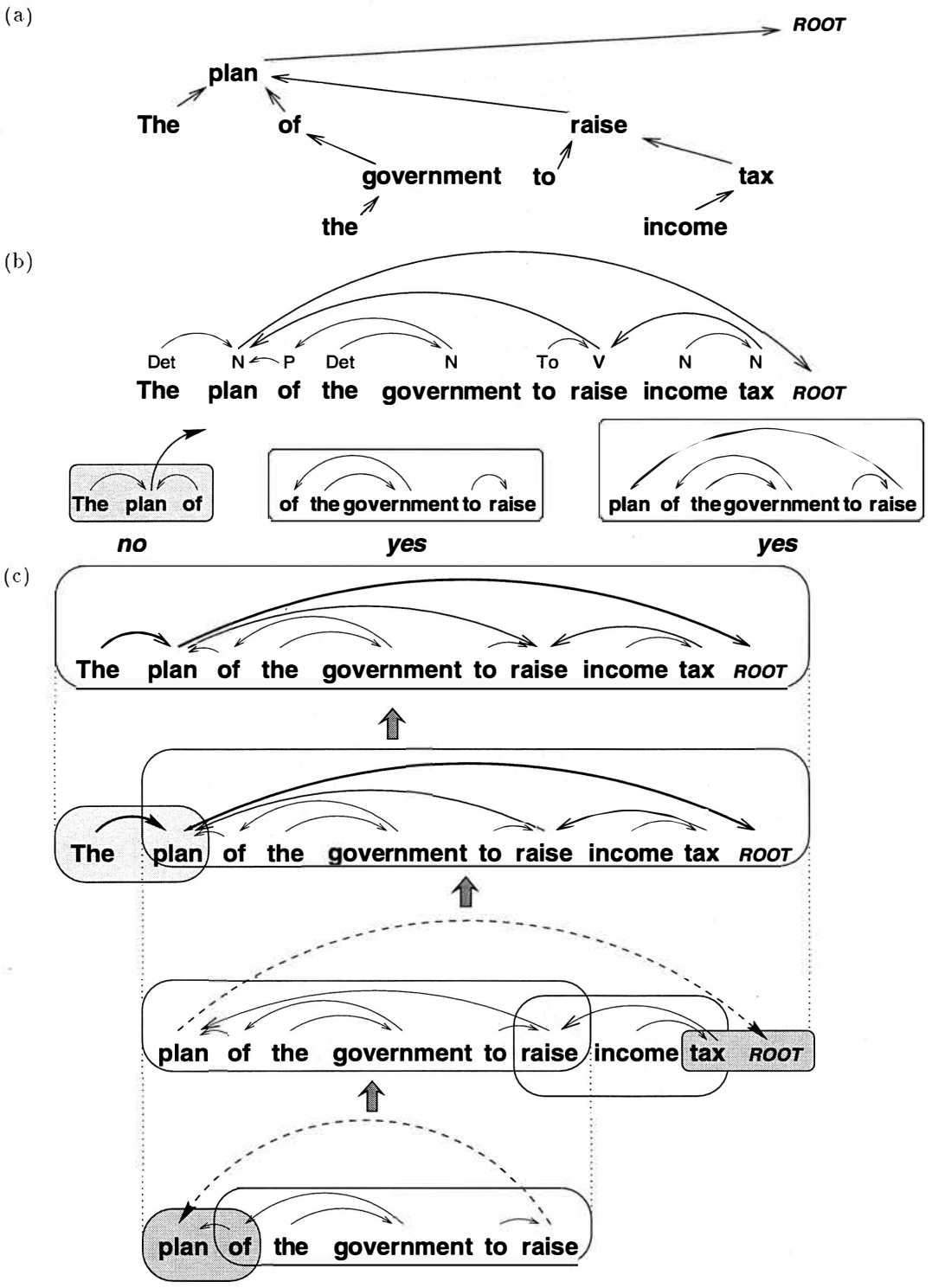


Figure 1: (a) An unlabeled dependency parse tree. (b) The same tree shown flattened out. A span of the tree is any substring such that no interior word of the span links to any word outside the span. One non-span and two spans are shown. (c) A span may be decomposed into smaller spans as repeatedly shown; therefore, a span can be built from smaller spans by following the arrows upward. The parsing algorithm builds successively larger spans in a dynamic programming table (chart). The minimal spans, used to seed the chart, are linked or unlinked word bigrams, such as *The*—*plan* or *tax* ROOT.

A string $\omega \in V^*$ is generated by the grammar, with analysis T, if T is a dependency parse tree and listing the node labels of T in infix order yields the string ω followed by ROOT. ω is called the **fringe** of T.

The term *bilexical* refers to the fact that (i) each $w \in V$ may specify a wholly different choice of automata ℓ_w and r_w, and furthermore (ii) each such automaton may make distinctions among individual words that are appropriate to serve as *children* of w. Thus the grammar is sensitive to specific *pairs* of lexical items. For example, it is possible for one lexical verb to select for a completely idiosyncratic set of nouns as subject, and another lexical verb to select for an entirely different set of nouns. Since it never requires more than a two-state automaton (though with many arcs!) to specify the set of possible subjects for a verb, there is no penalty for such behavior in the parsing algorithm to be described here.

2.2 Weighted Bilexical Grammars

The ability of a verb to subcategorize for an idiosyncratic set of nouns, as above, can be used to implement black-and-white ("hard") selectional restrictions. Where bilexical grammars are really useful, however, is in capturing *gradient* ("soft") selectional restrictions. A weighted bilexical grammar can equip each verb with an idiosyncratic *probability distribution* over possible object nouns, or indeed possible dependents of any sort. We now formalize this notion.

A **weighted finite-state automaton** A is a finite-state automaton that associates a real-valued **weight** with each arc and each final state. Following heavily-weighted arcs is intuitively "good," "probable," or "common"; so is stopping in a heavily-weighted final state. Each accepting path through A is automatically assigned a weight, namely, the sum of all arc weights on the path and the final-state weight of the last state on the path. Each string accepted by A is assigned the weight of its accepting path.

Now, we may define a weighted bilexical grammar as a bilexical grammar in which all the automata ℓ_w and r_w are weighted automata. The grammar assigns a weight to each dependency parse tree: namely, the sum of the weights of all strings of dependents, α and β, generated while expanding nodes to derive the tree. (This weight is well-defined: it does not depend on the order in which leaves were expanded.)

The goal of the parser is to determine whether a string $\omega \in V^*$ can be generated by the grammar, and if so, to determine the highest-weighted analysis for that sentence. It is convenient to set up the weighted automata so that each automaton formally accepts *all* strings in V^*, but assigns a weight of $-\infty$ to any that are not permitted by the competence grammar. Then a sentence is rejected as ungrammatical if its highest-weight analysis has weight only $-\infty$. The unweighted case is therefore a special case of the weighted case.

2.3 Lexical Selection

The above formalism must be extended to deal with lexical selection. Regrettably, the input to a parser is typically not a string in V^*. Rather, it contains ambiguous tokens such as *bank*, whereas the "words" in V are word senses such as $bank_1$, $bank_2$, and $bank_3$, or part-of-speech-tagged words such as *bank*/N and *bank*/V. One would like a parser to resolve these ambiguities as well as structural ambiguities.

We may modify the formalism as follows. Consider the unweighted case first. Let Ω be the real input—a string not in V^* but rather in $\mathcal{P}(V)^*$, where $\mathcal{P}$ denotes powerset. Thus the ith symbol of Ω is a **confusion set** of possibilities for the ith word of the input, e.g., $\{bank_1, bank_2, bank_3\}$. Ω is generated by the grammar, with analysis T, if some string $\omega \in V^*$ is so generated, where ω is formed by replacing each set in Ω with one of its elements. Note that ω is the fringe of T.

For the weighted case, each confusion set in the input string Ω assigns a weight to each of its members. Again, intuitively, the heavily-weighted members are the ones that are commonly correct, so the noun *bank*/N would be weighted more highly than the verb *bank*/V. We score dependency parse trees as before, except that now we also add to a tree's score the weights of all its fringe words, as selected from their respective confusion sets. Formally, we say that $\Omega = W_1 W_2 \ldots W_n \in \mathcal{P}(V)^*$ is generated by the grammar, with analysis T and weight $p + q_1 + \cdots + q_n$, if some string $\omega = w_1 w_2 \ldots w_n \in V^*$ is generated with analysis T and weight p, and $|\omega| = |\Omega| = n$ and for each $1 \leq i \leq n$, ω_i appears in the set W_i with weight q_i.

2.4 Adding String-Local Constraints

An extension is to allow the grammar to specify a list of **excluded bigrams**. If a tree's fringe contains an excluded bigram (two-word sequence), the tree is not considered a valid dependency parse tree, and is given

score $-\infty$.

§3.8 shows how this extension lets the scoring model consider such factors as the probability of the tag k-grams that appear in the parse (even for $k > 2$), as proposed by [Lafferty *et al.* 1992]. Consideration of such factors has been shown useful for probabilistic systems that simultaneously optimize tagging and parsing [Eisner 1996b].

3 Uses of Bilexical Grammars

Bilexical grammars, and the new parsing algorithm for them, are not limited to dependency-style structures. They are flexible enough to capture a variety of grammar formalisms and probability models. This section will illustrate the point with some key cases. We begin with the dependency case, and progress to phrase-structure grammars.

3.1 A Simple Case: Monolexical Dependency Grammar

It is straightforward to encode dependency grammars such as those of [Gaifman 1965]. We focus here on the case that [Milward 1994] calls Lexicalized Dependency Grammar (LDG). Milward demonstrates a parser for this case that requires $O(n^3g^3t^3) \approx O(n^3)$ time and $O(n^2g^2t^2) \approx O(n^2)$ space, using a left-to-right algorithm that maintains its state as an acyclic directed graph. Here t is taken to be the maximum number of dependents on a word. m does not appear because Milward takes tree edges to be unlabeled, i.e., $m = 1$.

LDG is defined to be only *monolexical*. Each word sense entry in the lexicon is for a word tagged with the type of phrase it projects. An entry for *helped*/S, which appears as head of the sentence *Nurses helped John wash*, may specify that it wants a left dependent sequence of the form w_1/N and a right dependent sequence of the form w_2/N, w_3/V. However, under LDG it cannot constrain the lexical content of w_1, w_2, or w_3, either discretely or probabilistically.[2]

By encoding a monolexical LDG as a bilexical grammar, and applying the algorithm described below in §4, we can improve parsing time and space by a factor of t. The encoding is straightforward. To capture the preferences for *helped*/S as above, we define $\ell_{helped/S}$ to be a two-state automaton that accepts exactly the set of nouns, and $r_{helped/S}$ to be a three-state automaton that accepts exactly those word sequences of the form (noun, verb). Obviously, $\ell_{helped/S}$ includes a great many arcs—one arc for every noun in V. This does not however affect parsing performance, which depends only on the number of *states* in the automaton.

3.2 Optional and Iterated Dependents

The use of automata makes the bilexical grammar considerably more flexible than its LDG equivalent. In the example of §3.1, $r_{helped/S}$ can be trivially modified so that the dependent verb is optional (*Nurses helped John*). LDG can accomplish this only by adding a new lexical sense of *helped*/S, increasing g.

Similarly, under a bilexical grammar, $\ell_{nurses/N}$ can be specified to accept dependent sequences of the form (adj, adj, adj, ... adj, (det)). Then *nurses* may be expanded into *weary Belgian nurses*. Unbounded iteration of this sort is not possible in LDG, where each word sense has a fixed number of dependents. In LDG, as in categorial grammars, *weary Belgian nurses* would have to be headed by the adjunct *weary*. Thus, even if LDG were sensitive to bilexicalized dependencies, it would not recognize *nurses*—*helped* as such a dependency.

3.3 Bilexical Dependency Grammar

In the example of §3.1, we may arbitrarily weight the individual noun arcs of the ℓ_{helped} automaton, according to how appropriate those nouns are as subjects of *helped*. (In the unweighted case, we might choose to rule out inanimate subjects altogether, by removing their arcs or assigning them the weight $-\infty$.) This turns the grammar from monolexical to bilexical, without affecting the cubic-time cost of the parsing algorithm of §4.

[2] What would happen if we tried to represent bilexical dependencies in such a grammar? In order to restrict w_2 to nouns denoting helpless animate creatures, the grammar would need a new nonterminal symbol, $\text{NP}_{\text{helpable}}$. All nouns in this class would then need additional lexical entries to indicate that they are possible heads of $\text{NP}_{\text{helpable}}$. The proliferation of such entries would drive g up to $|V|$ in Milward's algorithm, resulting in performance of $O(n^3|V|^3t^3)$ (or by ignoring rules that do not refer to lexical items in the input sentence, $O(n^6t^3) \approx O(n^6)$).

3.4 Probabilistic Bilexical Dependency Grammars

[Eisner 1996b] compares several probability models for dependency grammar. Each model simultaneously evaluates the part-of-speech tags and the dependencies in a given dependency parse tree. Given an untagged input sentence, the goal is to find the dependency parse tree with highest probability under the model.

Each of these models can be accomodated to the bilexical parsing framework, allowing a cubic-time solution. In each case, V is a set of part-of-speech-tagged words. For simplicity, M is taken to be a singleton set $\{\epsilon\}$. Each weighted automaton ℓ_w or r_w is defined so that it accepts any dependent sequence in V^*, but the automaton has 8 states, which enable interactions among successive dependents in a sequence. Any arc that accepts a noun (say) terminates in the Noun state. The w arc from Noun may be weighted differently than the w arcs from other states; so a given dependent word w may be more or less likely depending on whether the previous dependent in the sequence was a noun. The final-state weight of Noun may also be selected freely: so the sequence of dependents might be likely or unlikely to end with a noun.[3]

As sketched in [Eisner 1996a], each of Eisner's probability models is implemented as a particular scheme for weighting such an automaton. For example, model C regards ℓ_w and r_w as Markov processes, where each state specifies a probability distribution over its exit options, namely, its outgoing arcs and the option of halting. The weight of an arc or a final state is then the log of its probability. Thus if $r_{helped/V}$ includes an arc labeled with $(bathe/\mathrm{V}, \epsilon)$ and this arc is leaving the Noun state, then the arc weight is (an estimate of)

$$\log \Pr(\text{next right dependent is } bathe/\mathrm{V} \mid \text{parent is } helped/\mathrm{V} \text{ and previous right dependent was a noun})$$

The weight of a dependency parse tree under this probability model is a product of such factors, which means that it estimates Pr(dependency links & input words) according to a generative model. By contrast, model D estimates Pr(dependency links | input words), using arc weights that are roughly of the form

$$\log \Pr(bathe/\mathrm{V} \text{ is a right dep. of } helped/\mathrm{V} \mid \text{both words appear in sentence and prev. right dep. was a noun})$$

which is similar to the probability model of [Collins 1996]. Some of the models evaluated also rely on weighted string-local constraints, as implemented in §3.8 below.

3.5 Probabilistic Bilexical Phrase-Structure Grammar

In some situations, one wishes a parser to evaluate phrase-structure trees rather than dependency parse trees. [Collins 1997] observes that since VP and S are both verb-headed, the dependency grammars of §3.4 would falsely expect them to appear in the same environments. (The expectation is false because *continue* subcategorizes for VP only.) Phrase-structure trees address the problem by providing nonterminal labels. In addition, phrase-structure trees are less flat than dependency trees: a word's dependents attach to it at different levels, providing an obliqueness order on the dependents of a word. Obliqueness is of semantic interest, and is also exploited by [Wu 1995], whose statistical translation model preserves the topology (ID, not LP) of binary-branching parses.

Fortunately, it is possible to encode (headed) phrase structure trees as dependency trees with labeled edges. One can therefore use the fast bilexical parsing algorithm of §4 to generate the highest-weighted dependency tree, and then convert that tree to a phrase-structure tree.

For the encoding, we expand V so that all words are tagged not with single nonterminals but with **nonterminal chains**. Let us encode the phrase-structure tree *[John [continued$_V$ [to bathe$_V$ himself]$_{VP}$]$_{VP}$]$_S$*. The word *bathe* is the head of V and VP constituents, while *continued* is the head of V, VP, and S constituents (reading from the leaves upward). In the dependency tree, we therefore tag them as $bathe/V,VP$ and $continued/V,VP,S$. The automaton $r_{continued}$ can now require a dependent's tag to end with VP; this captures the ungrammaticality of **John continued [Oscar to bathe himself]*.

Next, we put M to be the set of nonterminals. To encode the fact that in the phrase-structure tree, *himself* modifies *bathe* by attaching at the VP level above *bathe*, we attach *himself* to *bathe* in the corresponding dependency tree with a VP-labeled edge.

Finally, we wish to ensure that any dependency tree the parser returns can be mapped back to a phrase-structure tree. For this reason, the bilexical grammar's automata must require that the left or right dependents of a word are appropriate to the word's tag. Thus, any edges depending from $continued/V,VP,S$ must be

[3] The eight states are START, Noun, Verb, Noun Modifier, Adverb, Preposition, Wh-word, and Punctuation.

labeled with V, VP, or S—the nonterminals that *continued* projects—and must fall in this order on each side. For instance, $r_{continued/V,VP,S}$ may not allow S-labeled right dependents to precede VP-labeled right dependents.

For this scheme to work, certain conditions must hold of the phrase-structure trees we are encoding. Only finitely many nonterminal chains may be available to tag a given word, and the nonterminals in a chain may not repeat. This is essentially in conformance with X-bar theory. The one difference is in the treatment of adjunction: in X-bar theory, three adjuncts to a VP would require 4 VP levels. One may work around this by stipulating a single VP-ADJUNCT level above VP, to which all VP adjuncts (0 or more) attach.[4]

This encoding scheme improves on that of [Collins 1996], because any dependency tree can be converted back to a phrase-structure tree, and this tree is unique. This makes it possible to use the fast bilexical parser, which (unlike Collins's) produces dependency trees without regard to whether they were derived from phrase structure trees. It also means that a probabilistic parsing model (unlike Collins's) need not be deficient, i.e., probability is not assigned to dependency structures that cannot be used by the phrase-structure grammar.

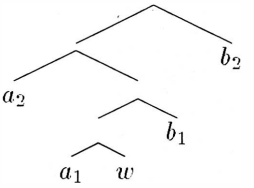
One interesting artifact of Collins's encoding scheme is that unary rules are impossible: every nonterminal level must have at least one dependent, whether on the left or on the right. If desired, the new scheme can be made to have this property as well. We augment V further, so that each nonterminal in a nonterminal chain is annotated with either "L" or "R," indicating that it requires children on the specified side. If VP is marked with "L," then the left-child automaton for that word must rule out any left-dependent sequence that does not have at least one dependent with a VP-labeled edge, and preserve the weights on all other dependent sequences.

3.6 Relationship to Head Automata

It should be noted that weighted bilexical grammars are essentially a special case of head-automaton grammars [Alshawi 1996]. As noted in the introduction, head-automaton grammars are bilexical in spirit. However, the left and right dependents of a word w are generated not separately, by automata ℓ_w and r_w, but in interleaved fashion by a single weighted automaton, d_w. d_w assigns weight to strings over the alphabet $V \times \{\text{—}, \text{—}\}$; each such string is an interleaving of lists of left and right dependents from V.

Head automata, as well as [Collins 1997], can model the case that §3.5 cannot: where nonterminal sequences may include nonterminal cycles. [Alshawi 1996] points out that this makes head automata are fairly powerful. An automaton corresponding to the regular expression $((a, \text{—})(b, \text{—}))^*$ requires its word to have an equal number of left and right children, i.e,. $a^n w b^n$. (By contrast, a bilexical grammar or dependency grammar may be made to generate $\{a^n w b^n : n \geq 0\}$ only by making words other than w the heads of these strings, so that the words that are allowed to interact bilexically would change.)

For syntactic description, this added power is probably unnecessary. (Linguistically plausible interactions among left and right subcat frames, such as fronting, can be captured in bilexical grammars simply via multiple word senses.) What head-automaton grammars offer over bilexical grammars is the ability for a head to specify an obliqueness order over all its dependents, including arbitrarily many adjuncts. A head-automaton parse tree for $a_2 a_1 w b_1 b_2$ is more finely detailed than in dependency grammar. It essentially gives the phrase headed by w a binary-branching analysis, such as



3.7 Idiom Encoding

To the bilexical construction of §3.3, one may add detectors for special phrases. Consider the idioms (a) *run scared*, (b) *run circles [around NP]*, and (c) *run NP [into the ground]*. (a), like most idioms, is only bilexical, so it may be captured "for free": simply increase the weight of the *scared* arc in $r_{run/V}$. But because (b) and (c) are trilexical, they require augmentation to the grammar. (b) requires a special state to be added to $r_{run/V}$, so that the dependent sequence (*circles*, *around*) may be recognized and weighted heavily. (c) requires a specialized lexical entry for *into*; this sense is a preferred dependent of *run* and has *ground* as a preferred dependent.

[4] This scheme conflates the two semantically distinct parses of "intentionally knock twice"; a later semantic component would have to resolve the scope ambiguity. The more powerful head automata of §3.6 could distinguish the readings, but at the cost of using a $O(n^5)$ rather than $O(n^3)$ parsing algorithm.

3.8 k-gram Tagging

One may use the string-local constraints of §2.4 to score dependency parse trees according to the trigram part-of-speech tagging model of [Church 1988]. Each input word w (including ROOT) is regarded as a confusion set over all tuples of the form (t'', t', t, w), where t is a tag for w and t'', t' are tags for the two words that precede w. (Thus, V consists of such tuples.) The weight of the tuple (t'', t', t, w) in its confusion set is $\log(\Pr(w \mid t) \cdot \Pr(t \mid t'', t'))$. The bigram $(t''_i, t'_i, t_i, w_i)(t''_{i+1}, t'_{i+1}, t_{i+1}, w_{i+1})$ is an excluded bigram unless $t'_i = t''_{i+1}$ and $t_i = t'_{i+1}$.

Because of the excluded bigrams, any dependency parse tree has a fringe (including ROOT) of the form

$$(\text{BOS}, \text{BOS}, t_1, w_1)(\text{BOS}, t_1, t_2, w_2)(t_1, t_2, t_3, w_3)\ldots(t_{n-1}, t_n, \text{EOS}, \text{ROOT})$$

Then the total weight accruing to the tree from the confusion sets is

$$\log(\Pr(t_1 \mid \text{BOS}, \text{BOS}) \cdot \Pr(t_2 \mid \text{BOS}, t_1) \cdot \Pr(t_3 \mid t_1, t_2) \cdots \Pr(\text{EOS} \mid t_{n-1}, t_n)$$

$$\cdot \Pr(w_1 \mid t_1) \cdots \Pr(w_n \mid t_n) \cdot \Pr(\text{ROOT} \mid \text{EOS}))$$

and to maximize this total is to maximize the probability product shown, just as [Church 1988] does. k-grams for $k > 3$ may be handled in exactly the same way.

Since the parser maximizes the above sum of confusion-set weights *and* the weights accruing from the choice of paths through the automata, grammatical structure also helps determine the highest-weighted parse.

4 Cubic-Time Parsing

This section begins by reviewing the general idea of chart parsing, presenting a general method drawn from context-free "dotted-rule" methods such as [Graham *et al.* 1980, Earley 1970]. Second, we will see why this method is inefficient when applied to bilexical grammars in the obvious way. Finally, a more efficient (cubic-time) algorithm is presented, which applies the general chart parsing method rather differently.

4.1 Generalized Chart Parsing Method

The input is a string $\Omega = W_1 W_2 \ldots W_n$ of confusion sets (so each $W_i \subseteq V$). C (the **chart**) is an $(n+1) \times (n+1)$ array. The chart **cell** $C_{i,j}$ holds a set of **partial analyses** of the input. Each partial analysis has a weight, and also a **signature** that concisely describes its ability to combine with other partial analyses. For each signature s, the **subcell** $C_{i,j}[s]$ holds the highest-weight partial analysis of the input substring $W_{i+1} W_{i+2} \ldots W_j$ for which s is the signature.[5]

Let Discover(i, j, P) be an operation that replaces $C_{i,j}[signature(P)]$ with the partial analysis P if P has higher weight than the partial analysis currently in $C_{i,j}[signature(P)]$.

A basic chart-parsing method is then as follows:

```
1.  for i := 1 to n + 1
2.      foreach w ∈ W_i, where W_{n+1} = {ROOT}      (* seed chart with members of the confusion set *)
3.          foreach partial analysis a of the single word w
4.              Discover(i − 1, i, a)
5.  for width := 1 to n + 1
6.      for start := 0 to (n + 1) − width
7.          end := start + width
```

[5] In a more general conception, $C_{i,j}[s]$ holds a summary of *all* known partial analyses of $W_{i+1} W_{i+2} \ldots W_j$ having signature s. Summaries must be defined in such a way that if f is an operation that combines two partial analyses, and A and B are sets of partial analyses of adjacent substrings, then $summary(\{f(a, b) : a \in A \text{ and } b \in B\})$ must be computable from $summary(A)$ and $summary(B)$. In addition, if A and B are both sets of partial analyses of $W_{i+1} W_{i+2} \ldots W_j$ having signature s, then $summary(A \cup B)$ must be computable from $summary(A)$ and $summary(B)$, so that new analyses can be added to the chart.

In the usual case, where the ultimate goal of the parser is to find the highest-weight dependency parse tree, $summary(A)$ is just the highest-weight parse tree in A (or, more generally, a forest of all parse trees in A that *tie* for the highest weight). However, if the parser is to return something else, one might set things up so that $summary(A)$ was a list of the 10 highest-weight parse trees in A—or even something more outré. If the parser is being used only to do language modeling for speech recognition, for instance, and the goal is to minimize the per-word error rate, then the summary might give a posterior probability distribution over the confusion set for the kth word, for each $i < k \leq j$; this would be determined by allowing the partial analyses in A to vote in proportion to their weight.

```
8.        for mid := start + 1 to end − 1
9.            foreach partial analysis a in C_{start,mid}      (* i.e., each C_{start,mid}[s] that is defined *)
10.               foreach partial analysis b in C_{mid,end}
11.                   foreach way of combining a and b into a new weighted analysis c (if any)
12.                       Discover(start, end, c)
13.   foreach partial analysis a in C_{0,n+1}
14.       if signature(a) indicates that a is a full (not partial) analysis of the sentence
15.           then print a
```

The iterations in lines 3 and 11 have yet to be defined, but the dynamic programming idea is clear (and familiar): analyses of length-1 substrings can be created from the substrings themselves (line 3), and analyses of successively longer substrings can be created by gluing together shorter analyses in pairs (line 11), until we have one or more analyses of the whole sentence.

In particular, the problem has the optimal substructure property: any *optimal* analysis of a long string can be found by gluing together just *optimal* analyses of shorter substrings. (An optimal analysis is defined to be a partial analysis of maximum weight for its signature; Discover() ensures that the chart contains only optimal analyses.) For suppose that a and a' are partial analyses of the same substring, and have the same signature, but a has less weight than a'. Then suboptimal a cannot be part of any optimal analysis b in the chart—for if it were, the definition of signature ensures that we could substitute a' for a in b to get an partial analysis b' of greater total weight than b and the same signature as b, which contradicts b's optimality.

The algorithm's running time is dominated by the six nested loops, yielding time $\Theta(n^3 S^2 d)$. Here S is the maximum number of possible signatures that may fall in a given chart cell, and d is the maximum number of ways to combine two adjacent partial analyses into a larger one.

4.2 Inefficient Chart Parsing of Bilexical Grammars

How might we apply the above method to parsing of bilexical grammars? The obvious way is for each partial analysis to represent a subtree. More precisely, each partial analysis would represent a kind of dotted subtree that may not yet have acquired all its children. The signature of such a dotted subtree is a triple (w, s_{ℓ_w}, s_{r_w}), where $w \in V$ is an input word, s_{ℓ_w} is a state of ℓ_w, and s_{r_w} is a state of r_w. If both s_{ℓ_w} and s_{r_w} are final states, then the signature is said to be *complete*.

It is clear that in line 3 of the algorithm, the sole analysis a is the triple (w, start state of ℓ_w, start state of r_w). It is also clear that line 14 should ensure that we print only trees with complete signatures as analyses of the sentence. Finally, consider line 11. If a has signature (w, s_{ℓ_w}, s_{r_w}) and b has a complete signature, then there are m possible values of c in which b is attached to the root of the dotted subtree a as a new right child. These analyses have signatures

$$\{(w, s_{\ell_w}, r(w, s_{r_w}, w', \mu)) : \mu \in M\} \qquad \text{(where } r(w, \ldots) \text{ is the transition function of } r_w \text{ as defined in §2.1)}$$

The case where a is attached to the root of b as a new left child is similar, and may give another m values for c.

Why is this method inefficient? Because there are too many possible signatures. The probability with which b attaches to a depends on the roots of both a and b. Since the root w of a could be any of the words at positions $start + 1, start + 2, \ldots mid$, and there may be $\min(n, |V|)$ distinct such words in the worst case, the number S of possible signatures for a is at least $\min(n, |V|)$. The same is true for b, whose root w' could likewise be any of many words. But then the runtime of the algorithm is $\Omega(n^3 \min(n, |V|)^2 |M|) \approx \Omega(n^5)$. In a nutshell, the problem is that each chart cell may have to maintain many differently-headed analyses.

4.3 Efficient Chart Parsing of Bilexical Grammars

To eliminate these two additional $\min(n, |V|)$ factors, we must reduce the number of possible signatures for a partial analysis. The solution is for partial analyses to represent some kind of contiguous string other than constituents. Each partial analysis in $C_{i,j}$ will be a new kind of object called a span, which consists of one or two "half-constituents" in a sense to be described. The headword(s) of a span in $C_{i,j}$ are *guaranteed* to be at positions i and j in the sentence. This guarantee means that where $C_{i,j}$ in the previous section had n-fold uncertainty about the correct location of the headword for the optimal analysis of $W_{i+1} W_{i+2} \ldots W_j$, here it will have only 3-fold uncertainty. The three possibilities are that w_i is an unattached headword, that w_j is, or that both are.

Given a dependency parse tree, we know what its constituents are: a constituent is any substring consisting of a word and all its descendants. The inefficient parsing algorithm of the §4.2 assembled the correct parse tree by finding and gluing together analyses of the tree's constituents in an approved way. For something similar to be possible with spans, we must define what the spans of a given dependency parse tree are, and how to glue analyses of spans together into analyses of larger spans. Not every substring of the sentence is a correct constituent, and in the same way, not every substring is a correct span.

Figure 1a–b illustrates what spans are. A span of the dependency parse tree in (a) and (b) is any substring $w_i w_{i+1} \ldots w_j$ $(j > i)$ of the tree's fringe, such that none of the interior words of the span communicate with any words outside the span. Formally: if $i < k < j$, and w_k is a dependent of $w_{k'}$ or vice-versa, then $i \leq k' \leq j$.

Since we will build the parse by assembling analyses of spans, and the interiors of adjacent spans are insulated from each other, we crucially never need to know anything about the internal analysis inside a span. When we combine two adjacent spans, we never add a link from or to the interior of either. For, by the definition of span, if such a link were necessary, then the spans being combined could not be spans of the true parse anyway. There is always some other way of decomposing the true parse (itself a span) into smaller spans so that no such links from or to interiors are necessary.

Figure 1c shows such a decomposition.[6] Any span of greater than two words, say again from w_i to w_j, can be decomposed uniquely by the following deterministic procedure. Choose $i < k < j$ such that w_k is the rightmost word (strictly inside the span) that connects to w_i; if there is no such word, put $k = i+1$. Because crossing links are not allowed, the substrings from $w_i \ldots w_k$ and $w_k \ldots w_j$ must also be spans. We can therefore assemble the original $w_i \ldots w_j$ span by concatenating the $w_i \ldots w_k$ and $w_k \ldots w_j$ spans, and optionally adding a link between the end words, w_i and w_j. By construction, there is never any need to add a link between any other pair of words. Notice that when the two narrower spans are concatenated, w_k gets its left children from one span and its right children from the other.

The procedure for choosing k can be rephrased declaratively. To wit, the left span in the concatenation, $w_i \ldots w_k$, must be **simple** in the following sense: it must have a direct link between w_i and w_k, or else have only two words.

It is useful to note that the analysis of a span always takes one of three forms; Figure 1b illustrates the first two (labeled "yes"). In the first case, the endwords w_i and w_j are not yet connected to each other: that is, the path between them in the final parse tree will involve words outside the span. Then the span consists of two "half-constituents"—w_i with all its right descendants, followed by w_j with all its left descendants. w_i and w_j both need parents. In the second case, w_j is a descendant of w_i via a chain of one or more leftward links within the span itself; then the span consists of w_i and all its right descendants to the left of w_j (inclusive), and only w_i still needs a parent. The third case is the mirror image of the second. (It is impossible for both w_i and w_j to both have parents inside the span: for then some word interior to the span would need a parent outside it.)

The signature of a span does not have to state anything about the internal analysis except which of these three cases holds—i.e., which of w_i, w_j need parents. This is needed so that the parser knows when it is able to add a link from i or j to a more distant word after concatenation, without creating multiple parents or otherwise jeopardizing the form of the dependency parse. To determine the allowability of the dependent introduced by such a new link, or the weight associated with it, the signature of a span from w_i to w_j must also include the states of the automata r_{w_i} and ℓ_{w_j}.

We can now understand the actual algorithm. It is convenient to slightly alter the definition of $C_{i,j}$, so that it stores the best analysis (as a span) of $W_i W_{i+1} \ldots W_j$.[7] We may represent an analysis of a span as a tuple $(linktype\epsilon, a, b)$, where $linktype\epsilon$ specifies the label $(\in M)$ and direction of the link, if any, between the leftmost and rightmost words of the span, and where a and b point to the narrower spans that were concatenated to obtain this one. If the span is only two words wide, then we represent an analysis of it as $(linktype\epsilon, w_1, w_2)$ so that the analysis specifies the words it has chosen from the confusion set.

As usual, the analyses we discover in a cell of the chart are organized into competitions or subcells by their signatures. The signature of an analysis has the form $(L?, R?, w_L, w_R, s_r, s_\ell, simple?)$. Here $L?$ and $R?$ are boolean variables stating whether the leftmost and rightmost words have parents in the span. w_L and w_R are the leftmost and rightmost words of the span (as chosen from the appropriate confusion sets). s_r is the state

[6] [Lafferty *et al.* 1992] give a related decomposition for the case of link grammar, and use it to construct an $O(n^3)$ top-down parsing algorithm. The bilexical parsing algorithm could be adapted to the case of link grammars, in which case it would resemble a bottom-up version of the independent algorithm of [Lafferty *et al.* 1992].

[7] Why not start with W_{i+1} as in §4.1? Because the spans we concatenate, unlike constituents that we concatenate, overlap in one word.

of the leftmost word's right-dependent automaton r_{w_L} after the automaton has read all the leftmost word's dependents within the span, and s_ℓ is similarly the state of the rightmost word's left-dependent automaton ℓ_{w_R}. Finally, *simple*ϵ? is true just if the span is simple, as defined above; we use this to prevent ourselves from finding the same analysis in multiple ways.

```
1.  for i := 1 to n
2.      foreach w_i ∈ W_i, w_{i+1} ∈ W_{i+1} such that w_i w_{i+1} is not an excluded bigram, where W_{n+1} = {ROOT}
3.          foreach linktypeε ∈ ({—, —} × M) ∪ {NONE}
4.              Discover(i, i+1, (linktypeε, w_i, w_{i+1}))      (* seed with two-word spans *)
5.  for width := 3 to n+1
6.      for start := 1 to (n+1) − width
7.          end := start + width
8.          for mid := start + 1 to end − 1
9.              foreach partial analysis a in C_{start,mid}      (* i.e., a is each C_{start,mid}[s] that is defined *)
10.                 if sig(a).simpleε?      (* where sig(a) is just s from comment above; don't recompute it *)
11.                     foreach partial analysis b in C_{mid,end} such that sig(b).w_L = sig(a).w_R
                                                    (* so a, b agree on sense of overlapping word *)
12.                         if (sig(a).R? xor sig(b).L?)      (* overlapping word gets exactly one parent *)
13.                             Discover(start, end, (NONE, a, b))      (* version without a covering link *)
14.                             if not (sig(a).L? or sig(b).R?)      (* prevents multiple parents, link cycles *)
15.                                 foreach linktypeε ∈ {—, —} × M
16.                                     Discover(start, end, (linktypeε, a, b))
17. foreach partial analysis a in C_{0,n+1}
18.     if sig(a).L?      (* so a is a connected tree rooted at ROOT *)
19.         then print a
```

Computing the signatures is fairly straightforward. For example, if *linktype*ϵ is rightward in line 16 (making w_{start} a dependent of w_{end}), then the new analysis (*linktype*ϵ, a, b) has the signature

$$(\text{TRUE}, \text{FALSE}, sig(a).w_L, sig(b).w_R, sig(a).s_r, \delta(sig(b).s_\ell, sig(a).w_L), \text{TRUE})$$

Computing weights of analyses is also fairly straightforward. In the above example, (*linktype*ϵ, a, b) has weight

$$weight(a) + weight(b) + arcweight(sig(b).s_\ell, sig(a).w_L) + stopweight(sig(a).s_\ell) + stopweight(sig(b).s_r)$$

Note the use of the final-state weights, *stopweight*, to reflect the fact that the overlapping word w_{mid} (see line 11) can no longer get new children once it is hidden in the interior of the span.[8] For a two-word span (*linktype*ϵ, w_i, w_{i+1}) with rightward link, as created in line 4, the weight is $arcweight(w_{i+1}.\text{START}, w_i)$ plus the weight of w_i (only!) in its confusion set.

The algorithm requires $O(n^2S)$ space for the chart, where S is the maximum number of signatures per cell. The running time is again dominated by the six nested loops. It is $O(n^3 \cdot S^2 \cdot |M|)$. Given the definition of signatures, $S = O(g^2t^2)$; that is, it is bounded by a constant times (max size of confusion set)^2 times (max states per automaton)^2. (Crucially, there are only g choices for each of w_L and w_R.) Thus, the grammar constant S is typically small.

With careful coding it is possible to improve the runtime somewhat, from $O(n^3g^4t^4|M|)$ to $O(n^3g^3t^2m)$. Perhaps only some link types are possible in line 15, so $|M|$ can be replaced by m. We can save a factor of g at line 11, because the restriction on $sig(b).w_L$ means that we only need to iterate through S/g of the signatures. Finally, we can reduce S to just $O(g^2t)$, which helps both time and space complexity. Instead of Discover() using a single chart to maintain the best analysis with signature (L?, R?, w_L, w_R, s_r, s_ℓ, *simple*ϵ?), it will use two charts to maintain, respectively, the best analyses with signatures (L?, R?, w_L, w_R, s_r,HALTED, *simple*ϵ?) and (L?, R?, w_L, w_R,HALTED, s_ℓ, *simple*ϵ?). Each of these two analyses has already had one stopweight added to its weight. The algorithm should now be modified to select a from the first chart and b from the second chart.

5 Conclusions

This paper has introduced a new formalism, weighted bilexical grammars, in which individual lexical items can have idiosyncratic selectional influences on each other. Such "bilexicalism" has been a theme of much current

[8] In the case where $start = 0$, also add $stopweight(sig(b).s_\ell)$; if $end = n+1$, also add $stopweight(sig(a).s_r)$.

work. The new formalism can be used to describe bilexical approaches to both dependency and phrase-structure grammars, and its scoring approach is compatible with a wide variety of probability models.

The obvious parsing algorithm for bilexical grammars (used by most authors) takes time $\Theta(n^5)$. A more efficient $O(n^3)$ method is exhibited. The new algorithm has been implemented and used in a large parsing experiment [Eisner 1996b].

References

[Alshawi 1996] Hiyan Alshawi. Head automata for speech translation. *Proceedings of the fourth International Conference on Spoken Language Processing*, Philadelphia. cmp-lg/9607006.

[Charniak 1995] Eugene Charniak. Parsing with context-free grammars and word statistics. Technical Report CS-95-28, Dept. of Computer Science, Brown Univ.

[Church 1988] Kenneth W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conf. on Applied NLP*, 136–148, Austin, TX.

[Collins 1996] Michael J. Collins. A new statistical parser based on bigram lexical dependencies. *Proceedings of the 34th Annual ACL*, Santa Cruz, July, 184–191. cmp-lg/9605012.

[Collins 1997] Michael J. Collins. Three generative, lexicalised models for statistical parsing. *Proceedings of the 35th ACL and 8th European ACL*, Madrid, July. cmp-lg/9706022.

[Earley 1970] Earley, J. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13(2): 94-102.

[Eisner 1996a] Jason M. Eisner. Three new probabilistic models for dependency parsing: an exploration. *Proceedings of COLING-96*, Copenhagen, August, 340–345. cmp-lg/9706003.

[Eisner 1996b] Jason M. Eisner. An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, IRCS, University of Pennsylvania. cmp-lg/9706004.

[Gaifman 1965] H. Gaifman. Dependency systems and phrase structure systems. *Information and Control* **8**, 304–337.

[Graham *et al.* 1980] Graham, S.L., Harrison, M.A. and Ruzzo, W.L. An Improved Context-Free Recognizer. *ACM Transactions on Prog. Languages and Systems* 2(3):415-463.

[Lafferty *et al.* 1992] John Lafferty, Daniel Sleator, and Davy Temperley. Grammatical trigrams: A probabilistic model of link grammar. In *Proc. of the AAAI Conf. on Probabilistic Approaches to Natural Language*, October.

[Mel'čuk1988] Igor A. Mel'čuk. *Dependency Syntax: Theory and Practice.* State University of New York Press.

[Milward 1994] David Milward. Dynamic dependency grammar. *Linguistics and Philosophy* **17**: 561–605. Netherlands: Kluwer.

[Pollard & Sag 1994] Carl Pollard & Ivan Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press.

[Resnik 1993] *Selection and Information: A Class-Based Approach to Lexical Relationships.* Ph.D. dissertation (Technical Report IRCS-93-42), University of Pennsylvania.

[Schabes *et al.* 1988] Yves Schabes, Anne Abeillé, & Aravind Joshi. Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. Proceedings of COLING-88, Budapest, August.

[Wu 1995] Dekai Wu. An algorithm for simultaneously bracketing parallel texts by aligning words. *Proceedings of ACL-95*, MIT.

AUTOMATON-BASED PARSING FOR LEXICALIZED GRAMMARS

Roger Evans
University of Brighton
Roger.Evans@itri.brighton.ac.uk

David Weir
University of Sussex
David.Weir@cogs.susx.ac.uk

Abstract

In wide-coverage lexicalized grammars many of the elementary structures have substructures in common. This means that during parsing some of the computation associated with different structures is duplicated. This paper explores ways in which the grammar can be precompiled into finite state automata so that some of this shared structure results in shared computation at run-time.

1 Introduction

This paper investigates grammar precompilation techniques aimed at improving the parsing performance of lexicalized grammars. In a wide-coverage lexicalized grammar, such as the XTAG grammar (XTAG-Group, 1995), many of the elementary structures[1] have substructures in common. If such structures are viewed as independent by a parsing algorithm, the computation associated with their shared structure may be duplicated. This paper explores ways in which the grammar can be precompiled so that some of this shared structure results in shared computation at run-time.

We assume as a starting point a conventional kind of parser for lexicalized grammars (such as Vijay-Shanker and Joshi (1985), Schabes (1990), Vijay-Shanker and Weir (1993) for LTAG), although the techniques described here are not restricted to the specific details of any particular parsing algorithm. Our approach is based on the observation that the process of traversing an elementary structure (ES) during parsing can be expressed as a finite state automaton (FSA), and hence that the whole parsing process can be viewed in terms of interacting FSA's[2]. By doing this, it becomes possible to apply standard FSA optimisation techniques (such as determinisation and minimisation) to sets of ES's, and to vary control strategy on a per-ES basis to achieve optimum results.

We shall begin by describing how an ES can be mapped onto an automaton, and how a conventional parsing algorithm can be adapted to work with such automata. Then, using a family of related ES's we shall give two examples of optimisation of the processing of ES's through a deterministic merging of their corresponding automata, and show how different control strategies yield different optimisations. We will illustrate the approach using Lexicalized Tree Adjoining Grammars (Joshi and Schabes, 1991), but we note that the techniques described here can also be applied to other lexicalized grammar formalisms[3].

2 Automaton-based parsing of LTAG

Generally speaking, parsing algorithms for LTAG have the property that a traversal is made through the nodes of elementary trees (the 'elementary structures' of an LTAG grammar). In bottom-up parsers, for example, this

[1] We use the term *elementary structures* to mean the basic components of grammatical description. In LTAG (Joshi and Schabes, 1991), for example, the elementary structures are initial and auxiliary trees. A grammar consists of a finite set of elementary structures that are built into derived structures using the composition operations of the formalism.

[2] Cf. Alshawi (1996) for a similar approach. but with different objectives and a different style of formalism. The approach we present here is also reminiscent of LR parsing in that the FSA used by an LR-parser to recognize viable prefixes can be viewed as the result of determinizing a nondeterministic finite state automaton constructed by linking with ϵ-transitions deterministic automata for the individual productions.

[3] The second author is currently involved in the development of a wide-coverage grammar and automaton-based parser using the formalism described in Rambow, Vijay-Shanker, and Weir (1995).

involves traversing the trees from the anchor of the tree up to its root as more and more of the input is spanned to the right and left of the anchor. Hence, the recognition of an elementary tree can be seen to involve a sequence of computation steps: eg, "find an NP to the right, then find a PP to the right and then find an NP to the left". The particular sequence of steps is determined by the tree structure and the labels attached to substitution (and adjunction) nodes. We shall call each of these steps an **elementary computation step** (or ECS) and the sequence corresponding to the processing of an entire elementary tree an **elementary computation**.

Consider, for example, the ditransitive tree given in Figure 1. A bottom-up parser will initially anchor this tree on an actual ditransitive verb in the input and start its traversal at the 'v' node. If a noun phrase is detected to the right of the verb, the first ECS will be to consume it and move to the 'np1' node. A preposition to the right of that will generate a second ECS, moving to the 'p' node, then an additional noun phrase will take it to the 'np2' node. Here the traversal continues to the 'pp' node and then the 'vp' node with no further inputs (and so no elementary computation steps), and so the final ECS consumes a noun phrase to the left, taking the parser to the 'np3' node, and from there straight to the 's' node.

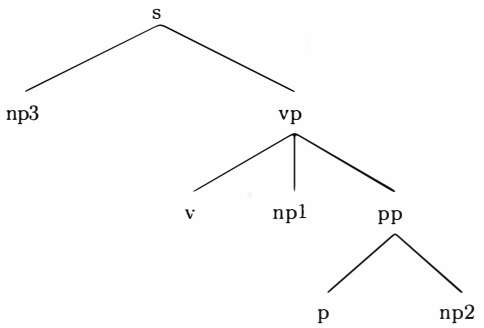


Figure 1: A ditransitive verb tree

This description of the processing clearly invites representation as a finite state automaton, and a possible automaton is given in Figure 2. This automaton has five states, with each transition between states corresponding to an ECS. The input language for the automaton consists of tokens representing parse table states that trigger the ECS ("NP to the left", notated as $\overleftarrow{\text{np}}$, for example). The additional loop attached to the fourth node allows for optional adjunctions at the 'vp' node (notated as $\overleftrightarrow{\text{vp}}$, and triggered by the presence of a complete auxiliary tree enveloping the completed 'vp' subtree). These introduce arbitrarily many additional ECS's without changing the state in the automaton, that is, without traversing the underlying elementary tree at all.

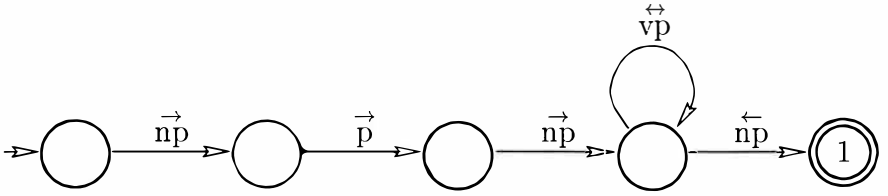


Figure 2: Finite state automaton for the ditransitive verb tree

Automata such as this can be 'executed' by a parser in a manner similar to a conventional tree traversal approach. The parser manipulates a table associating automaton states with segments of the input string (represented as quadruples of indices, or pairs of indices together with a stack of tree addresses – see Vijay-Shanker and Weir (1993)). The table is first populated with initial states for all the automata associated with trees anchored by the words in the input string. Thereafter, elementary computation steps are triggered by the presence of final states in the table. Final states are labelled with the tree they represent (such as the label '1' in Figure 2), and from this the parser can recover the information required to licence a new ECS – the root category of an initial tree (to licence a substitution) or the head and foot node categories of an auxiliary tree (to licence an adjunction). The parser looks for adjacent table entries encoding automata states that are waiting

for this ECS trigger (on the appropriate side). When such an adjacent entry is found, the ECS is executed by passing the trigger as input to the automaton, making a transition to a new state in the automaton, and then creating a new table entry for the new state spanning a suitably enlarged input segment.

The control strategy for identifying and executing ECS's is a detail of the parsing algorithm that need not concern us here. Parsing terminates when no further ECS's can be generated. At that point any table entries containing a final state of some automaton and spanning the entire input correspond to parses. Reading off a parse tree may involve further work, however. Notice that the automaton in Figure 2 does not directly encode all the structure of the tree, and could in fact correspond to a number of well-formed trees. But the tree label associated with its final state does provide the additional information we need to derive a tree from the parse table[4].

As presented so far, this transformation from trees to automata is straightforward and we will not dwell on the details of its application to any specific parsing algorithm here. Instead we will focus on the potential for exploiting the automaton-based representation more fully. For although the transformation is straightforward it is not completely trivial: there is a significant difference between the tree-based and the automaton-based parsers, and it is a difference in the distribution of *control* information. In the tree-based parser, the trees are passive data structures which are traversed in an order specified by the parsing algorithm. In the automaton-based parser, some of this control is vested in the automata themselves, rather than the parsing algorithm. The parser still controls the overall order of execution of ECS's, but the effect of an ECS on a particular automaton is determined by the automaton, not by the parsing algorithm.

There are two consequences of this that we will pursue in the rest of this paper:

- The parser can operate on any automata over the ECS input language – it does not depend on one particular mapping from trees to individual automata. This means we can use standard techniques for merging automata for several trees together into optimal combined representations, resulting in automata that implement several trees at once. This results in greater parsing efficiency, without having to change the parser or the grammar.

- Each automaton encodes a control strategy for traversing the corresponding tree (or set of trees). In the example above we chose one particular traversal, but others are also possible. There is no requirement that the same strategy be used in all the automata, since the parser just follows the automata specifications blindly. So we can consider different strategies giving different optimisations for different parts of the grammar.

3 Some Examples

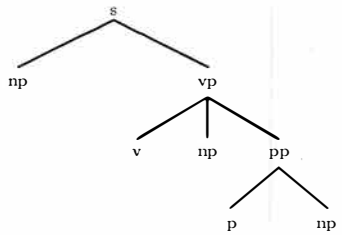
In this section we shall explore these ideas in more detail, by looking at a particular set of related trees, and some of the automata that can be derived from them. The trees we shall use are all potentially anchored by the same surface word so it is likely that they would all need to be considered together during a parse. We will describe automata corresponding to two different traversals of these trees, and show the different results that are obtained by determinising the union of all the automata in each case.

3.1 The Trees

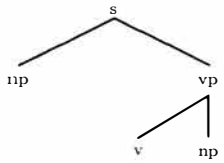
The set of trees we shall consider are a selection from the ditransitive verb family, including the basic ditransitive itself and some (but by no means all) of its regular syntactic and semantic alternations. One feature of these trees is that they can all be anchored by the same word (a present participle or gerund form, such as 'giving'), so that a parser may need to consider all of them simultaneously during parsing.

Tree 1 is the basic ditransitive form itself, which we take to include a noun phrase and a prepositional phrase. We assume here that 'v' is the lexical anchor and that adjunction can only occur at the 'vp' node. Note that although this tree is potentially complete for finite verb forms, our example sentence uses a present participle, which requires the adjunction of an auxiliary verb to form a valid sentence. In our example sentences, we shall indicate such additional words, not contained in the actual tree, but putting them in parentheses. Other adjunctions (of adverbial modifiers etc.) are also possible. Trees 2 and 3 are the regular transitive and intransitive alternants that any ditransitive verb has (where the missing complements are left unspecified).

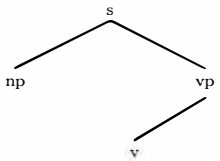
[4]For a simple recognizer, it is sufficient to label the final states with the category information the parser actually requires to identify ECS's – see also Section 3.2 below.



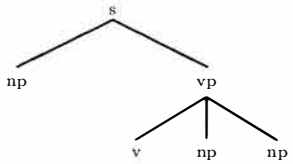
Tree 1: Ditransitive
Mary (is) giving presents to the children



Tree 2: Transitive
Mary (is) giving presents



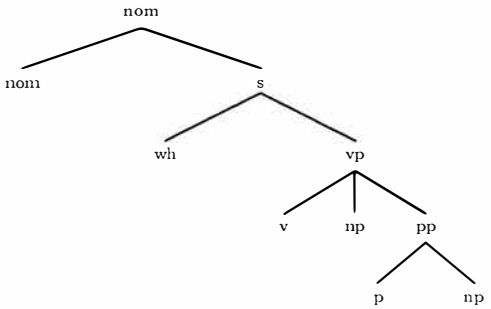
Tree 3: Intransitive
Mary (is) giving



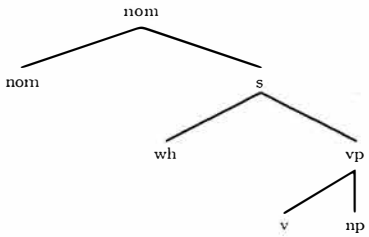
Tree 4: Dative
Mary (is) giving the children presents

Tree 4 assumes that we are in the specific subclass of ditransitives that can undergo dative movement, and provides the appropriate double-np subcategorisation for it.

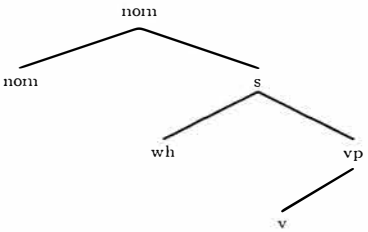
Trees 5–8 represent the wh-relative versions of Trees 1–4. Note that these are in fact auxiliary trees which can be adjoined into noun phrases. As well as having extra structure to support this, the subject position is marked as a 'wh' form. As before, adjunctions can take place at the 'vp' node.



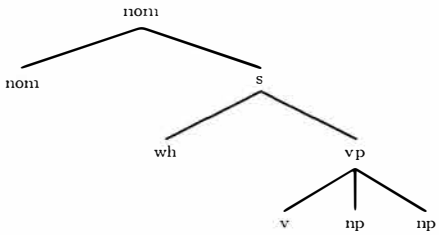
Tree 5: WH-ditransitive
(someone) who (is) giving presents to the children



Tree 6: WH-transitive
(someone) who (is) giving presents



Tree 7: WH-intransitive
(someone) who (is) giving

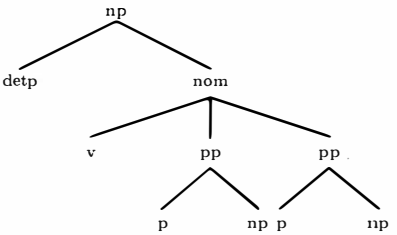


Tree 8: WH-dative
(someone) who (is) giving the children presents

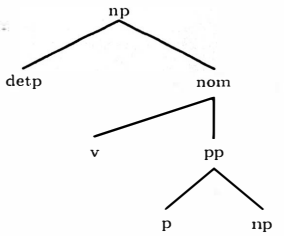
Trees 9–11 provide nominal variants of the ditransitive, transitive and intransitive forms, using a gerund. Here, the object noun phrase (when present) has become a prepositional phrase, and the trees represent a noun-phrase with a determiner rather than a sentence with a subject. Adjunction can only occur at the 'nom' node[5].

In all we have eleven trees, all anchored in the same word, and all clearly duplicating structure. There are many other trees in this family including those for topicalized sentences and wh-sentences. By mapping these

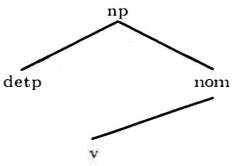
[5]This is not the only possible analysis of gerunds of course, but it is adequate for present illustrative purposes.



Tree 9: Ditransitive gerund
the giving of presents to the children



Tree 10: Transitive gerund
the giving of presents



Tree 11: Intransitive gerund
the giving

eleven into automata, we aim to collapse some of this structure using standard automata techniques, and so make parsing of this tree set more efficient. In general there is more than one way to do this, and the following two subsections illustrate two possibilities.

3.2 Bottom-up traversal

Our first example takes these eleven trees and applies the same traversal as we introduced above – a bottom-up traversal that builds complete local trees working from the anchor outwards before moving to higher structure. Figure 3 shows the resulting eleven automata.

The automata transitions are labelled according to the convention for elementary computation steps introduced above. $\overleftarrow{\text{np}}$ denotes substitution of a noun phrase on the left, $\overrightarrow{\text{np}}$ denotes substitution of a noun phrase on the right and $\overleftrightarrow{\text{vp}}$ denotes adjunction at a verb phrase node. The final states are labelled with the number of the tree that they complete.

The first step in optimising these automata is to union them into a single (nondeterministic) automaton by introducing a new initial state and ϵ-transitions from it to the start states of each of the above automata. Once this has been achieved, we can then reduce and determinize the resulting automaton, and this gives us the automaton in Figure 4. Here, as well as the labels on the final states, we have labelled non-final states with the list of tree numbers (in italic script) whose EC passes through that state. This is not relevant to the parser, but may make the diagram easier to follow.

How does this automaton compare with the original eleven? It is difficult to compare the behaviour of one automaton with that of eleven operating in concert without committing to a particular parsing algorithm. So it may be more useful to compare our determinized automaton with the single union automaton described in the previous paragraph. Even then, it is not clear what metrics might be lead to useful comparisons, but the following three (interrelated) metrics seem to provide interesting insights:

1. the total number of states in the automaton, as a basic indicator of complexity.

2. the average number of trees per state: each state represents a point in the traversal of one or more of the trees. By averaging the number of trees associated with each state, we get a measure of the amount of sharing that has been achieved. In each of the original automata this figure is 1 (and in the union it is close to 1, the additional start state introduces a little sharing), but higher numbers reflect more sharing of structure.

3. the representational expansion factor, that is, the ratio of the number of tree/state combinations compared with the original automata. Larger values for this ratio represent the introduction of *additional* structure not present in the original representation (for example to ensure determinism).

The table in Figure 4 gives values for these metrics for the union automaton and the determinized automaton. We can see from these figures that we have nearly halved the number of states with every state doing work for

two trees on average. The expansion factor reflects the introduction of new states to cope with the optionality of adjunction in a deterministic setting. The third line of results is for a version of the determinized automaton which only performs recognition, included for interest although we have not included the automaton itself. As we noted above, a recognizer need only distinguish the category information associated with final states, not the licencing tree, and this results in additional sharing of structure. In this case the improvement is even more marked – a fourfold improvement in size on the original automata.

3.3 Left-right traversal

The English grammar used in the XTAG system (XTAG-Group, 1995) has the property that foot nodes of auxiliary trees are always at the extreme left or extreme right of the tree. This means that when adjunction takes place all the nodes of the adjoined tree lie to the right or to the left of the node at which adjunction occurred. For grammars with this property we can distinguish between *left adjunction* (where the foot is on the extreme right of the tree) and *right adjunction* (where it is on the extreme left), and assume that these are the only kinds of adjunction that occur, that is, that no *two-sided* adjunctions occur.

The previous example used a traversal of the trees which built complete structures bottom-up before moving to parent nodes. If no two-sided adjunctions are permitted, we can consider a different traversal, which attempts to work left first and then right, independently of local tree structure. Using the same trees and starting at the 'v' anchor node, this time we look first for left adjunctions at the parent 'vp', then for a subject, and only then do we look for complements and finally right adjunctions. For the same eleven trees introduced above, the set of automata resulting from this traversal is given in Figure 5, and the determinized version in Figure 6. We use ↓ vp to denote the left adjunction of a 'vp' auxiliary tree and vp ↓ to denote right adjunction of a 'vp' auxiliary tree. As in Figure 4 we have annotated states with the numbers of trees whose EC passes through that state, but notice that in this case some states are final for one tree, but non-final for others.

The values for the three metrics introduced above are also given in the table in Figure 6. From these figures we see that something slightly different has happened. Although the determinized automata have the same number of states in the two examples, in this second case the amount of sharing is also higher, but this is counterbalanced by the expansion factor. Looking closely at the automaton, we see that there is more effective sharing of state (because the left hand sides of our trees are more similar than the subcategorisation frames, roughly speaking), but we also had to introduce more states to cope with the splitting up of the adjunction into two phases. The figures for the recognizer are still significantly better, but there seems to have been less scope for compression, again due to the need for expansion to cope with the adjunctions.

4 Discussion

We have presented a formulation of lexicalized grammar parsing that lends itself to a range of precompilation optimisations, and given examples of two such precompilations. One issue that we have not yet made clear is exactly which sets of trees should be precompiled in this way. In a lexicalized grammar, the usual distinction between the lexicon and the grammar is reflected in the distinction within the lexicon between the Elementary Structures Database and the Word Database:

- The Elementary Structures Database (ESD) is a set of unanchored ES's (typically several hundred). An unanchored ES is one in which no lexical item has been associated. Collections of related ES's are often organized into families.
- The Word Database (WD) is an association of words (perhaps several hundred thousand) with those elements of the ESD that they can anchor. Typically a word will anchor all the members of a family, and may anchor structures from several families.

This architecture has interesting consequences for our precompilation approach. The set of ES's associated with a given lexical token seems an obvious target for FSA optimisation. However, many tokens share their set of ES's with other tokens, often many other tokens. The distinction between the ESD and the WD allows us to optimize structure in the ESD only, so that these tokens can continue to share the optimized structures. On the other hand some tokens have overlapping but non-identical ES sets. ES's in the intersection may be integrated into two different automata, possibly in different ways, optimized to the particular requirements of each token.

But this apparent duplication of grammatical structure has no processing disadvantage, since only one of the two representations will ever be associated with any given input token (though they may both be active in the parse as a whole).

We saw above that alternative traversals of the underlying elementary structures leads to different automata, and hence different merging of elementary computations. Because the parser operates directly on the automata, different choices of traversal are equally valid, and will result in different parsing strategies. In fact, there is no reason why different parts of a single grammar should not use different traversals. This might be motivated by the linguistic structure (strategies for verbal or nominal subcategorisation might differ from those for coordination, for example), or by the relative sizes of the automata themselves, but a particularly interesting approach would be to try and optimise performance on a statistical basis.

Time is wasted during parsing when locally licensed ECS's are made in situations that cannot lead to complete parses. Let us call these *useless* steps. since the order of ECS's in an elementary computation varies from one traversal to another, it is possible that some traversals will lead to less occurrences of useless steps than others. It should, in principle, be possible to use the information provided by a large parsed corpus to select traversals that minimize the number of useless steps that will arise.

Finally, it is interesting to note the duality between the kind of optimisation proposed here and the more traditional representational optimisation through the expression of linguistic generalisations. We have previously argued (Evans, Gazdar, and Weir, 1995) that lexicalized grammars can benefit from the representational tools developed for lexical representation (such as inheritance hierarchies, exceptions and lexical rules). The present work does not exploit such generalisations at all, but rather assumes a completely flat base grammar. The implicit architecture promoted here views optimisation for parsing as independent of linguistic generalisation. To a large extent this is a consequence of the underlying parsing framework (which also cannot exploit lingusitic generalisations). It would be interesting further work to explore the relationship between these linguistic and parsing optimisations, to see, for example, whether linguistic generalisations might inform the optimisation process, or conversely whether they might arise as emergent properties of parsing optimisations carried out for other reasons, or else to conclude that the two are indeed best treated as independent aspects of language structure.

References

Alshawi, Hiyan. 1996. Head automata and bilingual tilings: Translation with minimal representations. In *34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 167–176.

Evans, Roger, Gerald Gazdar, and David Weir. 1995. Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 77–84.

Joshi, Aravind K. and Yves Schabes. 1991. Tree-adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees*. Elsevier.

Rambow, Owen, K. Vijay-Shanker, and David Weir. 1995. D-Tree Grammars. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 151–158.

Schabes, Yves. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Vijay-Shanker, K. and Aravind Joshi. 1985. Some computational properties of tree adjoining grammars. In *23rd Meeting of the Association for Computational Linguistics (ACL'85)*, pages 82–93.

Vijay-Shanker, K. and David Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

XTAG-Group, The. 1995. A lexicalized tree adjoining grammar for English. Technical Report IRCS Report 95-03, The Institute for Research in Cognitive Science, University of Pennsylvania.

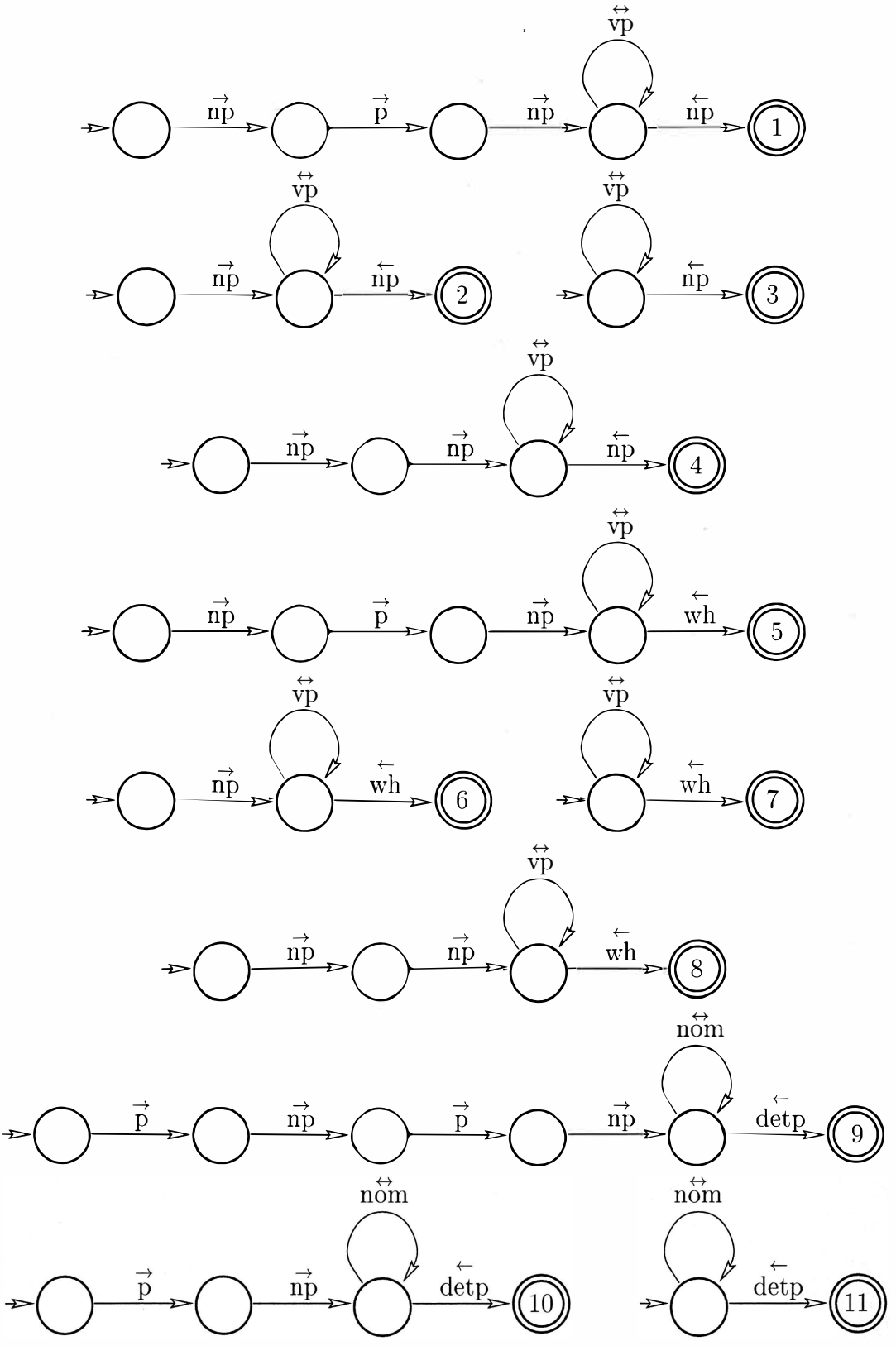
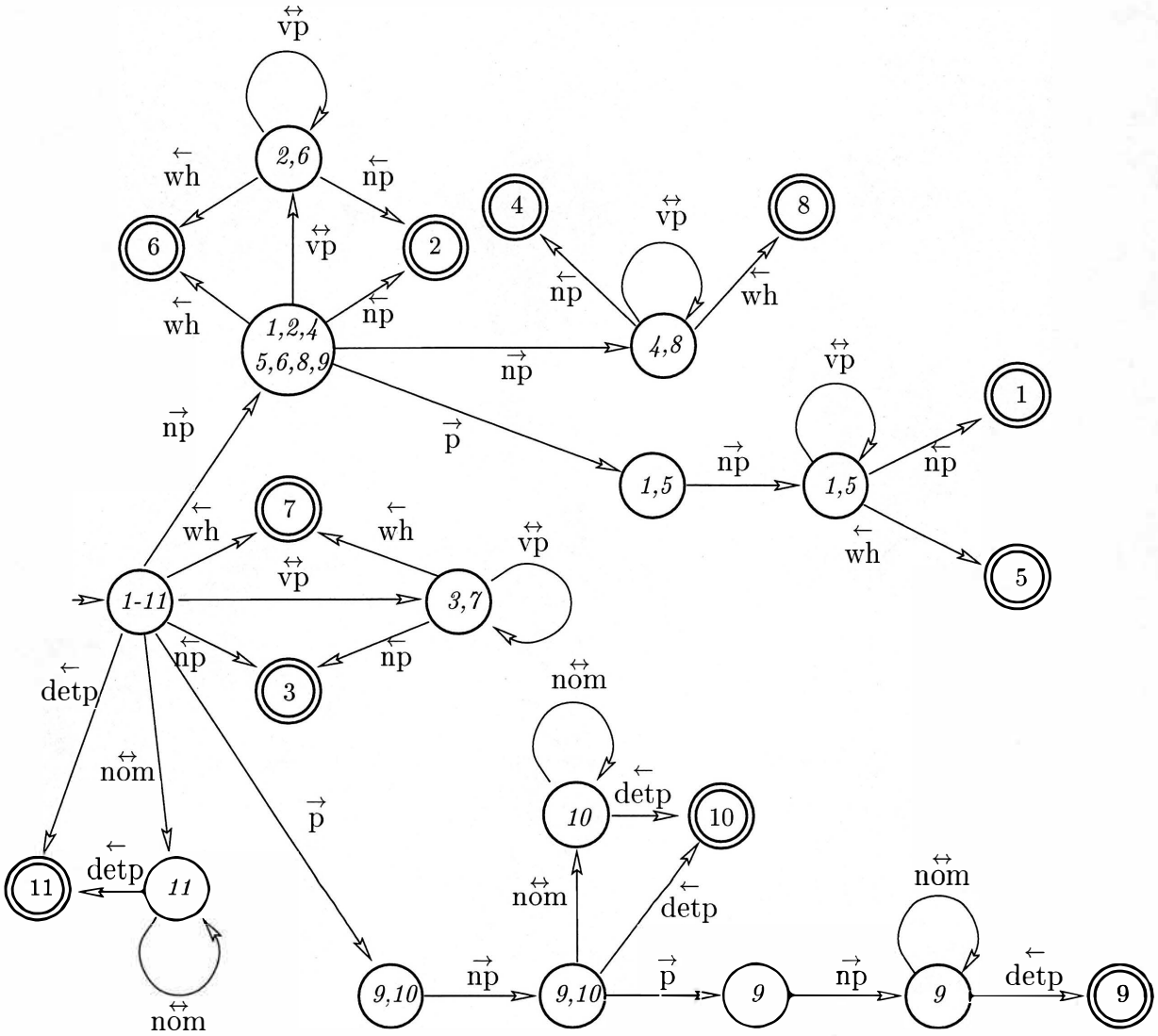


Figure 3: Bottom-up automata for the eleven trees



Metrics for BU Automaton	*States*	*Trees per state*	*Expansion factor*
Union of eleven automata	41	1.244	1.025
Determinized automaton (parser)	24	1.92	1.15
Determinized automaton (recognizer)	11	4	1.1

Figure 4: Bottom-up deterministic automaton for all eleven trees

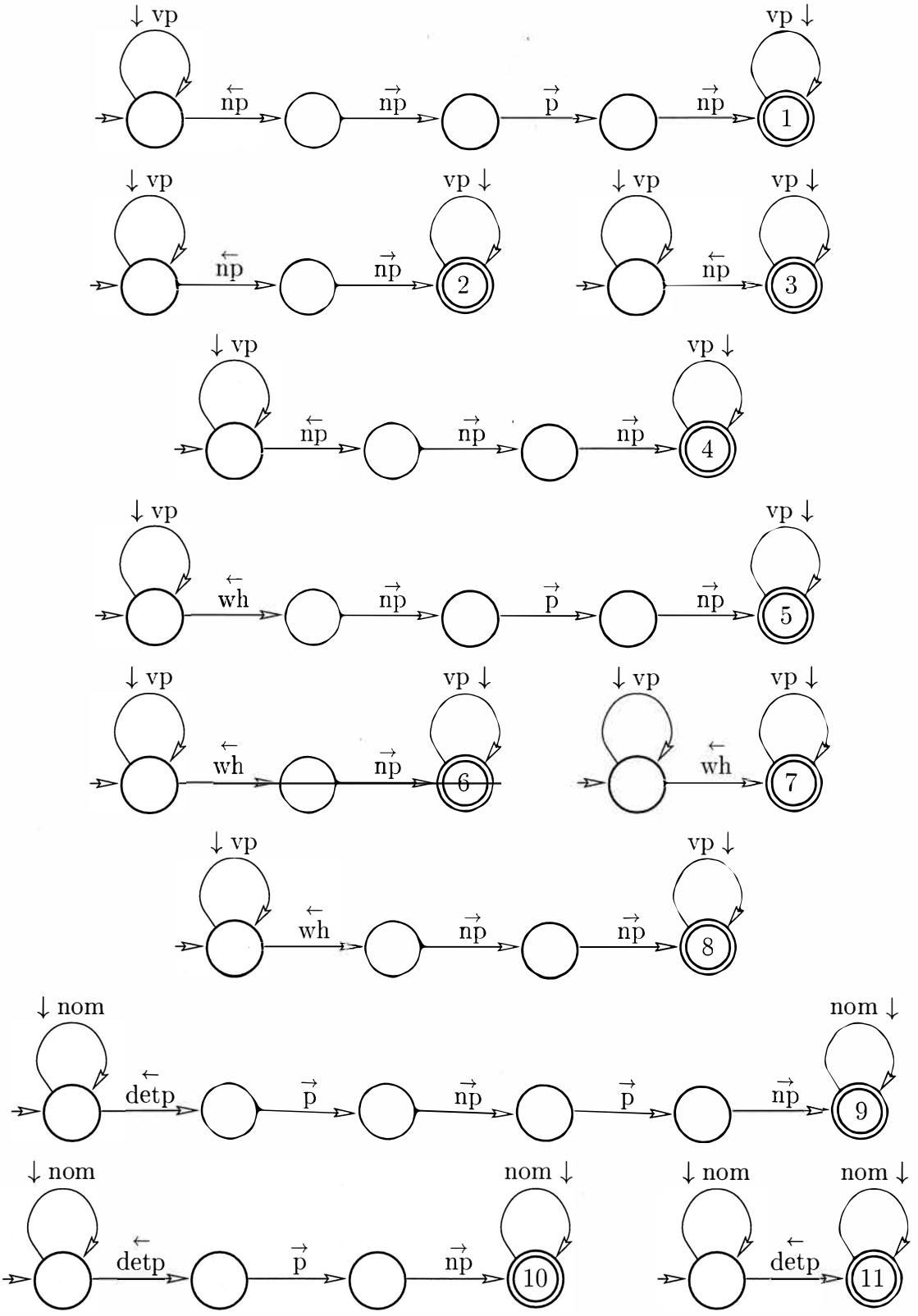
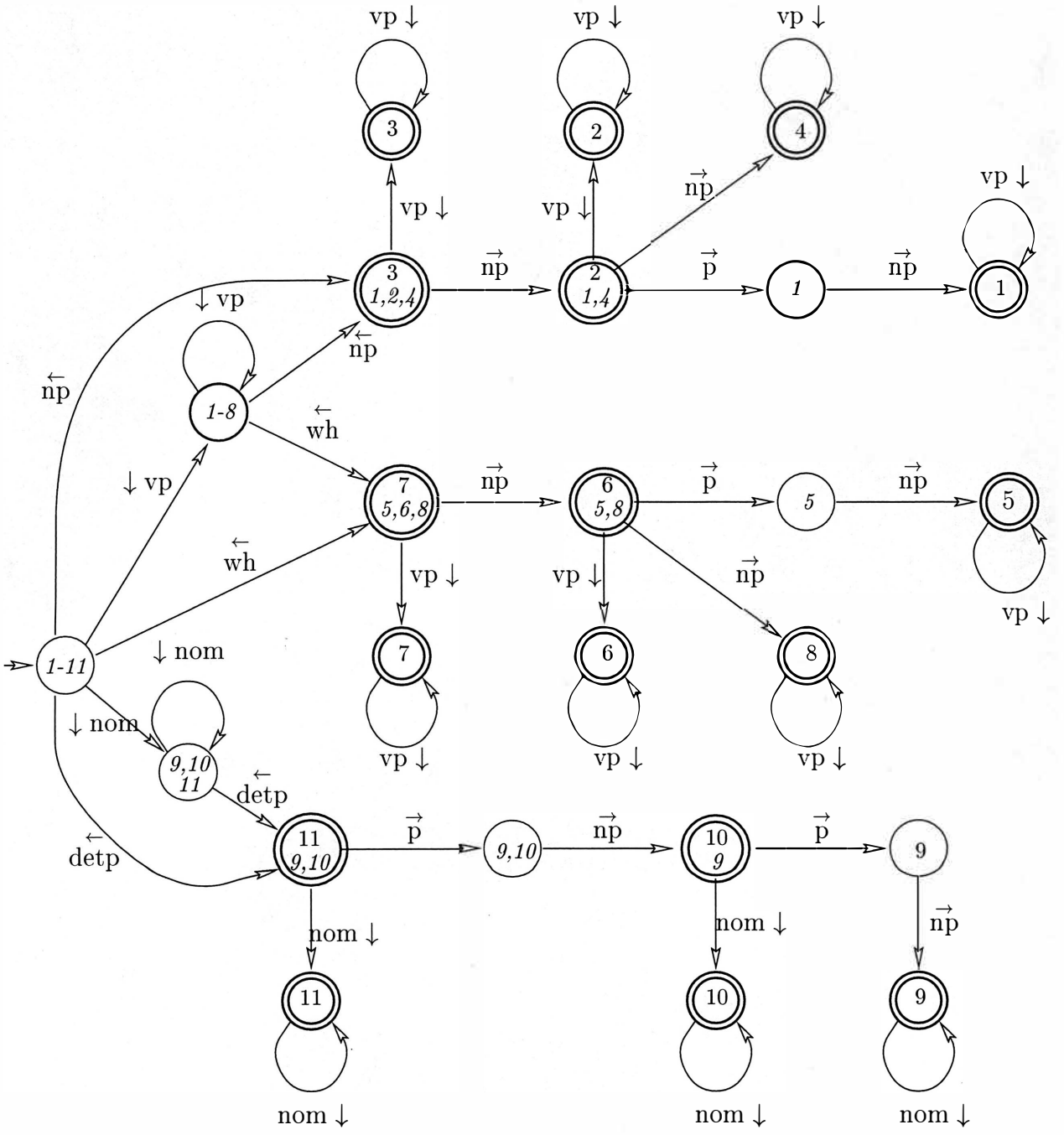


Figure 5: Left-right automata for the eleven trees



Metrics for L-R automaton	*States*	*Trees per state*	*Expansion factor*
Union of eleven automata	41	1.244	1.025
Determinized automaton (parser)	24	2.375	1.425
Determinized automaton (recognizer)	16	3.562	1.425

Figure 6: Left-right deterministic automaton for all eleven trees

From Part of Speech Tagging to Memory-based Deep Syntactic Analysis

Emmanuel Giguet and **Jacques Vergne**
GREYC — CNRS UPRESA 6072 — Université de Caen
14032 Caen cedex — France

{Emmanuel.Giguet,Jacques.Vergne}@info.unicaen.fr

Abstract

This paper presents a robust system for deep syntactic parsing of unrestricted French. This system uses techniques from Part-of-Speech tagging in order to build a constituent structure and uses other techniques from dependency grammar in an original framework of memories in order to build a functional structure. The two structures are build simultaneously by two interacting processes. The processes share the same aim, that is, to recover efficiently and reliably syntactic information with no explicit expectation on text structure.

1 Introduction

This paper describes a robust system for deep syntactic parsing of unrestricted French. In this system, deep syntactic analysis means identifying constituents and linking them together. A brief presentation of the problematics will clarify the choice we have made.

In western-european natural languages such as French, both constraints and liberties exist in respectively two distinct levels. The stylistic liberties of the French author (Molière, 1670) illustrate this phenomenon in the famous following lines:

"[Belle marquise], [vos beaux yeux] [me font] [mourir] [d'amour]"
"[D'amour] [mourir] [me font], [belle marquise], [vos beaux yeux]"
"[Vos beaux yeux] [d'amour] [me font], [belle marquise], [mourir]"

The order of small chunks of words (here bracketed) is fairly free. However, we point out a roughly fixed word-order within these chunks which strong constraints are applied to. These two kinds of opposite and independent behaviors in two distinct levels led us to consider that these levels could not be managed properly with a unique process and a unique representation. One will note that chunks are traditional syntactic groups as defined in (Le Goffic, 1993) *without* their dependents[1]. These chunks are Non-Recursive phrases (*nr*-phrases).

The first level, called word-level, is achieved within the framework of partial parsing also called shallow parsing (Voutilainen and Järvinen, 1995; Abney, 1996; Chanod and Tapanainen, 1996): it uses Part-Of-Speech Tagging and Chunking techniques in order to reliably and efficiently build up *nr*-phrases on unrestricted texts.

The second level, called *nr*-phrase-level, works with no explicit expectation on the input structure in order to deal with unrestricted text. We have developped our own approach to relations computation by revising the definition of specifying them. This approach allows a flexible management of both short and long dependencies. Moreover, it intrinsically enables the interaction with higher-level knowledge sources required in a framework of deep syntactic analysis.

Each of the two levels has its own process and its own representation; both representations are build simultaneously by the two interacting input-driven processes. The interaction is a requirement since many ambiguities arising at word-level can not be solved reliably there; the *nr*-phrase-level helps solving them.

Hereafter, we first describe the architecture of the parser. Then, we introduce the reader to the building of *nr*-phrases. Then, we emphasize on the way of linking *nr*-phrases together. After presenting how the two levels interacts, a concrete analysis is detailed. Then, we show the adequacy of the model, studying the resolution of some major linguistic problems. Finally, a precise evaluation is carried out, empirically demonstrating the adequacy of our novel concepts.

[1] Dependents are excluded in order to unify the representation of contiguous and discontiguous dependents.

2 The Architecture

The architecture of the process combines two techniques: (1) partial parsing, or more precisely Part-Of-Speech tagging techniques at word-level that build a constituent structure (each constituent is an *nr*-phrase); (2) dependency rules at *nr*-phrase-level that link *nr*-phrases to build a functional structure. In our approach, both constituent and functional structures are build simultaneously by two interacting processes. The analysis is carried out as shown in figure 1.

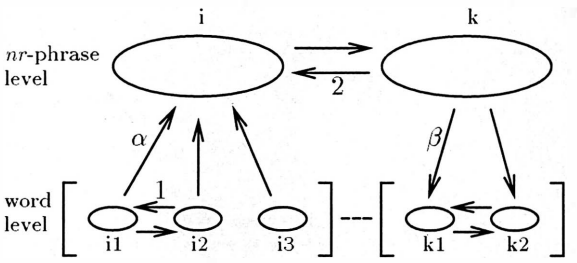


Figure 1: Process of analysis

Figure 1 shows two processes, labelled 1 and 2, managing respectively the word-level and the *nr*-phrase-level. The first process assigns tags to each part-of-speech and defines *nr*-phrase boundaries, shown as square brackets (section 3). The second process defines relations between *nr*-phrases (section 4). The two labels α and β show the interactions between word-level and *nr*-phrase-level (section 5). The execution of an entire basic cycle of deductions is successively: $1, \alpha, 2, \beta$.

The parse is done from left to right with a possible change of past deductions (and their consequences) when new knowledge invalidates them. Let us now describe precisely the two processes and their interaction.

3 Building Non-Recursive phrases

3.1 Framework

Word-level builds up *nr*-phrases with the help of an extended Part-Of-Speech (POS) tagger using hand-coded *affirmative* ϵ rules.

In rule-based POS tagging, two approaches exist. In the negative approach, a set of tags is first assigned to each token then contextual rules try to discard tags in order to avoid impossible contiguities. In the affirmative approach, contextual rules assign tags which are certain in the local context.

While tagging, two special tags are assigned as *nr*-phrase boundaries (beginning/end of *nr*-phrase). The *nr*-phrases correspond to "chunks" as defined in (Abney, 1996).

In this section, the problematics of tagging is assumed to be known and will not be completely detailed. Specific features required for the general understanding of the whole system are described and several tagging improvements are suggested.

3.2 Tag Assignment

The tagging is achieved in two steps: an initial step which can be seen as a bootstrap for a second step, called the deduction step.

The initial step is first to assign grammatical words with their most likely tag. One will note that during the deduction process, contextual rules may change these default tags. For instance, in French the ambiguity determiner/clitic[2] is solved by assigning the default tag *determiner*. Subsequently, contextual rules may change this tag into *clitic* if a personal pronoun, an other clitic or a verb appears in its local context. This process has a close relative in (Chanod and Tapanainen, 1995b; Constant, 1991). In (Brill, 1995), most likely tags are not only assigned to grammatical words (or most frequent words) but also to every other word.

[2] Clitic: French adverbial pronouns (*en,y*) and object personal pronouns (*le,la,les,lui,leur,l'*) always next to the verb.

In the initial step, we also use a lexicon containing all inflexional information on verbs (and their explicit polycategory) and a guesser that studies endings in order to give default assignments to other lexical words. Explicit polycategory of lexical words is pointed out by assigning sets of tags. During the deduction process, contextual rules may change the default tags to handle the incompleteness of knowledge(see section 3.4).

In the deduction step, all the rules have the following template: "the tag of this word is in this set of tags in this local context". Obviously, the set of tags contains only one item if the context is not ambiguous. If a set of tags is already assigned to a word, the intersection of the two sets is assigned to the word. If the intersection of the two sets is empty, the old set of tags is replaced by the new one.

While tagging, two special tags are assigned as *nr*-phrase boundaries (beginning/end of *nr*-phrase). The *nr*-phrases correspond to "chunks" as defined in (Abney, 1996). This is a generalization of "baseNP" used by (Collins, 1996) or of minimal NP to every other kind of phrases (e.g., verbal, adjectival). The syntactic features of *nr*-phrases are defined by the interaction α with the *nr*-phrase-level (see section 5.1).

3.3 Access to Knowledge Sources

In the earlier version of our system, access to the lexicon was always done first, then to morphological information and finally to contextual information. This strategy is also chosen in most systems but is this strategy the best one ?

Tests show that about 85% of the default tags that are assigned in the initial step are not changed by the contextual rules. However, contextual rules that change default tags always replace them ignoring their default value. Moreover, the changes done by the contextual rules are always right. Thus, in order to express that contextual information is always stronger than lexical and morphological information, the latter should only be accessed when no more information can be found through contextual deductions rather than be accessed solely in the initial step. Here is an analysis of the tagging process for *je le bois* (I drink it).
Notation:

$\xrightarrow{la}$: lexical access,
$\xrightarrow{ci}$: contextual information access.

Tagging with lexical access first:

$$je\ le\ bois \xrightarrow{la} je_{pp}\ le\ bois \xrightarrow{la} je_{pp}\ le_{det}\ bois \xrightarrow{la} je_{pp}\ le_{det}\ bois_{vb|nm} \xrightarrow{ci} je_{pp}\ le_{po}\ bois_{vb|nm} \xrightarrow{ci} je_{pp}\ le_{po}\ bois_{vb}.$$

Tagging with lexical access when required:

$$je\ le\ bois \xrightarrow{la} je_{pp}\ le\ bois \xrightarrow{ci} je_{pp}\ le_{po}\ bois \xrightarrow{ci} je_{pp}\ le_{po}\ bois_{vb}.$$

A partial implementation of this principle has been carried out and already shows a better interaction between the knowledge sources.

3.4 Handling Double Incompleteness of Lexicon

While processing unrestricted text, the problem of incompleteness of lexical knowledge is often pointed out. The incompleteness of lexicon is partly due to unknown words, such as domain-specific words, borrowings from one language to another, neologisms, spelling mistakes and so on (Chanod and Tapanainen, 1995a). It is also due to an interesting effect of missing categories for particular words in the lexicon.

The use of guessers is a traditional way to deal with unknown words but its sole advantage is to partially overcome one cause of the incompleteness. Furthermore, guessers can introduce irrelevant tags on unknown words and are efficient only on highly inflected languages. For these reasons, we have to find out complementary ways to handle incompleteness of lexical and morphological knowledge if we want to be able to deal with the common problems which usually arise while processing unrestricted text.

In this system, using the fact that contextual information is stronger than lexical and morphological information, we have added many affirmative contextual rules which can partially handle the two causes of incompleteness. For instance, a rule such as "after a personal pronoun, there is a verb" can handle sentences such as "*je positive.*" where *positive* is first person singular present of the verb *positiver*, a neologism that probably can not be found in a lexicon. This kind of rules can also deal with missing categories. For instance, if *bois* only occured as a verb in the lexicon (I drink), contextual rules could deduce it can also be a noun (wood) as in "*dans le bois*" (in the wood).

It is interesting to notice that negative constraints can not be used in this case since they discard tags assigned by a previous access to lexical or morphological information.

3.5 Reliability of Local Deductions

We saw in the introduction that *nr*-phrase order is fairly free, therefore, contiguous words of different *nr*-phrases are not supposed to have any relations.

With respect to this principle, when the local context defined by an *nr*-phrase contains too little information, some words should remain ambiguous since they can not be disambiguated reliably enough at word-level by POS tagging techniques. In other words, reliable deductions are those whose context has a scope restricted to one *nr*-phrase, using its inside strong syntactic constraints. Involving information outside this scope is not reliable since deductions that have a scope of several *nr*-phrases can not capture linguistic generalities.

Thus at word-level, we try to write rules propagating contextual deductions on words, and this solely within *nr*-phrases. Usually, these rules involve a grammatical word which helps categorizing the contiguous lexical words of their chunk. Remaining ambiguities are solved by the interaction with higher-level knowledge involved at *nr*-phrase-level.

4 Linking Non-Recursive phrases

4.1 Framework

The linguistic background of our work is based on the work of Lucien Tesnière (Tesnière, 1959). From his first approach to dependency definition "*Between a word and its neighbours, the mind foresees some connections.*", we have derived our own concepts for specifying *nr*-phrases relations. We have pointed out that the traditional static descriptive relation definition is not precise enough in order to be used efficiently in a parsing process. Thus, we introduce a dynamic analysis-oriented relation definition, that takes into account the linking constraints of the other relations. In our system, we have implemented this definition so that it handles all major dependency relations, the coordination relation and the antecedence relation.

The approach revises the notion of dependency as a relation between *nr*-phrases, and not between words, as opposed to (Covington, 1990; Covington, 1994; Tapanainen and Järvinen, 1997). As said in (Abney, 1996), "*By reducing the sentence to chunks* [i.e., *nr*-phrases], *there are fewer units whose associations must be considered, and we can have more confidence that the pairs being considered actually stand in the syntactic relation of interest, rather than being random pairs of words that happen to appear near each other*".

4.2 A new approach to relation specification

Dependency grammar-based formalisms allow for the specification of general relations by defining (1) the two structures being considered in the relation of interest and (2) static constraints existing between these two structures. This way of specifying relations leads to a failure since either the constraints on the structure of the two items are too relaxed and the silence is high, or they are too strict and the noise is too high.

Such static constraints on structures are unavoidable. Introducing constraints of possible or impossible occurences of structure between the two items can only lead to a failure since any such rule can be proved wrong within any unrestricted corpus.

Specifying a relation only with constraints on structure does not seem to be the right solution. Thus, we make the hypothesis that relations have to be defined with the help of three kinds of linguistic knowledge:

1. the two structures being considered in the relation of interest,
2. the constraints existing between these two structures,
3. the *other linking constraints* which, within the sentence, constrain these two items to be linked.

The two structures have to be computed *dynamically* in order to be robust. The constraints between the two structures are made of *static* knowledge. The interdependency of relations, as defined in point 3, has to be handled *dynamically*.

4.3 Dynamic handling of relation interdependencies

In our research, we have emphasized the handling of relation interdependencies which has become the predominant feature of our architecture. In other words, we have studied how the instanciation of a relation reduces the complexity of further decisions by discarding potential choices.

An example illustrates this general concept which happens to be the process definition.

Considering the sentence: "*The flight from Paris is cancelled because of a strike*"

By instanciating a subject-verb relation between *The flight* and *is cancelled*, the process discards *from Paris* as expecting any other relation. Thus, when *because of a strike* occurs it can only be attached to *The flight* or to *is cancelled*.

4.4 Implementation of the linking process

Our linking process is an implementation of the linguistic process described above.

The process is both data-driven and declarative: condition-action rules does not describe syntactic structures but the linking process. They manage both relations instanciation and linking constraints between relations. Relation instanciations are achieved in two distinct steps by two distinct kinds of rule actions:

1. *store* an *nr*-phrase as a candidate for some particular relations of interest in the relevant memories,
2. *attach* one *nr*-phrase to another one and *discard* some particular items as possible candidates for some particular relations from the relevant memories.

Thus, building up the functional structure is constrained by the interactions of the rules through the memories. In fact, when a rule discards an item in a memory, this corresponds to the death of a potential relation.

For instance, here are the two independent rules written to manage a subject-verb relation.

if the current *nr*-phrase
  is a nominal *nr*-phrase and
  is not object and
  is not already subject and
  is not attached to a preposition
then
  it is stored as possible subject
    into the subject-verb memory.

if the current *nr*-phrase
  is a verbal *nr*-phrase and
  there are possible subjects
    in the subject-verb memory
then
  retrieve the best fit subject from the memory
  attach the verb to this subject,
  discard this subject from the memory,
  discard items located between the subject and
    the verb from every memory.

The method which selects the best fit candidate is described in section 4.5.

The conditions within the rules allow the manipulation of: (1) relations in the dependency tree (defined by the functional structure); (2) heads of *nr*-phrases; (3) features of *nr*-phrases; (4) and status of the memories.

There are two kinds of actions within the rules: (1) actions on a memory (storing an *nr*-phrase and linking two *nr*-phrases, erasing the content of a memory, discarding an item in a memory), (2) actions on an *nr*-phrase (to change/add a feature).

The analysis is carried out from left to right. When an action updates *nr*-phrase features, coherence with word-level is achieved via interactions (see section 5.2).

Such an implementation requires the system to store candidates for possible expectations (e.g, nominal *nr*-phrase possibly expecting a verb for a subject-verb relation) and to retrieve the best-fit candidate for a particular relation. This ability is provided by the memory-based framework.

4.5 Memory-Based Framework

Memories as favoured places to perform relations

The process is based on a set of memories. Each memory is dedicated to the management of one specific relation (e.g, subject-verb, verb-object, coordination, PP attachment). A memory contains *nr*-phrases whose

association with a future *nr*-phrase must be considered. For instance, the memory that manages the subject-verb dependency relation contains nominal *nr*-phrases which can be involved in a future relation with a verbal *nr*-phrase.

The power of such an approach is that all relevant candidates are together in a single location and at the time when the relation has to be computed (a memory is a limited search-space): for a specific relation, the required knowledge sources can choose a successful candidate in the best conditions (see section 4.5).

Moreover, when the selection has to be performed, the process does not have to consider the past of the analysis but the current state of the memories. Therefore, far discontiguous relations are handled the same way as contiguous relations (if necessary, ways to distinguish them exist).

An other interesting point is that memories contain candidates for an association with a future *nr*-phrase. No requirement is made on the presence of the future *nr*-phrase. If such an *nr*-phrase never occurs then at the end of the sentence, the memory is erased: the candidates are forgotten. This means that when a new *nr*-phrase is added to a memory, no explicit expectation on structure is done, only implicit expectations are described by the rules. This kind of behaviour is to be related to tagging techniques and is fundamental to deal with unrestricted text.

Selection in a memory

Each memory is dedicated to the management of a specific relation. It is obvious that the knowledge required for selecting a candidate in the different memories is not always the same. In this system, every memory has its own specific method for choosing the successful candidate.

For instance, in our system, syntactic knowledge is involved for constraining the search space (i.e., the memory) depending on number, person and gender in a subject-verb dependency relation; similarity of structures is considered for coordination relation; psycholinguistic knowledge constrain the distance between the future associated *nr*-phrases.

It is interesting to point out that the above-mentioned knowledge sources are not enough to deal with complex phenomena. In memories, semantic and pragmatic knowledge sources can also interact with other knowledge sources to constrain the search space.

Focusing on the Subject-Verb memory

It is interesting to show in a concrete way how modularity of memories leads to flexibility, and to clarify how it helps us mastering the triggering of adequate knowledge sources and which items the triggered sources will act on.

The subject-verb memory is an example of such a memory where several kinds of knowledge are combined in order to handle the corresponding relation in a reliable and robust way. We will see that all the knowledge which deals with subject selection is clearly located in a single place:

- Syntactic constraints on agreement: these constraints are based on coordination relations, person and number of *nr*-phrases.

- Structural constraints on *nr*-phrase: they are involved in specific configurations in order to favour subject with determiner rather than subject without determiner.

- Basic semantic constraints are used to avoid some particular temporal NP to be taken as subject.

- This memory selects the leftmost possible subject close to the first barrier (e.g, a relative pronoun, a subordination conjunction) located on the left-hand side of the verb. This models the linking process of a subject with its verb, taking into account embedded clauses.

The latter shows the tight links between memories and the dynamic linking process which feeds them. Selection in memories is usually achieved with the help of a standard constraints relaxation mechanism.

5 Interaction between levels

As seen in section 2, the two levels are build up simultaneously, so that information has to be sent from one level to the other for the sake of coherence. The two processes parse the input from left to right. We choose to make deductions as soon as possible in the two structures even if, sometimes, this implies the ability to change some decisions when new information appears.

To every *nr*-phrase defined at *nr*-phrase-level corresponds a chunk of words delimited at word-level by two *nr*-phrase boundaries. Deductions from word-level to *nr*-phrase-level have a restricted scope: from words within a chunk to their corresponding *nr*-phrase, and vice-versa.

5.1 Interaction from word to *nr*-phrase

The interaction from word-level to *nr*-phrase-level (labelled α in figure 1) allows the assignment of features to *nr*-phrases. These informations activate the *nr*-phrases in the building process of the functional structure.

Un fichier d'incidents complète le dispositif. (1)

(A file of incidents completes the device.)

For instance, in sentence 1, when *masculine singular determiner* is assigned to "*Un*" at word-level, a nominal *nr*-phrase is created and the feature *masculine singular* is added to this *nr*-phrase.

5.2 Interaction from *nr*-phrase to word

The interaction from *nr*-phrase-level to word-level (labelled β in figure 1) is needed to constrain word features. In fact, we explained in section 3.5 that the local context of a word can contain too little information to be fully disambiguated.

For instance, in sentence 1, as soon as we know at *nr*-phrase-level that "*complète*" is a verbal *nr*-phrase (because it has been linked to its subject "*Un fichier*"), the tag *verb* is assigned to "*complète*" at word-level, thus removing the *adjective/verb* ambiguity.

6 Detailed Analysis

Here is a detailed analysis of the sentence "*Un fichier d'incidents complète le dispositif.*" (a file of incidents completes the device) as solved by our system. The word *complète*$_{adj|vb}$ has to be disambiguated. (1) and (2) are the two processes; (α) and (β) are the two interactions defined in section 2.

• 1st execution cycle :
(1) builds the chunk "*un fichier*" (square bracketed),
(α) creates the corresponding NP (oval in the Figure),
(2) stores this NP in two memories: as possible subject, and as possibly expecting a PP
(β) does nothing.

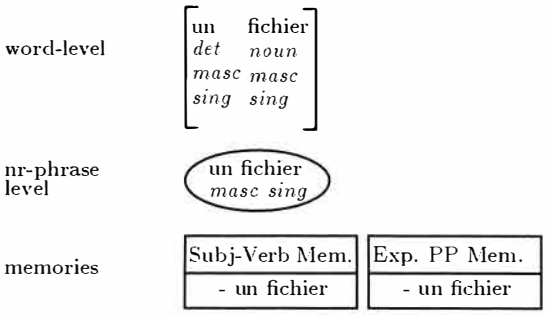


Figure 2: Cycle 1

• 2nd execution cycle:
(1) builds the chunk "*d'incidents*",
(α) creates the corresponding PP: PP_1,
(2) selects in the "Exp. PP memory" the NP "*un fichier*" as regent of PP_1, stores PP_1 as possible regent of an other PP.
(β) does nothing.

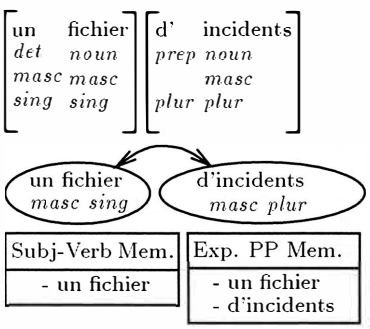


Figure 3: Cycle 2

• 3rd execution cycle:
(1) builds the chunk "*complète*" (the word "*complète*" is ambiguous),
(α) creates the ambiguous corresponding VP|AP (verbal or adjectival *nr*-phrase),
(2) supposes it is a VP, finds a matching NP as possible subject ("*un fichier*"), tags the ambiguous *nr*-phrase as VP, then it links this VP to the NP, and updates the memories,
(β) tags the word "*complète*" as *verb* for coherence.
This figure displays the chunk structure after (1) and the structure in progress at the end of the third cycle.

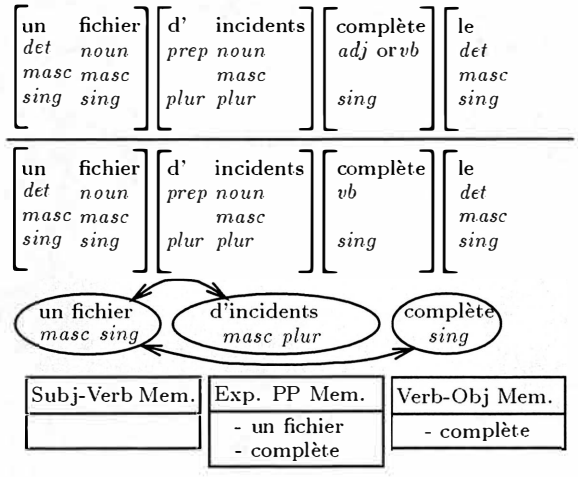


Figure 4: Cycle 3

• 4th execution cycle:
(1) builds the chunk "*le dispositif*",
(α) creates the corresponding NP,
(2) finds the matching VP "*complète*" to attach this NP as object, stores the NP as possibly expecting a PP.
(β) does nothing.

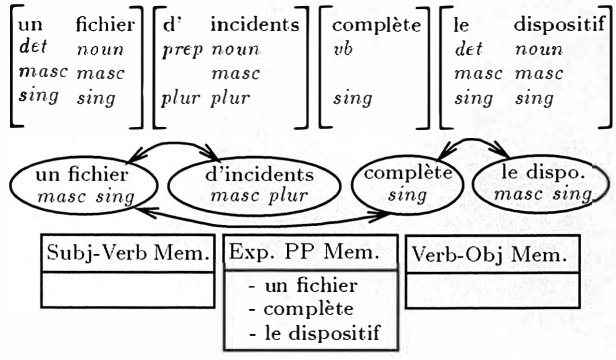


Figure 5: Cycle 4

At the end of the sentence, some memories contains items: they are erased.

7 Some linguistic studies

Some well-known difficult problems find a solution in this framework.

• *soit... soit* (either... or)
The first "*soit*" is either the subjonctive form of "*être*" (to be) or a conjunction (either).
"*C'est à l'inspection académique désormais de proposer aux familles, (...), soit un autre établissement - ce qui reviendrait sans doute à déplacer le problème - soit une formule de cours par correspondance (...).*"
In our framework, the previous example is processed as follows: the first "*soit*" is not the subjonctive form of "*être*" because no subject is available; it is stored in a memory, expecting an eventual second "*soit*"; when this second "*soit*" arrives, the first "*soit*" is found in the memory, and a coordination between the two nominal *nr*-phrases is set up.
The structure *ni... ni* (neither... nor) is solved in the same way.

• *plus... que* (more... than)
"*Les appels à la recomposition (...) me paraissent relever plus de la mise en valeur d'un parti socialiste syndical (...) que de la construction d'une réelle organisation syndicale indépendante.*"
"*plus*" is stored in a memory, expecting an eventual "*que*"; then, the following PP is linked to "*relever*"; when "*que*" occurs, its following PP is also linked to "*relever*".

• "*de*": *preposition* or *partitive* is a recurrent problem in French. It finds a natural solution in our system.
"*De nombreuses molécules sont transférées dans les réseaux trophiques marins.*"
"*De nombreuses molécules*" is an ambiguous *nr*-phrase: NP if the *nr*-phrase is either subject or object, PP otherwise. If it is NP then "*De*" is a *determiner*, otherwise it is a *preposition*: the solution is impossible to find

with POS tagging. In our system, when the verb "*sont transférées*" occurs, the process call the subject-verb memory to look for a subject, "*De nombreuses molécules*" is the only possible candidate: it becomes NP and an interaction tags "*De*" as *partitive*.

• Verbal ellipsis resolution:

Ellipsis resolution is a major difficulty to handle in traditional formalisms. In our approach, it can be handled as relation computation, defining it as a process.

"*Les lapins ne se contentent plus de luzerne, les cochons de pommes de terre et les poulets de grains de maïs*"
(Rabbits do not contend themselves with grass, pigs with potatoes and chicken with corn seeds.)

This kind of ellipsis can be handled in our model as an enumeration of possible clauses whose heads are subjects: *Rabbits*, *pigs* and *chicken*. This enumeration is managed as every other relation in a dedicated memory. When *with potatoes* occurs, its similarity with the verb dependent *with grass*, the absence of Subject-Verb relation involving *pigs* and the presence of a possible elliptic verb in the ellipsis memory allows *with potatoes* to be linked to *pigs* via a Subject-Verb-Object relation. Then, the Subject-Verb relation *pigs-contend* and Verb-PP *contend-with potatoes* can be computed.

8 Evaluation

The evaluation has been carried out on both structures build by the parser: a tagging evaluation and a relation computation evaluation.

The precise evaluation we offer on relations is restricted to subject-verb relations since no french treebank is available yet. However, it is possible to use our syntactic parse viewer on internet at http://www.info.unicaen.fr/~giguet (for Java-enabled browsers) to have an idea of the parser reliability for other relations.

8.1 Corpus Metrics

The evaluation of the parser has been carried out on a set of articles from the newspaper "Le Monde". This corpus has not been used to build up the parsing rules. This set is made of 24 articles (dealing with politics, economics, fashion, high-technology, every day life, ...) representing 474 sentences (max. length: 82 words, avg. length: 24.43 words). The definition of sentence is standard but includes two additional boundaries ";" and ":".

8.2 Tagging Evaluation

The corpus has been manually annotated by a linguist from the GRACE[3] tagging evaluation project, with the standard tagset proposed in MULTEXT[4] and EAGLES[5] projects. This tagset is made of 11 main categories. Each category can be completed with a maximum of 6 attributes which can take their value from a set containing up to 8 distinct values.

For the needs of the evaluation, a translation function has been written to convert our tagset into the tagset of the annotated corpus.

The fine grain tokenization (e.g., apostrophies are tokens) makes 12691 tokens appear.

In the protocol, an assigment is (1) *correct* if the parser assigns one tag with the correct value to every field, (2) *ambiguous* if the parser assigns more than one tag or more than one field value but the correct tag exists, (3) *incorrect* if the correct tag can not be found.

Two evaluations are carried out. In the first one, the tag has only one field: the main category. In the second one, the tag is composed of the main category and the relative attributes.

Figure 6 presents the results. These results are not the official results of the GRACE evaluation project since the evaluation contest is not started yet.

These results are computed from the output of the parser, that is, not only with process 1 at word-level but also with interactions β generated at *nr*-phrase-level when relations are computed by process 2 (see figure 1). About 1% of the correct tags are computed thanks to a relation computation, using interactions β, the others are computed at word-level.

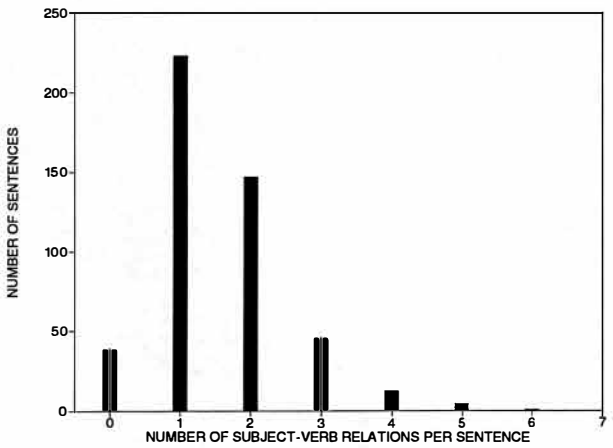
[3] http://www.ciril.fr/~pap/grace.html
[4] http://www.lpl.univ-aix.fr/projects/multext
[5] http://www.ilc.pi.cnr.it/EAGLES/home.html

Evaluation	tokens	correct	ambiguous	incorrect	% correct
Complete Tag	12691	11502	516	673	**90.63%**
Main Category	12691	12524	0	167	**98.68%**

Figure 6: Tagging Evaluation

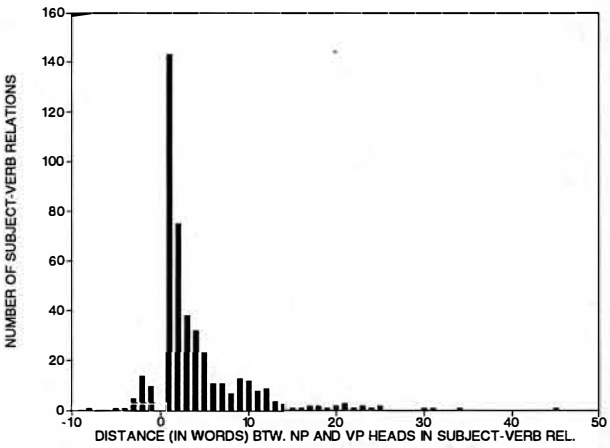
8.3 Relation computation evaluation

Subject-Verb relations in the corpus



nature of subject	number
NP	458
Infinitive VP	2
Relative Pronoun	85
Personal Pronoun	193
Total	738

Figure 7: Subject-Verb Relations



Max. dist. btw. NP-subject and VP heads	
in standard relations	45 words
in inverted relations	8 words

Figure 8: NP-subject in Subject-Verb Relations

In this corpus, there are 738 Subject-Verb relations. Figure 7 shows the span of subject-verb relations in the sentences: 39 sentences do not have subject verb relations and the maximum number of subject-verb relation per sentence is 6. According to the nature of the subject, we distinguish 4 kinds of SV relations: relations involving (1) an NP subject, (2) an Infinitive VP subject, (3) a Relative Pronoun subject and (4) a Personal Pronoun subject.

An other interesting metric is the distance between the verb and its subject. Figure 8 illustrates this phenomenon only for NP-subject. This metric is less relevant for other relations, even if several sentences contain personal pronouns and relative pronouns that are far subjects, for instance in cases of verb enumeration or prepositional phrase insertion. The figure shows that the distance between an NP-subject and its verb can reach up to 45 words.

Evaluation on Subject-Verb Relations

The evaluation function is based on the following principle: every verb has to be attached to no more than one subject. From this starting point, 3 cases exist: it is a *correct* relation if the verb is attached to the expected subject, *incorrect* if not and a *silence* if no subject is provided but one was expected.

In cases of subjects coordination, each verb depending on the coordination has to be attached to the head of this coordination, that is, to the head of the first item. In cases of verbs coordination, one correct relation counts for each verb attached to the expected subject and one incorrect relation for each verb not attached to the expected subject.

The results are listed in Figure 9. *Precision* is the ratio of correct links over the number of computed links. *Recall* is the ratio of correct links over the number of expected links. The very high rate that we report (96.39% precision and 94.04% recall) empirically validate the approach of defining relations as a linguistic process.

Our results can still be improved since this evaluation was the first on large corpora. The 42 silences and incorrect relations can be classified in 5 categories: (1) incorrect implementation of agreement check, (2) illformed *nr*-phrases, (3) coordination not found, (4) inverted subject in reported speech, (5) incorrect *nr*-phrase tags. We have pointed out better ways of solving the three first classes. The fourth and fifth classes requires further studies to be carried out in a general way.

Nature of subject	number	correct	incorrect	silence	precision	recall
NP	458	418	26	14	94.14%	91.27%
Infinitive VP	2	2	0	0	100.00%	100.00%
Relative Pronoun	85	85	0	0	100.00%	100.00%
Personal Pronoun	193	191	0	2	100.00%	98.96%
Total	738	694	26	16	**96.39%**	**94.04%**

Figure 9: Evaluation on Subject-Verb Relations

Comparison with other systems

It is still difficult to compare these results with other french systems since no strictly comparable experiment has been carried out under such difficult evaluation conditions. However, Xerox has evaluated its last incremental finite-state parser(Aït-Mokhtar and Chanod, 1997) for *subject recognition*. This task is less complex than ours since subject-verb relation computation includes the resolution of both subject and verb coordinations. On a half-sized evaluation corpus from "Le Monde" (5872 words, 249 sentences), their parser achieves 92.6% precision and 82.6% recall.

9 Conclusion and Future Work

We have described a system for syntactic parsing of unrestricted French. The contribution of this work can be summarized in two points. First, we have shown that a restriction of POS tagging to deductions within *nr*-phrase could lead to better global results since an interaction with the linking process is more powerful for deductions between *nr*-phrases. Second, we have provided a flexible memory-based framework for unrestricted relations and a process which has no explicit expectation on structures.

The success of our system is due to the new approach to computing relations: dynamically taking into account all the relevant relations which, within the sentence, constrain the two items to be linked.

The result is a flexible architecture which has the ability to handle in a natural way all the major syntactic relations in a unique framework: standard relation such as subject-verb, verb-object, PP attachment, but also complex relations such as coordination, enumeration, apposition, antecedence, and ellipsis.

Running on a collection of newspaper articles from "Le Monde" (11583 words, 474 sentences and 739 subject verb relations) where very complex structures appear, we get 96.39% precision and 94.04% recall for suject-verb relations. These first results empirically validate the approach and we can say the parser is very reliable for this relation. Moreover, it is robust since one parse is always provided (sometimes a partial parse). The present version of the linking process is very efficient: it is deterministic and it has a linear complexity in time. Today, we are working on a slightly modified version of the parsing process in order to enable new knowledge to change past deductions. In this case, these deductions and their consequences are discarded.

We now have to continue precise evaluation of our parser for all the other kinds of relations (a hard work since no treebank is available at the moment) and generally to continue improving the parser. A demo is available at http://www.info.unicaen.fr/~giguet.

References

Steven Abney. 1996. Part-of-speech tagging and partial parsing. In Ken Church, Steve Young, and Gerrit Bloothooft, editors, *An Elsnet Book*, Corpus-Based Methods in Language and Speech. Kluwer Academic, Dordrecht.

Salah Aït-Mokhtar and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proceedings of the fifth Conference on Applied Natural Language Processing (ANLP'97)*, pages 72–79, Washington, DC USA, April. Association for Computational Linguistics.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, December.

Jean-Pierre Chanod and Pasi Tapanainen. 1995a. Creating a tagset, lexicon and guesser for a french tagger. In *Proceedings of the European Chapter of the ACL SIGDAT Workshop "From text to tags : Issues in Multilingual Language Analysis"*, pages 51–57, Dublin, Ireland, March.

Jean-Pierre Chanod and Pasi Tapanainen. 1995b. Tagging french — comparing a statistical and a constraint based method. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)*, Dublin, March. Association for Computational Linguistics.

Jean-Pierre Chanod and Pasi Tapanainen. 1996. A robust finite-state parser for french. In *ESSLLI'96 workshop on robust parsing*, Prague, Czech, August.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, University of California, Santa Cruz, June. Association for Computational Linguistics.

Patrick Constant. 1991. *Analyse Syntaxique Par Couches.* Ph.D. thesis, École Nationale Supérieure des Télécommunications, April.

Michael A. Covington. 1990. A dependency parser for variable-word-order languages. Technical Report AI-1990-01, Artificial Intelligence Programs, The University of Georgia Athens, Georgia 30602 USA, January.

Michael A. Covington. 1994. Discontinuous dependency parsing of free and fixed word order: Work in progress. Technical Report AI-1994-02, Artificial Intelligence Programs, The University of Georgia Athens, Georgia 30602 USA.

Pierre Le Goffic, 1993. *Grammaire de la Phrase Française*, chapter 1-2, pages 7–51. Hachette Éducation.

Molière, 1670. *Le Bourgeois gentilhomme*, pages 39–48. Presse Pocket. Acte II Scène 4.

Pasi Tapanainen and Timo Järvinen. 1997. A non-projective dependency parser. In *Proceedings of the fifth Conference on Applied Natural Language Processing (ANLP'97)*, pages 64–71, Washington, DC USA, April. Association for Computational Linguistics.

Lucien Tesnière. 1959. *Éléments de syntaxe structurale.* Klincksieck, Paris.

Atro Voutilainen and Timo Järvinen. 1995. Specifying a shallow grammatical representation for parsing purposes. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)*, Dublin, Ireland, March. Association for Computational Linguistics.

Probabilistic Feature Grammars *

Joshua Goodman
Harvard University
40 Oxford St.
Cambridge, MA 02138
Email: goodman@eecs.harvard.edu

Abstract

We present a new formalism, *probabilistic feature grammar* (PFG). PFGs combine most of the best properties of several other formalisms, including those of Collins, Magerman, and Charniak, and in experiments have comparable or better performance. PFGs generate features one at a time, probabilistically, conditioning the probabilities of each feature on other features in a local context. Because the conditioning is local, efficient polynomial time parsing algorithms exist for computing inside, outside, and Viterbi parses. PFGs can produce probabilities of strings, making them potentially useful for language modeling. Precision and recall results are comparable to the state of the art with words, and the best reported without words.

1 Introduction

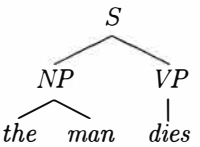
Recently, many researchers have worked on statistical parsing techniques which try to capture additional context beyond that of simple probabilistic context-free grammars (PCFGs), including Magerman (1995), Charniak (1996), Collins (1996; 1997), Black, Lafferty, and Roukos (1992), Eisele (1994) and Brew (1995). Each has tried to capture the hierarchical nature of language, as typified by context-free grammars, and to then augment this with additional context sensitivity based on various *features* of the input. Unfortunately, none of these works combines the most important benefits of all the others, and most lack a certain elegance. We have therefore tried to synthesize these works into a new formalism, *probabilistic feature grammar* (PFG). PFGs have several important properties. First, PFGs can condition on features beyond the nonterminal of each node, including features such as the head word or grammatical number of a constituent. Also, PFGs can be parsed using efficient polynomial-time dynamic programming algorithms, and learned quickly from a treebank. Finally, unlike most other formalisms, PFGs are potentially useful for language modeling or as one part of an integrated statistical system (e.g. Miller et al., 1996) or for use with algorithms requiring outside probabilities. Empirical results are encouraging: our best parser is comparable to those of Magerman (1995) and Collins (1996) when run on the same data. When we run using part-of-speech (POS) tags alone as input, we perform significantly better than comparable parsers.

2 Motivation

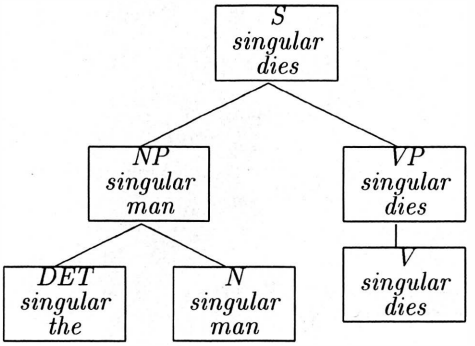
PFG can be regarded in several different ways: as a way to make history-based grammars (Magerman, 1995) more context free, and thus amenable to dynamic programming; as a way to generalize the work of Black et al. (1992); as a way to turn Collins' parser (Collins, 1996) into a generative probabilistic language model; or as an extension of language-modeling techniques to stochastic grammars. The resulting formalism, which is relatively simple and elegant, has most of the advantages of each of the systems from which it is derived.

Consider the following simple parse tree for the sentence "The man dies":

* This material is based upon work supported by the National Science Foundation under Grant No. IRI-9350192 and a National Science Foundation Graduate Student Fellowship. I would like to thank Stanley Chen, Lillian Lee, David Magerman, Wheeler Ruml, Stuart Shieber, and Fernando Pereira for helpful comments and discussions, as well as the anonymous reviewers for their useful comments and suggestions.



While this tree captures the simple fact that sentences are composed of noun phrases and verb phrases, it fails to capture other important restrictions. For instance, the NP and VP must have the same number, both singular, or both plural. Also, a man is far more likely to die than spaghetti, and this constrains the head words of the corresponding phrases. This additional information can be captured in a parse tree that has been augmented with features, such as the category, number, and head word of each constituent, as is traditionally done in many feature-based formalisms, such as HPSG, LFG, etc.



While a normal PCFG has productions such as

$$S \rightarrow NP\ VP$$

we will write these augmented productions as, for instance,

$$(S, singular, dies) \rightarrow (NP, singular, man)(VP, singular, dies)$$

In a traditional probabilistic context-free grammar, we could augment the first tree with probabilities in a simple fashion. We estimate the probability of $S \rightarrow NP\ VP$ using a tree bank to determine $\frac{C(S \rightarrow NP\ VP)}{C(S)}$, the number of occurrences of $S \rightarrow NP\ VP$ divided by the number of occurrences of S. For a reasonably large treebank, probabilities estimated in this way would be reliable enough to be useful (Charniak, 1996). On the other hand, it is not unlikely that we would never have seen any counts at all of

$$\frac{C((S, singular, dies) \rightarrow (NP, singular, man)(VP, singular, dies))}{C((S, singular, dies))}$$

which is the estimated probability of the corresponding production in our grammar augmented with features.

The introduction of features for number and head word has created a data sparsity problem. Fortunately, the data-sparsity problem is well known in the language-modeling community, and we can use their techniques, n-gram models and smoothing, to help us. Consider the probability of a five word sentence, $w_1 ... w_5$:

$$P(w_1 w_2 w_3 w_4 w_5) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1 w_2) \times P(w_4|w_1 w_2 w_3) \times P(w_5|w_1 w_2 w_3 w_4)$$

While exactly computing $P(w_5|w_1 w_2 w_3 w_4)$ is difficult, a good approximation is $P(w_5|w_1 w_2 w_3 w_4) \approx P(w_5|w_3 w_4)$.

Let $C(w_3 w_4 w_5)$ represent the number of occurrences of the sequence $w_3 w_4 w_5$ in a corpus. We can then empirically approximate $P(w_5|w_3 w_4)$ with $\frac{C(w_3 w_4 w_5)}{C(w_4 w_5)}$. Unfortunately, this approximation alone is not enough; there may still be many three word combinations that do not occur in the corpus, but that should not be assigned zero probabilities. So we *smooth* this approximation, for instance by using

$$P(w_5|w_3 w_4) \approx \lambda_1 \frac{C(w_3 w_4 w_5)}{C(w_3 w_4)} + (1 - \lambda_1) \left(\lambda_2 \frac{C(w_4 w_5)}{C(w_4)} + (1 - \lambda_2) \frac{C(w_5)}{\sum_w C(w)} \right)$$

Now, we can use these same approximations in PFGs. Let us assume that our PFG is binary branching and has g features, numbered $1...g$; we will call the parent features a_i, the left child features b_i, and the right child features c_i. In our earlier example, a_1 represented the parent nonterminal category; a_2 represented the parent number (singular or plural); a_3 represented the parent head word; b_1 represented the left child category; etc. We can write a PFG production as $(a_1, a_2, ..., a_g) \rightarrow (b_1, b_2, ..., b_g)(c_1, c_2, ..., c_g)$. If we think of the set of features for a constituent A as being the random variables $A_1, ..., A_g$, then the probability of a production is the conditional probability

$$P(B_1 = b_1, ..., B_g = b_g, C_1 = c_1, ..., C_g = c_g | A_1 = a_1, ..., A_g = a_g)$$

We write a_1^k to represent $A_1 = a_1, ..., A_k = a_k$, and sometimes write a_i as shorthand for $A_i = a_i$. We can then write this conditional probability as

$$P(b_1^g, c_1^g | a_1^g)$$

This joint probability can be factored as the product of a set of conditional probabilities in many ways. One simple way is to arbitrarily order the features as $b_1, ..., b_g, c_1, ..., c_g$. We then condition each feature on the parent features and all features earlier in the sequence.

$$P(b_1^g, c_1^g | a_1^g) = P(b_1 | a_1^g) \times P(b_2 | a_1^g, b_1^1) \times P(b_3 | a_1^g, b_1^2) \times \cdots \times P(c_g | a_1^g, b_1^g, c_1^{g-1})$$

We can now approximate the various terms in the factorization by making independence assumptions. For instance, returning to the concrete example above, consider feature c_1, the right child nonterminal or terminal category. The following approximation should work fairly well in practice:

$$P(c_1 | a_1^g b_1^g) \approx P(c_1 | a_1, b_1)$$

That is, the category of the right child is well determined by the category of the parent and the category of the left child. Just as n-gram models approximate conditional lexical probabilities by assuming independence of words that are sufficiently distant, here we approximate conditional feature probabilities by assuming independence of features that are sufficiently unrelated. Furthermore, we can use the same kinds of backing-off techniques that are used in smoothing traditional language models to allow us to condition on relatively large contexts. In practice, a grammarian determines which features should be considered independent, and the optimal order of backoff, possibly using experiments on development test data for feedback. It might be possible to determine the optimal order of backoff automatically, a subject of future research.

Intuitively, in a PFG, features are produced one at a time. The probability of a feature being produced depends on a subset of the features in a local context of that feature. Figure 1 shows an example of this feature-at-a-time generation for the noun phrase "the man." To the right of the figure, the independence assumptions made by the grammarian are shown.

3 Formalism

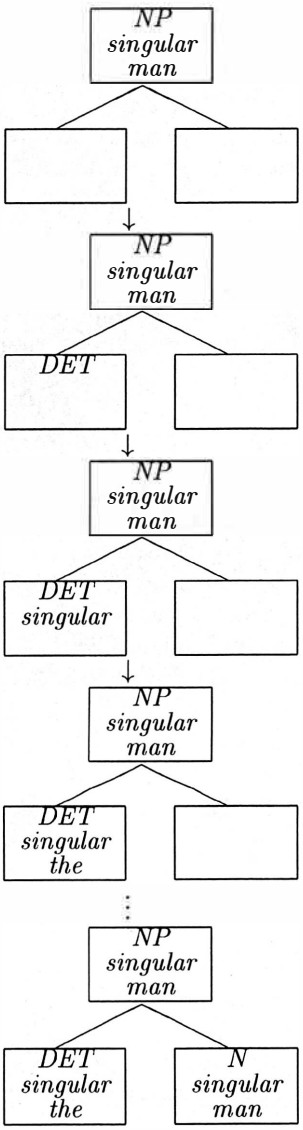
In a PCFG, the important concepts are the terminals and nonterminals, the productions involving these, and the corresponding probabilities. In a PFG, a vector of features corresponds to the terminals and nonterminals. PCFG productions correspond to PFG *events* of the form $(a_1, ..., a_g) \rightarrow (b_1, ..., b_g)(c_1, ..., c_g)$, and our PFG rule probabilities correspond to products of conditional probabilities, one for each feature that needs to be generated.

3.1 Events and EventProbs

There are two kinds of PFG events of immediate interest. The first is a *binary event*, in which a feature set a_1^g (the parent features) generates features $b_1^g c_1^g$ (the child features). Figure 1 is an example of a single such event. Binary events generate the whole tree except the start node, which is generated by a *start event*.

The probability of an event is given by an *EventProb*, which generates each new feature in turn, assigning it a conditional probability given all the known features. For instance, in a binary event, the EventProb assigns probabilities to each of the child features, given the parent features and any child features that have already been generated.

Formally, an EventProb $\mathcal{E}$ is a 3-tuple $\langle \mathcal{K}, \overline{N}, \overline{F} \rangle$, where $\mathcal{K}$ is the set of conditioning features (the Known features), $\overline{N} = N_1, N_2, ..., N_n$ is an ordered list of conditioned features (the New features), and $\overline{F} = f_1, f_2, ..., f_n$



$$P(B_1{=}DET|A_1 {=}NP,\ A_2{=}singular,\ A_3{=}man) \approx P(B_1{=}DET|A_1 {=}NP,\ A_2{=}singular)$$

$$P(B_2{=}singular|A_1 {=}NP,\ A_2{=}singular,\ A_3{=}man,\ B_1{=}DET) \approx P(B_2{=}singular|A_2{=}singular,\ B_1{=}DET)$$

$$P(B_3{=}the|A_1 {=}NP,\ A_2{=}singular,\ldots,\ B_2{=}singular) \approx P(B_3{=}the|B_1 {=}DET,\ A_3{=}man)$$

$$P(C_3{=}man|A_1 {=}NP,\ A_2{=}singular,\ldots,\ C_2{=}singular) \approx P(C_3{=}man|A_3 {=}man,\ A_1{=}NP)$$

Figure 1: Producing *the man*, one feature at a time

is a parallel list of functions. Each function $f_i(n_i, k_1, ..., k_k, n_1, n_2, ..., n_{i-1})$ returns $P(N_i = n_i | K_1 = k_1, ...K_k = k_k, N_1 = n_1, N_2 = n_2, ..., N_{i-1} = n_{i-1})$, the probability that feature $N_i = n_i$ given all the known features and all the lower indexed new features.

For a binary event, we may have $\mathcal{E}_B = \langle \{a_1, a_2, ..., a_g\}, \langle b_1, ..., b_g, c_1, ..., c_g \rangle, \overline{F}_B \rangle$; that is, the child features are conditioned on the parent features and earlier child features. For a start event we have $\mathcal{E}_S = \langle \{\}, \langle a_1, a_2, ..., a_g \rangle, \overline{F}_S \rangle$; i.e. the parent features are conditioned only on each other.

3.2 Terminal Function, Binary PFG

We need one last element: a function T from a set of g features to $\langle T, N \rangle$ which tells us whether a part of an event is terminal or nonterminal: the *terminal function*. A Binary PFG is then a quadruple $\langle g, \mathcal{E}_B, \mathcal{E}_S, T \rangle$: a number of features, a binary EventProb, a start EventProb, and a terminal function.

Of course, using binary events allows us to model n-ary branching grammars for any fixed n: we simply add additional features for terminals to be generated in the future, as well as a feature for whether or not this intermediate node is a "dummy" node (the continuation feature).[1]

4 Comparison to Previous Work

PFG bears much in common with previous work, but in each case has at least some advantages over previous formalisms.

Some other models (Charniak, 1996; Brew, 1995; Collins, 1996; Black, Lafferty, and Roukos, 1992) use probability approximations that do not sum to 1, meaning that they should not be used either for language modeling, e.g. in a speech recognition system, or as part of an integrated model such as that of Miller et al. (1996). Some models (Magerman, 1995; Collins, 1996) assign probabilities to parse trees conditioned on the strings, so that an unlikely sentence with a single parse might get probability 1, making these systems unusable for language modeling. PFGs use joint probabilites, so can be used both for language modeling and as part of an integrated model.

Furthermore, unlike all but one of the comparable systems, PFGs can compute outside probabilities, which are useful for grammar induction, some parsing algorithms (Goodman, 1996), and, as we will show, pruning (Goodman, 1997).

4.1 Bigram Lexical Dependency Parsing

Collins (1996) introduced a parser with extremely good performance. From this parser, we take many of the particular conditioning features that we will use in PFGs. As noted, this model cannot be used for language modeling. There are also some inelegancies in the need for a separate model for Base-NPs, and the treatment of punctuation as inherently different from words. The model also contains a non-statistical rule about the placement of commas. Finally, Collins' model uses memory proportional to the sum of the squares of each training sentence's length. PFGs in general use memory which is only linear.

4.2 Generative Lexicalized Parsing

Collins (1997) worked independently from us to construct a model similar to ours. In particular, Collins wished to adapt his previous parser (Collins, 1996) to a generative model. In this he succeeded. However, while we present a fairly simple and elegant formalism, which captures all information as features, Collins uses a variety of different techniques: variables (analogous to our features); a special stop category; modifications of nonterminal categories; and information computed as a function of child nodes. This lack of homogeneity fails to show the underlying structure of the model, and the ways it could be expanded.

Furthermore, our model of generation is very general. While our implementation captures head words through the particular choice of features, Collins' model explicitly generates first the head phrase, then the right children, and finally the left children. Thus, our model can be used to capture a wider variety of grammatical theories, simply by changing the choice of features.

[1] Unary branching events are in general difficult to deal with (Stolcke, 1993). We introduce an additional EventProb for unary events, and do not allow more than one unary event in a row.

Finally, there are some subtle interesting differences with respect to the distance metric. While both bigram lexical dependency parsing and generative lexicalized parsing use a distance metric, generative lexicalized parsing does not include symbols in the child node on the parent head side as part of the distance, because it is difficult to do so in that model. Our use of features allows this information to be captured.

4.3 Simple PCFGs

Charniak (1996) showed that a simple PCFG formalism in which the rules are simply "read off" of a treebank can perform very competitively. Furthermore, he showed that a simple modification, in which productions at the right side of the sentence have their probability boosted to encourage right branching structures, can improve performance even further. PFGs are a superset of PCFGs, so we can easily model the basic PCFG grammar used by Charniak, although the boosting cannot be exactly duplicated. However, we can use more principled techniques, such as a feature that captures whether a particular constituent is at the end of the sentence, and a feature for the length of the constituent. Charniak's boosting strategy means that the scores of constituents are no longer probabilities, meaning that they cannot be used with the inside-outside algorithm. Furthermore, the PFG feature-based technique is not extra-grammatical, meaning that no additional machinery needs to be added for parsing or grammar induction.

4.4 Stochastic HPSG

Brew (1995) introduced a stochastic version of HPSG. In his formalism, in some cases even if two features have been constrained to the same value by unification, the probabilities of their productions are assumed independent. The resulting probability distribution is then normalized so that probabilities sum to one. This leads to problems with grammar induction pointed out by Abney (1996). Our formalism, in contrast, explicitly models dependencies to the extent possible given data sparsity constraints.

4.5 IBM Language Modeling Group

Researchers in the IBM Language Modeling Group developed a series of successively more complicated models to integrate statistics with features.

The first model (Black, Garside, and Leech, 1993; Black, Lafferty, and Roukos, 1992) essentially tries to convert a unification grammar to a PCFG, by instantiating the values of the features. Due to data sparsity, however, not all features can be instantiated. Instead, they create a grammar where many features have been instantiated, and many have not; they call these partially instantiated features sets *mnemonics*. They then create a PCFG using the mnemonics as terminals and nonterminals. Features instantiated in a particular mnemonic are generated probabilistically, while the rest are generated through unification. Because no smoothing is done, and because features are grouped, data sparsity limits the number of features that can be generated probabilistically, whereas because we generate features one at a time and smooth, we are far less limited in the number of features we can use. Their technique of generating some features probabilistically, and the rest by unification, is somewhat inelegant; also, for the probabilities to sum to one, it requires an additional step of normalization, which they appear not to have implemented.

In their next model (Black et al., 1992; Magerman, 1994, pp. 46–56), which strongly influenced our model, five attributes are associated with each nonterminal: a syntactic category, a semantic category, a rule, and two lexical heads. The rules in this grammar are the same as the mnemonic rules used in the previous work, developed by a grammarian. These five attributes are generated one at a time, with backoff smoothing, conditioned on the parent attributes and earlier attributes. Our generation model is essentially the same as this. Notice that in this model, unlike ours, there are two kinds of features: those features captured in the mnemonics, and the five categories; the categories and mnemonic features are modeled very differently. Also, notice that a great deal of work is required by a grammarian, to develop the rules and mnemonics.

The third model Magerman:94a, extends the second model to capture more dependencies, and to remove the use of a grammarian. Each decision in this model can in principal depend on any previous decision and on any word in the sentence. Because of these potentially unbounded dependencies, there is no dynamic programming algorithm: without pruning, the time complexity of the model is exponential. One motivation for PFG was to capture similar information to this third model, while allowing dynamic programming. This third model uses

for each length l, shortest to longest
  for each start s
    for each split length t
      for each b_1^g s.t. $chart[s, s+t, b_1^g] \neq 0$
        for each c_1^g s.t. $chart[s+t, s+l, c_1^g] \neq 0$
          for each a_1 consistent with $b_1^g c_1^g$

                $\vdots$

            for each a_g consistent with $b_1^g c_1^g a_1^{g-1}$
              $chart[s, s+l, a_1^g] += \mathcal{E}_B(a_1^g \rightarrow b_1^g c_1^g)$
return $\sum_{a_1^g} \mathcal{E}_S(a_1^g) \times chart[1, n+1, a_1^g])$

Figure 2: PFG Inside Algorithm

a more complicated probability model: all probabilities are determined using decision trees; it is an area for future research to determine whether we can improve our performance by using decision trees.

4.6 Probabilistic LR Parsing with Unification Grammars

Briscoe and Carroll describe a formalism (Briscoe and Carroll, 1993; Carroll and Briscoe, 1992) similar in many ways to the first IBM model. In particular, a context-free covering grammar of a unification grammar is constructed. Some features are captured by the covering grammar, while others are modeled only through unifications. Only simple plus-one-style smoothing is done, so data sparsity is still significant. The most important difference between the work of Briscoe and Carroll (1993) and that of Black, Garside, and Leech (1993) is that Briscoe et. al. associate probabilities with the (augmented) transition matrix of an LR Parse table; this gives them more context sensitivity than Black et. al. However, the basic problems of the two approaches are the same: data sparsity; difficulty normalizing probabilities; and lack of elegance due to the union of two very different approaches.

5 Parsing

The parsing algorithm we use is a simple variation on probabilistic versions of the CKY algorithm for PCFGs, using feature vectors instead of nonterminals (Baker, 1979; Lari and Young, 1990). The parser computes inside probabilities (the sum of probabilities of all parses, i.e. the probability of the sentence) and Viterbi probabilities (the probability of the best parse), and, optionally, outside probabilities. In Figure 2 we give the inside algorithm for PFGs. Notice that the algorithm requires time $O(n^3)$ in sentence length, but is potentially exponential in the number of children, since there is one loop for each parent feature, a_1 through a_g.

When parsing a PCFG, it is a simple matter to find for every right and left child what the possible parents are. On the other hand, for a PFG, there are some subtleties. We must loop over every possible value for each feature. At first, this sounds overwhelming, since it requires guessing a huge number of feature sets, leading to a run time exponential in the number of features; in practice, most values of most features will have zero probabilities, and we can avoid considering these. For instance, features such as the length of a constituent take a single value per cell. Many other features take on very few values, given the children. For example, we arrange our parse trees so that the head word of each constituent is dominated by one of its two children. This means that we need consider only two values for this feature for each pair of children. The single most time consuming feature is the Name feature, which corresponds to the terminals and non-terminals of a PCFG. For efficiency, we keep a list of the parent/left-child/right-child name triples which have non-zero probabilities, allowing us to hypothesize only the possible values for this feature given the children. Careful choice of features helps keep parse times reasonable.

5.1 Pruning

We use two pruning methods to speed parsing. The first is a simple beam-search method, inspired by techniques used by Collins (1996) and Charniak (1996), and described in detail by Goodman (1997). Within each cell in the parse chart, we multiply each entry's inside probability by the prior probability of the parent features of that entry, using a special EventProb, $\mathcal{E}_P$. We then remove those entries whose combined probability is too much lower than the best entry of the cell.

In speech recognition, multiple-pass recognizers (Zavaliagkos et al., 1994) have been very successful. We can use an analogous technique, multiple-pass parsing (Goodman, 1997) with either PCFGs or PFGs. We use a simple, fast grammar for the first pass, which approximates the later pass. We then remove any events whose combined inside-outside product is too low: essentially those events that are unlikely given the complete sentence. The first pass is fast, and does not slow things down much, but allows us to speed up the second pass significantly. The technique is particularly natural for PFGs, since for the first pass, we can simply use a grammar with a superset of the features from the previous pass. Actually, the features we used in our first pass were Name, Continuation, and two new features especially suitable for a fast first pass, the length of the constituent and the terminal symbol following the constituent. Since these two features are uniquely determined by the chart cell of the constituent, they are especially suitable for use in a first pass, since they provide useful information without increasing the number of elements in the chart. However, when used in our second pass, these features did not help performance, presumably because they captured information similar to that captured by other features. Multiple-pass techniques have dramatically sped up PFG parsing.

6 Experimental Results

The PFG formalism is an extremely general one that has the capability to model a wide variety of phenomena. Still, it is useful to apply the formalism to actual data, as a proof of concept.

We used two PFGs, one which used the head word feature, and one otherwise identical grammar with no word based features, only POS tags. The grammars had the features shown in Figure 4. A sample parse tree with these features is given in Figure 3.

The feature D_L is a 3-tuple, indicating the number of punctuation characters, verbs, and words to the left of a constituent's head word. To avoid data sparsity, we do not count higher than 2 punctuation characters, 1 verb, or 4 words. So, a value of $\mathrm{D}_L = 014$ indicates that a constituent has no punctuation, at least one verb, and 4 or more words to the left of its head word. D_R is a similar feature for the right side. Finally, D_B gives the numbers between the constituent's head word, and the head word of its other child. Words, verbs, and punctuation are counted as being to the left of themselves: that is, a terminal verb has one verb and one word on its left.

Notice that the continuation of the lower NP in Figure 3 is R1, indicating that it inherits its child from the right, with the 1 indicating that it is a "dummy" node to be left out of the final tree.

The probability functions we used were similar to those of Section 2. There were three important differences. The first is that in some cases, we can compute a probability exactly. For instance, if we know that the head word of a parent is "man" and that the parent got its head word from its right child, then we know that with probability 1, the head word of the right child is "man." In cases where we can compute a probability exactly, we do so. Second, we smoothed slightly differently. In particular, when smoothing a probability estimate of the form

$$p(a|bc) \approx \lambda \frac{C(abc)}{C(bc)} + (1-\lambda)p(a|b)$$

we set $\lambda = \frac{C(bc)}{k+C(bc)}$, using a separate k for each probability distribution. Finally, we did additional smoothing for words, adding counts for the unknown word.

It would take quite a bit of space to give the table that shows for each feature the order of backoff for that feature. We instead discuss a single example, the order of backoff for the 2_R feature, the category of the second child of the right feature set. The least relevant features are first: $W_R, P_R, C_L, N_L, C_P, N_P, 1_R, H_R, C_R, N_R$. We back off from the head word feature first, because, although relevant, this feature creates significant data sparsity. Notice that in general, features from the same feature set, in this case the right features, are kept longest; parent features are next most relevant; and sibling features are considered least relevant. We leave out

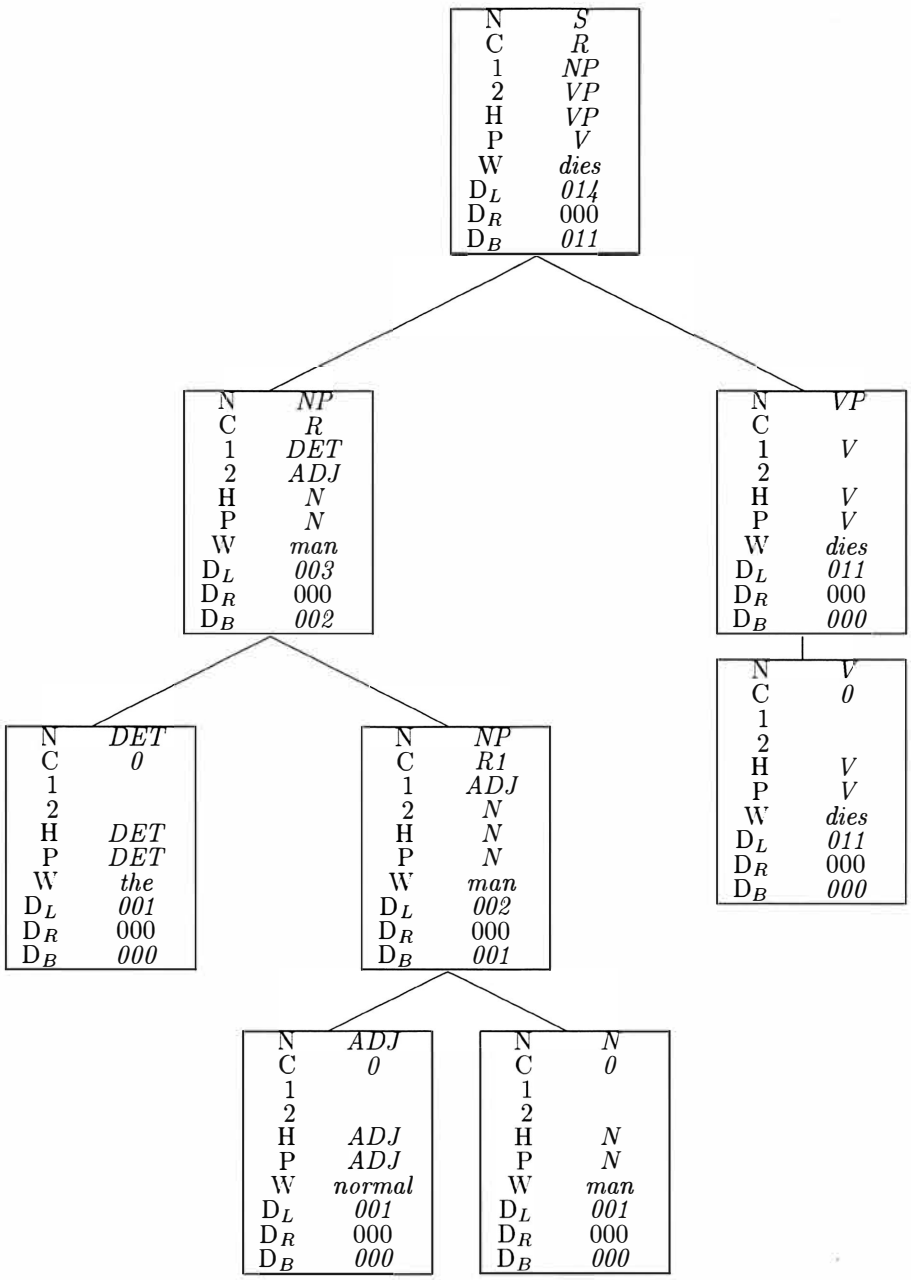


Figure 3: Example tree with features: The normal man dies

N **Name** Corresponds to the terminals and nonterminals of a PCFG.

C **Continuation** Tells whether we are generating modifiers to the right or the left, and whether it is time to generate the head node.

1 **Child1** Name of first child to be generated.

2 **Child2** Name of second child to be generated. In combination with Child1, this allows us to simulate a second order Markov process on nonterminal sequences.

H **Head name** Name of the head category.

P **Head pos** Part of speech of head word.

W **Head word** Actual head word. Not used in POS only model.

$\mathbf{D}_L$ **Δ left** Count of punctuation, verbs, words to left of head.

$\mathbf{D}_R$ **Δ right** Counts to right of head.

$\mathbf{D}_B$ **Δ between** Counts between parent's and child's heads.

Figure 4: Features Used in Experiments

entirely features that are unlikely to be relevant and that would cause data sparsity, such as W_L, the head word of the left sibling.

6.1 Results

We used the same machine-labeled data as Collins (1996; 1997): TreeBank II sections 2-21 for training, section 23 for test, section 00 for development, using all sentences of 40 words or less.[2] We also used the same scoring method (replicating even a minor bug for the sake of comparison).

Our results are the best we know of from POS tags alone, and, with the head word feature, fall between the results of Collins and Magerman, as given by Collins (1997).

Model	Labeled Recall	Labeled Precision	Crossing Brackets	0 Crossing Brackets	$\leq$ 2 Crossing Brackets
PFG Words	84.8%	85.3%	1.21	57.6%	81.4%
PFG POS only	81.0%	82.2%	1.47	49.8%	77.7%
Collins 97 best	88.1%	88.6%	0.91	66.5%	86.9%
Collins 96 best	85.8%	86.3%	1.14	59.9%	83.6%
Collins 96 POS only	76.1%	76.6%	2.26		
Magerman	84.6%	84.9%	1.26	56.6%	81.4%

7 Conclusions and Future Work

While the empirical performance of probabilistic feature grammars is very encouraging, we think there is far more potential. First, for grammarians wishing to integrate statistics into more conventional models, the features of PFG are a very useful tool, corresponding to the features of DCG, LFG, HPSG, and similar formalisms. TreeBank II is annotated with many semantic features, currently unused in all but the simplest way by all systems; it should be easy to integrate these features into a PFG.

PFG has other benefits that we would like to explore, including the possibility of its use as a language model, for applications such as speech recognition. Furthermore, the dynamic programming used in the model is amenable to efficient rescoring of lattices output by speech recognizers.

[2]We are grateful to Michael Collins and Adwait Ratnaparkhi for supplying us with the part-of-speech tags.

Another benefit of PFG is that both inside and outside probabilities can be computed, making it possible to reestimate PFG parameters. We would like to try experiments using PFGs and the inside/outside algorithm to estimate parameters from unannotated text.

While the generality and elegance of the PFG model makes these and many other experiments possible, we are also encouraged by the very good experimental results. Wordless model performance is excellent, and the more recent models with words are comparable to the state of the art.

References

Abney, Steve. 1996. Stochastic attribute-value grammars. Available as cmp-lg/9610003, October.

Baker, J.K. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA, June.

Black, Ezra, George Garside, and Geoffrey Leech. 1993. *Statistically-Driven Computer Grammars of English: the IBM/Lancaster Approach*, volume 8 of *Language and Computers: Studies in Practical Linguistics*. Rodopi, Amsterdam.

Black, Ezra, Frederick Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. 1992. Towards history-based grammars: Using printer models for probabilistic parsing. In *Proceedings of the February 1992 DARPA Speech and Natural Language Workshop*.

Black, Ezra, John Lafferty, and Salim Roukos. 1992. Development and evaluation of a broad-coverage probabilistic grammar of english-language computer manuals. In *Proceedings of the 30th Annual Meeting of the ACL*, pages 185–192.

Brew, Chris. 1995. Stochastic HPSG. In *Proceedings of the Seventh Conference of the European Chapter of the ACL*, pages 83–89, Dublin, Ireland, March.

Briscoe, Ted and John Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19:25–59.

Carroll, John and Ted 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*, pages 33–38, Cambridge, MA.

Charniak, Eugene. 1996. Tree–bank grammars. Technical Report CS-96-02, Department of Computer Science, Brown University. Available from `ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-02.ps.Z` .

Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 184–191, Santa Cruz, CA, June.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the ACL*, pages 16–23, Madrid, Spain.

Eisele, Andreas. 1994. Towards probabilistic extensions of constraint-based grammars. DYANA-2 Deliverable R1.2.B, September. Available from `ftp://ftp.ims.uni-stuttgart.de/pub/papers/DYANA2/R1.2.B`.

Goodman, Joshua. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 177–183, Santa Cruz, CA, June.

Goodman, Joshua. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*.

Lari, K. and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Magerman, David. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University University, February.

Magerman, David. 1995. Statistical decision–models for parsing. In *Proceedings of the 33rd Annual Meeting of the ACL*, pages 276–283, Cambridge, MA.

Miller, Scott, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 55–61, Santa Cruz, CA, June.

Stolcke, Andreas. 1993. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Technical Report TR-93-065, International Computer Science Institute, Berkeley, CA.

Zavaliagkos, G., T. Anastasakos, G. Chou, C. Lapre, F. Kubala, J. Makhoul, L. Nguyen, R. Schwartz, and Y. Zhao. 1994. Improved search, acoustic and language modeling in the BBN Byblos large vocabulary CSR system. In *Proceedings of the ARPA Workshop on Spoken Language Technology*, pages 81–88, Plainsboro, New Jersey.

Message-Passing Protocols for Real-World Parsing — An Object-Oriented Model and its Preliminary Evaluation

Udo Hahn and Peter Neuhaus and Norbert Bröker

Computational Linguistics Lab

Freiburg University, Werthmannplatz, D-79085 Freiburg, Germany

{hahn,neuhaus,nobi}@coling.uni-freiburg.de

http://www.coling.uni-freiburg.de

Abstract

We argue for a performance-based design of natural language grammars and their associated parsers in order to meet the constraints imposed by real-world NLP. Our approach incorporates declarative and procedural knowledge about language and language use within an object-oriented specification framework. We discuss several message-passing protocols for parsing and provide reasons for sacrificing completeness of the parse in favor of efficiency based on a preliminary empirical evaluation.

1 Introduction

Over the past decades the design of natural language grammars and their parsers was almost entirely based on *competence* considerations (Chomsky, 1965). These hailed pure declarativism (Shieber, 1986) and banned procedural aspects of natural language use out of the domain of language theory proper. The major premises of that approach were to consider sentences as the primary object of linguistic investigation, to focus on syntactic descriptions, and to rely upon perfectly well-formed utterances for which complete grammar specifications of arbitrary depth and sophistication were available. In fact, promising efficiency results can be achieved for parsers operating under corresponding optimal laboratory conditions. Considering, however, the requirements of natural language *understanding*, i.e., the integration of syntax, semantics, and pragmatics, and taking *ill-formed* input or *incomplete* knowledge into consideration, their processing costs either tend to increase at excessive rates or linguistic processing even fails completely.

As a consequence, the challenge to meet the specific requirements imposed by real-world texts has led many researchers in the NLP community to re-engineer competence grammars and their parsers and to provide various add-ons in terms of constraints (Uszkoreit, 1991), heuristics (Huyck & Lytinen, 1993), statistics-based weights (Charniak, 1993), etc. In contradistinction to these approaches, our principal goal has been to incorporate performance conditions already in the design of natural language grammars, yielding so-called *performance grammars*. Thus, not only declarative knowledge (as is common for competence grammars), but also *procedural* knowledge (about control and parsing strategies, resource limitations, etc.) has to be taken into consideration at the *grammar specification* level proper. This is achieved by providing self-contained description primitives for the expression of procedural knowledge. We have taken care to transparently separate declarative (structure-oriented) from procedural (process-oriented) knowledge pieces. Hence, we have chosen a formally homogeneous, highly modularized object-oriented grammar specification framework, *viz.* the actor model of computation which is based on concurrently active objects that communicate by asynchronous message passing (Agha, 1990).

The parser whose design is based on these performance considerations forms part of a text knowledge acquisition system, operational in two domains, *viz.* the processing of test reports from the information technology field (Hahn & Schnattinger, 1997) and medical reports (Hahn et al., 1996b). The analysis of texts (instead of isolated sentences) requires, first of all, the consideration of textual phenomena by a dedicated *text grammar*. Second, text understanding is based on drawing inferences by which text propositions are integrated on the fly into the text knowledge base with reference to a canonical representation of the underlying *domain knowledge*. This way, grammatical (language-specific) and conceptual (domain-specific) knowledge are closely coupled. Third, text understanding in humans occurs immediately and at least within specific processing cycles in parallel (Thibadeau et al., 1982). These processing strategies we find in human language processing are taken as hints how the complexity of natural language understanding can reasonably be overcome

by machines. Thus, text parsing devices should operate *incrementally* and *concurrently*. In addition, the consideration of *real-world* texts forces us to supply mechanisms which allow for the *robust* processing of extra- and ungrammatical input. We take an approach where — in the light of abundant specification gaps at the grammar and domain representation level — the degree of underspecification of the knowledge sources or the impact of grammar violations directly corresponds to a lessening of the precision and depth of text knowledge representations, thus aiming at a sophisticated *fail-soft* model of *partial* text parsing.

2 The Grammar

The performance grammar we consider contains fully *lexicalized* grammar specifications (Hahn et al., 1994). Each lexical item is subject to configurational constraints on word classes and morphological features as well as conditions on word order and conceptual compatibility a head places on possible modifiers. Grammatical conditions of these types are combined in terms of *valency* constraints (at the phrasal and clausal level) as well as *textuality* constraints (at the text level of consideration), which concrete dependency structures and local as well as global coherence relations must satisfy. The compatibility of grammatical features including order constraints (encapsulated by methods we refer to as SYNTAXCHECK) is computed by a unification mechanism, while the evaluation of semantic and conceptual constraints (we here refer to as CONCEPTCHECK) relies upon the terminological and rule-based construction of a consistent conceptual representation. Thus, while the dependency relations represent the linguistic structure of the input, the conceptual relations yield the targeted representation of the text content (for an illustration, cf. Fig. 7).

In order to structure the underlying lexicon, *inheritance* mechanisms are used. Lexical specifications are organized along the grammar hierarchy at various abstraction levels, e.g., with respect to generalizations on word classes. Lexicalization of this form already yields a fine-grained decomposition of declarative grammar knowledge. It lacks, however, an equivalent description at the procedural level. We therefore provide lexicalized communication primitives to allow for heterogeneous and local forms of interaction among lexical items.

Following the arguments brought forward, e.g., by Jackendoff (1990) and Allen (1993), there is no distinction at the representational level between semantic and conceptual interpretations of texts. Hence, semantic and domain knowledge specifications are based on a common hybrid classification-based knowledge representation language (for a survey, cf. Woods & Schmolze (1992)). Ambiguities which result in interpretation variants are managed by a context mechanism of the underlying knowledge base system.

Robustness at the grammar level is achieved by several means. Dependency grammars describe binary, functional relations between words rather than contiguous constituent structures. Thus, ill-formed input often has an (incomplete) analysis in our grammar. Furthermore, it is possible to specify lexical items at different levels of syntactic or semantic granularity such that the specificity of constraints may vary. The main burden of robustness, however, is assigned to a dedicated message-passing protocol we will discuss in the next section.

3 The Parser

Viewed from a parsing perspective, we represent lexical items as *word actors* which are acquainted with other actors representing the heads or modifiers in the current utterance. A specialized actor type, the *phrase actor*, groups word actors which are connected by dependency relations and encapsulates administrative information about each phrase. A message does not have to be sent directly to a specific word actor, but will be sent to the mediating phrase actor which forwards it to an appropriate word actor. Furthermore, the phrase actor holds the communication channel to the corresponding interpretation context in the domain knowledge base system. A *container actor* encapsulates several phrase actors that constitute alternative analyses for the *same* part of the input text (i.e., structural ambiguities). Container actors play a central role in controlling the parsing process, because they keep information about the *textually* related (*preceding*) container actors holding the left context and the *chronologically* related (*previous*) container actors holding a part of the head-oriented parse history.

Basic Parsing Protocol (incl. Ambiguity Handling). We use a graphical description language to sketch the message-passing protocol for establishing dependency relations as depicted in Fig. 1 (the phrase actor's active head is visualized by $\oplus$). A searchHeadFor message (and *vice versa* a searchModifierFor message if searchHeadFor fails) is sent to the textually preceding container actor (precedence relations are depicted by bold dashed lines), which simultaneously directs this message to its encapsulated phrase actors. At the level of a single phrase actor, the distribution of the searchHeadFor

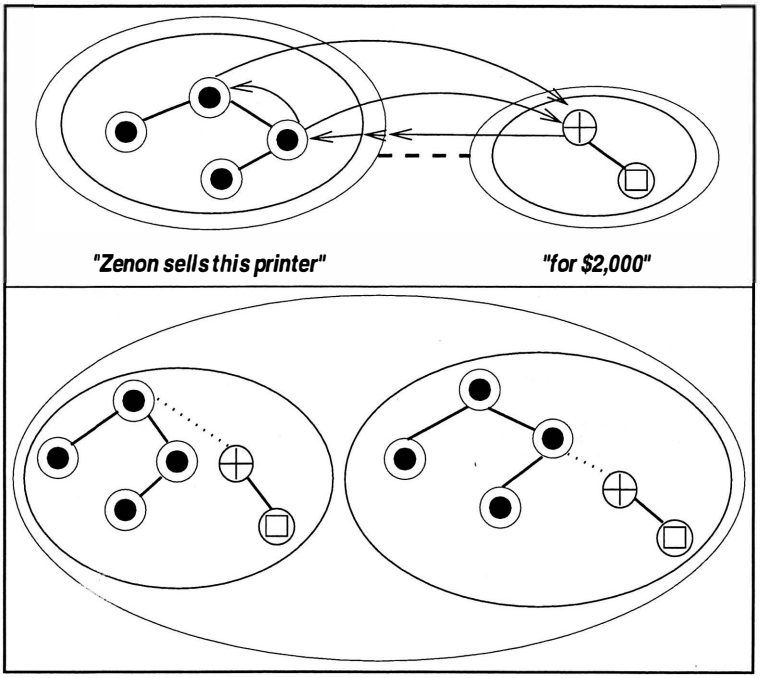


Figure 1: Basic Mode (incl. Structural Ambiguities)

message occurs for all word actors at the "right rim" of the dependency tree (depicted by ◉). A word actor that receives a searchHeadFor message from another word actor concurrently forwards this message to its head (if any) and tests in its local site whether a dependency relation can be established by checking its corresponding valency constraints (applying SYNTAXCHECK and CONCEPTCHECK). In case of success, a headFound message is returned, the sender and the receiver are copied (to enable alternative attachments in the concurrent system, i.e., no destructive operations are carried out), and a dependency relation, indicated by a dotted line, is established between those copies which join into a phrasal relationship (for a more detailed description of the underlying protocols, cf. Neuhaus & Hahn (1996)). For illustration purposes, consider the analysis of a phrase like *"Zenon sells this printer"* covering the content of the phrase actor which textually precedes the phrase actor holding the dependency structure for *"for $2,000"*. The latter actor requests its attachment as a modifier of some head. The resultant new container actor (encapsulating the dependency analysis for *"Zenon sells this printer for $2,000"* in two phrase actors) is, at the same time, the historical successor of the phrase actor covering the analysis for *"Zenon sells this printer"*.

The structural ambiguity inherent in the example is easily accounted for by this scheme. The criterion for a structural ambiguity to emerge is the reception of at least two positive replies to a single searchHeadFor (or searchModifierFor) message by the initiator. The basic protocol already provides for the concurrent copying and feature updates. In the example from Fig. 1, two alternative readings are parsed, one phrase actor holding the attachment to the verb (*"sells"*), the other holding that to the noun (*"printer"*). The crucial point about these ambiguous syntactic structures is that they have conceptually different representations in the domain knowledge base. In the case of Fig. 1 verb attachment leads to the instantiation of the PRICE slot of the corresponding SELL action, while the noun attachment leads to the corresponding instantiation of the PRICE slot of PRINTER.

Robustness: Skipping Protocol. Skipping for robustnes purposes is a well known mechanism though limited in its reach (Lavie & Tomita, 1993). But in free word-order languages as German skipping is even vital for the analysis of entirely well-formed structures, e.g., those involving scrambling or discontinuous constructions. For brevity, we will base the following explanation on the robustness issue and refer the interested reader to Neuhaus & Bröker (1997). The incompleteness of linguistic and conceptual specifications is ubiquitous in real-world applications and, therefore, requires mechanisms for a fail-soft parsing behavior. Fig. 2 illustrates a typical "skipping" scenario. The currently active container addresses a searchHeadFor (or searchModifierFor) message to its textually immediately preceding container actor. If *both* types of messages fail, the immediately preceding container of the active container forwards these messages — in the canonical order — to its immediately preceding container actor. If any of these two message types succeeds after that mediation, a corresponding (discontinuous) dependency structure is built up. Furthermore, the skipped container is moved

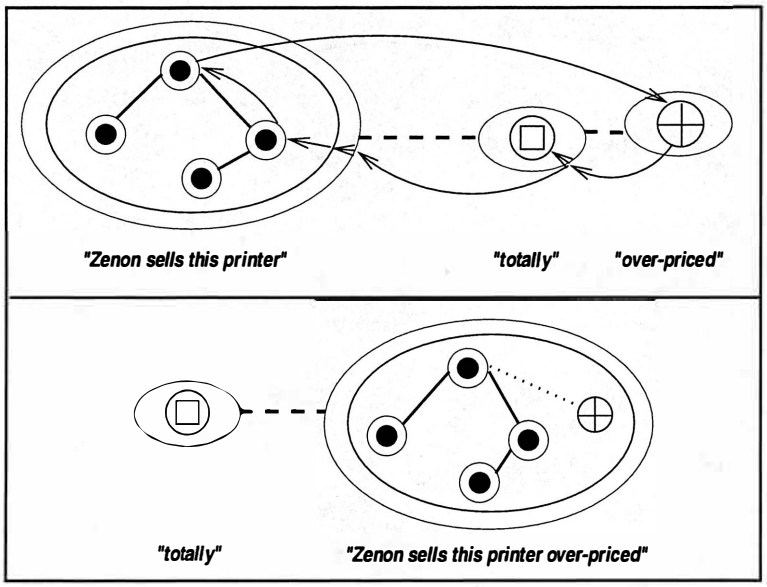


Figure 2: Skipping Mode

to the left of the newly built container actor. Note that this behavior results in the reordering of the lexical items analyzed so far such that skipped containers are continuously moved to the left. As an example, consider the phrase *"Zenon sells this printer"* and let us further assume *"totally"* to be a grammatically unknown item which is followed by the occurrence of *"over-priced"* as the active container. Skipping yields a structural analysis for *"Zenon sells this printer over-priced"*, while *"totally"* is simply discarded from further consideration. This mode requires an extension of the basic protocol in that searchHeadFor and searchModifierFor messages are forwarded across non-contiguous parts of the analysis when these messages do not yield a positive result for the requesting actor relative to the *immediately* adjacent container actor.

Backtracking Protocol. Backtracking to which we still adhere in our model of constrained concurrency accounts for a state of the analysis where none of the aforementioned protocols have terminated successfully in *any textually* preceding container, i.e., several repeated skippings have occurred, until a linguistically plausible barrier is encountered. In this case, backtracking takes place and messages are now directed to *historically* previous containers, i.e., to containers holding fragments of the parse history. This is realized in terms of a protocol extension by which searchHeadFor (or searchModifierFor) messages, first, are reissued to the *textually* immediately preceding container actor which then forwards these messages to its *historically* previous container actor. This actor contains the head-centered results of the analysis of the left context prior to the structural extension held by the historical successor.[1] Attachments for heads or modifiers are now checked referring to the historically preceding container and the active container as depicted in Fig. 3a.

If the valency constraints are met, a new phrase actor is formed (cf. Fig. 3b) necessarily yielding a discontinuous analysis. A slightly modified protocol implements reanalysis, where the skipped items send reSearchHeadFor (or reSearchModifierFor) messages to the new phrase actor, which forwards them directly to those word actors where the discontinuity occurs. As an example, consider the fragment *"the customer bought the silver"* (with *"silver"* in the noun reading, cf. Fig. 3a). This yields a perfect analysis which, however, cannot be further augmented when the word actor *"notebook"* asks for a possible attachment.[2] Two intervening steps of reanalysis (cf. Fig. 3b and 3c) yield the final structural configuration depicted in Fig. 3d.

Prediction Protocol. The depth-first approach of the parser brings about a decision problem whenever a phrase cannot be integrated into (one of) the left-context analyses. Either, the left context and the current phrase are to be related by a word not yet read from the input and, thus, the analysis should proceed without an attachment.[3] Or, depth-first analysis was misguided and a backtrack should be invoked to revise a former decision with respect to attachment information available by now.

[1] Any container which holds the modifying part of the structural analysis of the historical successor (in Fig. 3a this relates to "the" and "silver") is deleted. Hence, this deletion renders the parser incomplete in spite of backtracking.

[2] Being an arc-eager parsing system, a *possible* dependency relation will always be established. Hence, the adjective reading of *"silver"* will not be considered in the initial analysis.

[3] This effect occurs particularly often for verb-final languages such as German.

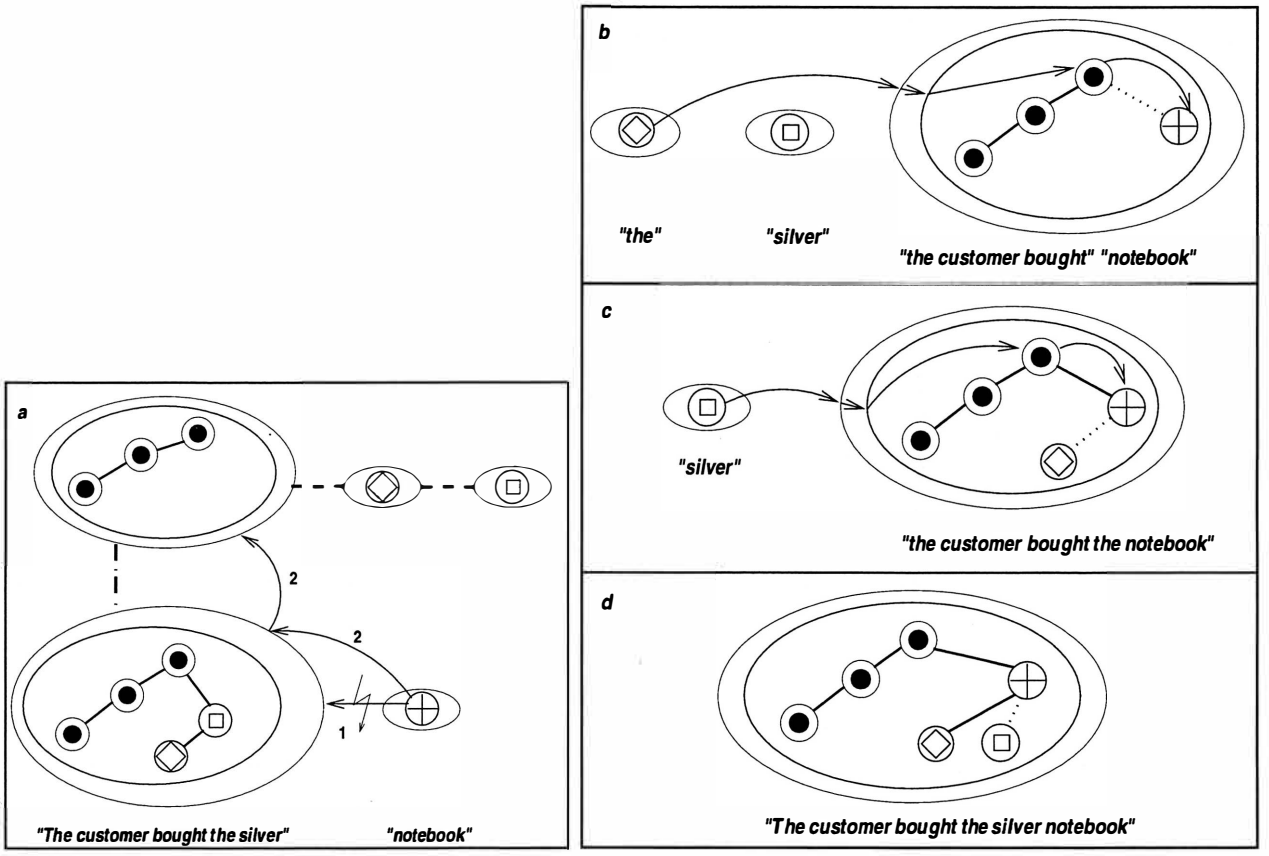


Figure 3: Backtracking Mode

Prediction can be used to carry out a more informed selection between these alternatives. Words not yet read, but required for a complete analysis, can be derived from the input analyzed so far, either top-down (predicting a modifier) or bottom-up (predicting a head). Both types of prediction are common in phrase-structure based parsers, e.g. Earley-style top-down prediction (Earley, 1970) or left-corner strategies with bottom-up prediction (Kay, 1986). Since dependency grammars, in general, do not employ non-lexical categories which can be predicted, so-called *virtual words* are constructed by the parser, which are later to be instantiated with lexical content as it becomes available when the analysis proceeds.

Whenever an active phrase cannot attach itself to the left context, the head of this phrase may predict a virtual word as tentative head of a new phrase under which it is subordinated. The virtual word is specified with respect to its word class, morphosyntactic features, and order restrictions, but is left vacuous with respect to its lexeme and semantic specification. In this way, a determiner immediately constructs an NP (cf. Fig. 4a), which can be attached to the left context and may incrementally incorporate additional attributive adjectives until the head noun is found (cf. Fig. 4b).[4] The virtual word processes a searchPredictionFor protocol initiated by the next lexical item. The virtual word and this lexical item are *merged* iff the lexical item is at least as specific as the virtual word (concerning word class and features) and it is able to govern all modifiers of the virtual word (cf. Fig. 4c).

This last criterion may not always be met, although the prediction, in general, is correct. Consider the case of German verb-final subclauses. A top-down prediction of the complementizer constructs a virtual finite verb (designated by Ⓟ), which may govern any number of NPs in the subclause (cf. Fig. 5a). If the verbal complex, however, consists of an infinite full verb preceding a finite auxiliary, the modifiers of the virtual verb must be distributed over two lexical items.[5] An extension of the prediction protocol accounts for this case: A virtual word can be split if it may govern the lexical item and some modifiers can be transferred to the lexical item. In this case, the lexical item is subordinated to a newly

[4]This procedure implements the notion of *mother node constructing categories* proposed by Hawkins (1994), which are a generalization of the notion *head* to all words which unambiguously determine their head. The linguistic puzzle about NP vs. DP is thus solved. In contrast to Hawkins, we also allow for multiple predictions.

[5]We here assume the finite auxiliary to govern the subject (enforcing agreement), while the remaining complements are governed by the infinite full verb.

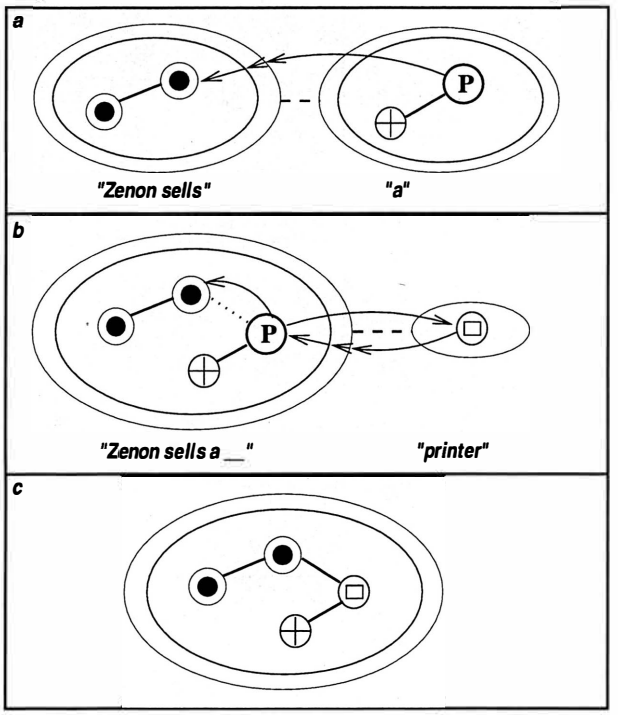


Figure 4: Predicting and merging a noun

created virtual word (indicated by Ⓟ in Fig. 5b) governing the remaining modifiers. Since order restrictions are available for virtual words, even non-projectivities can be accounted for by this scheme (cf. Fig. 5b).[6]

Although prediction allows parsing to proceed incrementally and more informed (to the potential benefit of increased efficiency), it engenders possible drawbacks: In underspecified contexts, a lot of false predictions may arise and may dramatically increase the number of ambiguous analyses. Furthermore, the introduction of additional operations (prediction, split, and merge) increases the search space of the parser. Part of the first problem is addressed by our extensive usage of the word class hierarchy. If a set of predictions contains all subclasses of some word class W, only one virtual word of class W is created.

Text Phenomena. A particularly interesting feature of the performance grammar we propose is its capability to seamlessly integrate the sentence and text level of linguistic analysis. We have already alluded to the notoriously intricate interactions between syntactic criteria and semantic constraints at the phrasal and clausal level. The interaction is even more necessary at the text level of analysis as semantic interpretations have an immediate update effect on the discourse representation structures to which text analysis procedures refer. Their status and validity directly influence subsequent analyses at the sentence level, e.g., by supplying proper referents for semantic checks when establishing new dependency relations. In addition, lacking recognition and referential resolution of textual forms of pronominal or nominal anaphora (Strube & Hahn, 1995), textual ellipses (Hahn et al., 1996a) and metonymies (Markert & Hahn, 1997) leads to invalid or incohesive text knowledge representation structures. These not only yield invalid parsing results (at the methodological level) but also preclude proper text knowledge acquisition (at the level of system functionality). Hence, we stress the neat integration of syntactic and semantic checks during the parsing process at the sentence and the text level. We now turn to text grammar specifications concerned with anaphora resolution and their realization by a special text parsing protocol.

The protocol which accounts for local text coherence analysis makes use of a special actor, the *centering actor*, which keeps a backward-looking center (C_b) and a preferentially ordered list of forward-looking centers (C_f) of the previous utterance (we here assume a functional approach (Strube & Hahn, 1996) to the well-known centering model originating from Grosz et al. (1995)). These lists are accessed to establish proper referential links between an anaphoric expression in the current utterance and the valid antecedent in the preceding ones. Nominal anaphora (cf. the occurrences of *"the company"* and *"these printers"* in Fig. 6) trigger a special searchNomAntecedent message. When it reaches the C_f list, possible antecedents are accessed in the given preference order. If an antecedent and the anaphor fulfill certain

[6]Non-projectivities often arise, e.g. due to the fronting of a non-subject relative pronoun. As indicated by the dashed line in Fig. 5b and 5c, we employ additional projective relations to restrain ordering for discontinuities.

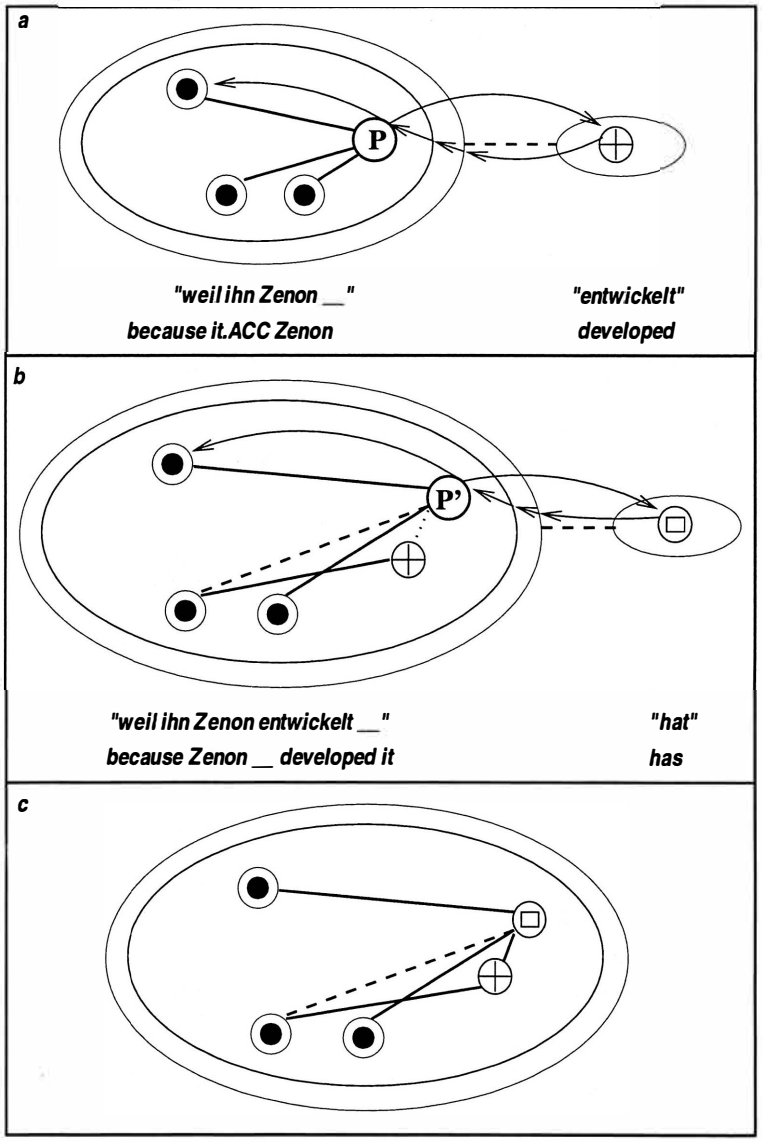


Figure 5: Predicting and splitting a verb

grammatical and conceptual compatibility constraints, an antecedentFound message is issued to the anaphor, and finally, the discourse referent of the antecedent replaces the one in the original anaphoric expression in order to establish local coherence. In case of successful anaphor resolution an anaphorSucceed message is sent from the resolved anaphor to the centering actor in order to remove the determined antecedent from the C_f list (this avoids illegal follow-up references). The effects of these changes at the level of text knowledge structures are depicted in Fig. 7, which contains the terminological representation structures for the sentences in Fig. 6.

4 Preliminary Evaluation

Any text understanding system which is intended to meet the requirements discussed in Section 1 faces severe performance problems. Given a set of strong heuristics, a computationally complete depth-first parsing strategy usually will increase the parsing efficiency in the *average case*, i.e., for input that is in accordance with the parser's preferences. For the rest of the input further processing is necessary. Thus, the *worst case* for a depth-first strategy applies to input which cannot be assigned any analysis at all (i.e., in cases of extra- or ungrammaticality). Such a failure scenario leads to an exhaustive search of the parse space. Unfortunately, under realistic conditions of real-world text input these cases occur

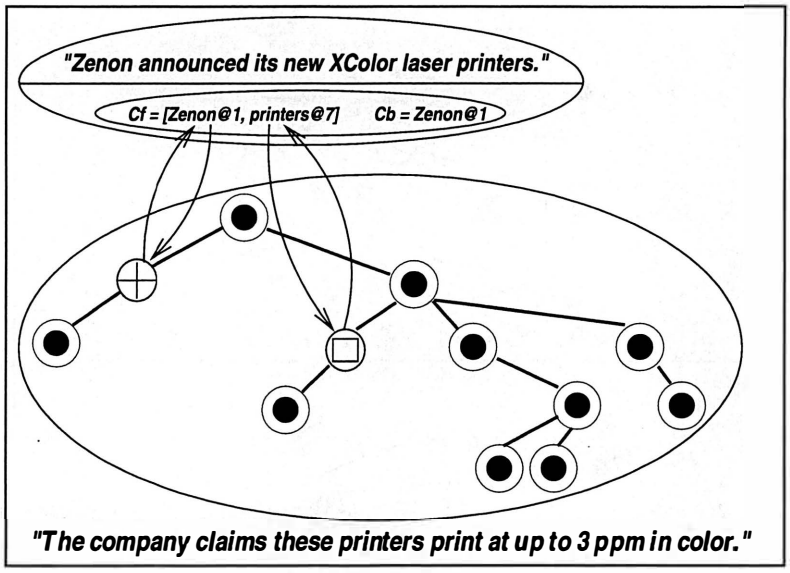


Figure 6: Anaphora Resolution Mode

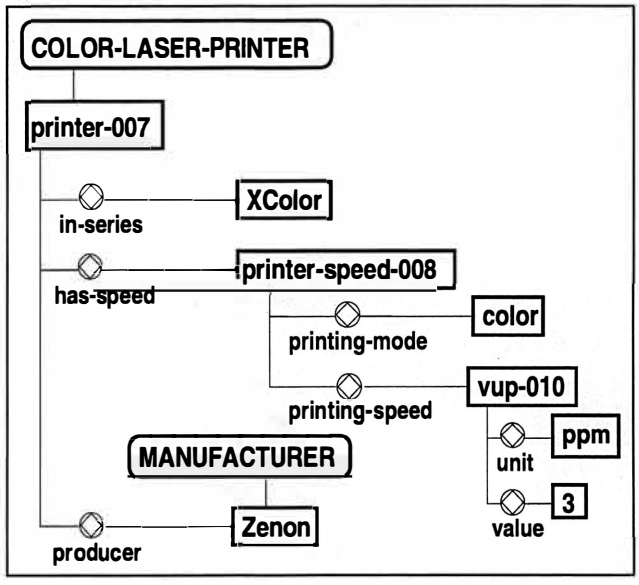


Figure 7: Sample Output of Text Parsing

quite often. Hence, by using a computationally complete depth-first strategy one merely would trade space complexity for time complexity.

To maintain the potential for efficiency of depth-first operation it is necessary to prevent the parser from exhaustive backtracking. In our approach this is achieved by two means. First, by restricting memoization of attachment candidates for backtracking (e.g., by retaining only the head portion of a newly built phrase, cf. footnote 1). Second, by restricting the accessibility of attachment candidates for backtracking (e.g., by bounding the forwarding of backtracking messages to linguistically plausible barriers such as punctuation actors). In effect, these restrictions render the parser *computationally incomplete*, since some input, though covered by the grammar specification, will not be correctly analyzed.

4.1 Performance Aspects

The stipulated efficiency gain that results from deciding against completeness is empirically substantiated by a comparison of our PARSETALK system, henceforth designated as *PT*, with a standard chart parser,[7] abbreviated as *CP*. As the CP does not employ any robustness mechanisms (one might,e.g., incorporate those proposed by Mellish (1989)) the current comparison had to be restricted to entirely grammatical sentences. We also do not account for prediction mechanisms the necessity of which we argued for in Section 2. For the time being, an evaluation of the prediction mechanisms is still under way. Actually, the current comparison of the two parsers is based on a set of 41 sentences from our corpus (articles from computer magazines) that do not exhibit the type of structure requiring prediction (cf. Fig. 5 and the example therein). For 40 of the test sentences[8] the CP finds all correct analyses but also those over-generated by the grammar. In combination, this leads to a ratio of 2.3 of found analyses to correct ones. The PT system (over-generating at a ratio of only 1.6) finds 36 correct analyses, i.e., 90% of the analyses covered by the grammar (cf. the remark on 'near misses' in Section 4.2). Our preliminary evaluation study rests on two measurements, viz. one considering concrete run-time data, the other comparing the number of method calls.

number of samples	speed-up factor min–max	average
25	1.1–4.2	2.8
10	5.1–8.9	6.9
6	10.9–54.8	45.2

Table 1: Ratio of run times of the CP and the PT system, chunked by speed-up.

The loss in completeness is compensated by a reduction in processing costs on the order of one magnitude on the average. Since both systems use the identical dependency grammar and knowledge representation the implementation of which rests on identical Smalltalk and LOOM/Common Lisp code, a run time comparison seems reasonable to some degree. For the test set the PT parser turned out to be about 17 times faster than the CP parser (per sentence speed-up averaged at over 10). Table 1 gives an overview of the speed-up distribution. 25 short to medium long sentences were processed with a speed-up in a range from 1.1 to 4.2 times faster than the chart parser averaging at 2.8. Another 10 longer and more complex sentences show the effects of complexity reduction even more clearly, averaging at a speed-up of 6.9 (of a range from 5.1 to 8.9). One of the remaining 6 very complex sentences is discussed below.

Accordingly to these factors, the PT system spent nearly two hours (on a SPARCstation 10 with 64 MB of main memory) processing the entire test set, while the CP parser took more than 24 hours. The exorbitant run times are largely a result of the (incremental) conceptual interpretation, though these computations are carried out by the LOOM system (Mac Gregor & Bates, 1987), still one of the fastest knowledge representation systems currently available (Heinsohn et al., 1994).

While the chart parser is completely coded in Smalltalk, the PT system is implemented in Actalk (Briot, 1989) — an extension of Smalltalk which simulates the asynchronous communication and concurrent execution of actors on sequential architectures. Thus, rather than exploiting parallelism, the PT parser currently suffers from a scheduling overhead. A more thorough comparison abstracting from these implementational considerations can be made at the level of method calls. We here consider the computationally expensive methods SYNTAXCHECK and CONCEPTCHECK (cf. Section 2). Especially the latter consumes large computational resources, as mentioned above, since for each syntactic interpretation variant a context has to be built in the KB system and its conceptual consistency must be checked continuously. The number of calls to these methods is given by the plots in Figs. 8 and 9. Sentences are ordered by increasing numbers of calls to SYNTAXCHECK as executed by the CP (this correlates fairly well with the syntactic complexity of the input). The values for sentences 39--41 in Fig. 8 are left out in order to preserve a proper scaling of the figure for plotting (39: 14389, 40=41: 27089 checks). A reduction of the total numbers of syntactic as well as semantic checks by a factor of nine to ten can be observed applying the strategies discussed for the PT system, i.e., the basic protocol plus skipping and backtracking.

[7] The active chart parser by Winograd (1983) was adapted to parsing a dependency grammar. No packing or structure sharing techniques could be used since the analyses have continuously to be interpreted in conceptual terms. We just remark that the polynomial time complexity known from chart parsing of context-free grammars does not carry over to linguistically adequate versions of dependency grammars (Neuhaus & Bröker, 1997).

[8] The problem caused by the single missing sentence is discussed in Section 4.2.

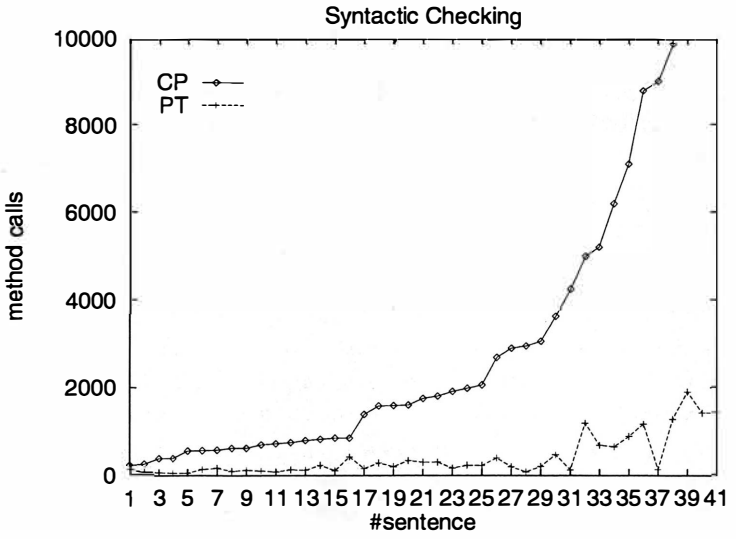


Figure 8: Calls to SYNTAXCHECK

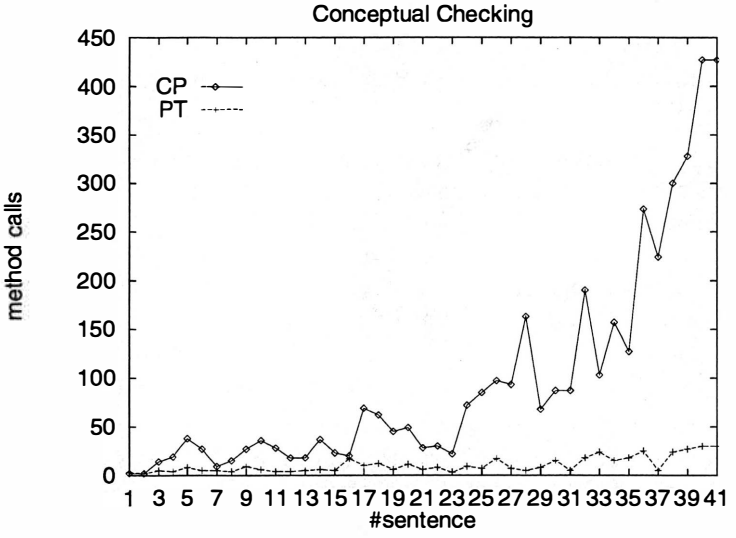


Figure 9: Calls to CONCEPTCHECK

4.2 Linguistic Aspects

The well-known PP attachment ambiguities pose a high processing burden for any parsing system. At the same time, PP adjuncts often convey crucial information from a conceptual point of view as in sentence 40: *Bei einer Blockgröße, die kleiner als 32 KB ist, erreicht die Quantum-Festplatte beim sequentiellen Lesen einen Datendurchsatz von 1.100 KB/s bis 1.300 KB/s. [For a block size of less than 10 KB, the Quantum hard disk drive reaches a data throughput of 1.100 KB/s to 1.300 KB/s for sequential reading].* Here, the chart parser considers all 16 globally ambiguous analyses stemming from ambiguous PP attachments.

Apart from the speed-up discussed above the PT parser behaves robust in the sense that it can gracefully handle cases of underspecification or ungrammaticality. For instance, sentence 36 (*Im direkten Vergleich zur Seagate bietet sie für denselben Preis weniger Kapazität. [In direct comparison to the Seagate drive, it (the tested drive) offers less capacity for the same price.]*) contains an unspecified word 'weniger' (i.e. 'less') such that no complete and correct analysis could be produced. Still, the PT parser was able to find a 'near miss', i.e., a *discontinuous* analysis skipping just that word.

A case where the PT parser failed to find the correct analysis was sentence 39: *Die Geräuschentwicklung der Festplatte ist deutlich höher als die Geräuschentwicklung der Maxtor 7080A. [The drive's noise level is clearly higher than the noise level of the Maxtor 7080A].* When the adverb 'deutlich' (i.e. 'clearly') is processed it is immediately attached to the

matrix verb as an adjunct. Actually it should modify 'höher' (i.e. 'higher'), but as it is not mandatory no backtrack is initiated by the PT parser to find the correct analysis.

5 Conclusion

The incomplete depth-first nature of our approach leads to a significant speed-up of processing approximately in the order of one magnitude, which is gained at the risk of not finding a correct analysis at all. This lack of completeness resulted in the loss of about 10% of the parses in our experiments and correlates with fewer global ambiguities. We expect to find even more favorable results for the PT system when processing the complete corpus, i.e., when processing material that requires prediction mechanisms.

Acknowledgments. We would like to thank our colleagues in the CLIF group for fruitful discussions and instant support. P. Neuhaus was supported by a grant from the interdisciplinary Graduate Program *"Menschliche und maschinelle Intelligenz"* (*"Human und machine intelligence"* at Freiburg University, N. Bröker was partially supported by a grants from DFG (Ha 2097/1-3).

References

Agha, Gul (1990). The structure and semantics of actor languages. In J. W. de Bakker et al. (Eds.), *Foundations of Object-Oriented Languages*, pp. 1–59. Berlin: Springer.

Allen, James F. (1993). Natural language, knowledge representation, and logical form. In M. Bates & R.M. Weischedel (Eds.), *Challenges in Natural Language Processing*, Studies in Natural Language Processing, pp. 146–175. Cambridge, MA: Cambridge University Press.

Briot, Jean-Pierre (1989). Actalk: A testbed for classifying and designing actor languages in the Smalltalk-80 environment. In *Proc. of the European Workshop on Object-Based Concurrent Computing. Nottingham, U.K., 10–14 Jul 1989*, pp. 109–129.

Charniak, Eugene (1993). *Statistical Language Learning*. Cambridge, MA: MIT Press.

Chomsky, Noam (1965). *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Earley, Jay (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Grosz, Barbara J., Aravind K. Joshi & Scott Weinstein (1995). Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225.

Hahn, Udo, Katja Markert & Michael Strube (1996a). A conceptual reasoning approach to textual ellipsis. In *ECAI '96 — Proc. of the 12th European Conference on Artificial Intelligence*, pp. 572–576, Budapest, Hungary, August 11-16, 1996. Chichester etc.: J. Wiley.

Hahn, Udo, Susanne Schacht & Norbert Bröker (1994). Concurrent, object-oriented natural language parsing: The PARSETALK model. *International Journal of Human-Computer Studies*, 41(1–2):179–222.

Hahn, Udo & Klemens Schnattinger (1997). Deep knowledge discovery from natural language texts. In *KDD – 97: Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, Newport Beach, Calif., August 14–17, 1997.

Hahn, Udo, Klemens Schnattinger & Martin Romacker (1996b). Automatic knowledge acquisition from medical texts. In *Proc. of the 1996 AMIA Annual Fall Symposium (formerly SCAMC). Beyond the Superhighway: Exploiting the Internet with Medical Informatics*, pp. 383–387, Washington, D.C., October 26-30, 1996.

Hawkins, John A. (1994). *A Performance Theory of Order and Constituency*. Cambridge Studies in Linguistics 73. Cambridge: Cambridge University Press.

Heinsohn, Jochen, Daniel Kudenko, Bernhard Nebel & Hans-Jürgen Profitlich (1994). An empirical analysis of terminological representation systems. *Artificial Intelligence*, 68(2):367–397.

Huyck, Christian R. & Steven L. Lytinen (1993). Efficient heuristic natural language parsing. In *Proc. of the 11th National Conf. on Artificial Intelligence. Washington, D.C., 11–15 Jul 1993*, pp. 386–391.

Jackendoff, Ray (1990). *Semantic Structures*. Cambridge, MA: MIT Press.

Kay, Martin (1986). Algorithm schemata and data structures in syntactic processing. In Barbara J. Grosz, Karen Sparck Jones & B. L. Webber (Eds.), *Readings in Natural Language Processing*, pp. 35–70. Los Altos, CA: M. Kaufmann. Originally published as a Xerox PARC technical report CSL-80-12, 1980.

Lavie, Alon & Masaru Tomita (1993). GLR* — an efficient noise-skipping parsing algorithm for context free grammars. In *Proc. of the 3rd Int. Workshop on Parsing Technology. Tilburg, NL & Durbuy, BE, 1993*, pp. 123–134.

Mac Gregor, Robert & Raymond Bates (1987). *The LOOM Knowledge Representation Language*. Technical Report RS-87-188: Information Sciences Institute, University of Southern California.

Markert, Katja & Udo Hahn (1997). On the interaction of metonymies and anaphora. In *IJCAI '97 — Proc. of the 15th Intl. Joint Conference on Artificial Intelligence*, Nagoya, Japan, August 23–29, 1997. San Francisco/CA: Morgan Kaufmann.

Mellish, Chris S. (1989). Some chart-based techniques for parsing ill-formed input. In *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics. 1989*, pp. 102–109.

Neuhaus, Peter & Norbert Bröker (1997). The complexity of recognition of linguistically adequate dependency grammars. In *Proc. of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conf. of the European Chapter of the Association for Computational Linguistics. Madrid, ES, 7–12 Jul 1997.*

Neuhaus, Peter & Udo Hahn (1996). Restricted parallelism in object-oriented lexical parsing. In *COLING '96 — Proc. of the 16th Intl. Conf. on Computational Linguistics*, Vol. 1, pp. 502–507, Copenhagen, Denmark, August 5-9, 1996.

Shieber, Stuart M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. Lecture Notes 4. Stanford, CA: CSLI.

Strube, Michael & Udo Hahn (1995). PARSETALK about sentence- and text-level anaphora. In *Proc. of the 7th Conf. of the European Chapter of the Association for Computational Linguistics. Dublin, Ireland, 27–31 Mar 1995*, pp. 237–247.

Strube, Michael & Udo Hahn (1996). Functional centering. In *Proc. of the 34th Annual Meeting of the Association for Computational Linguistics*, pp. 270–277, University of California at Santa Cruz, California, USA, 24-27 June 1996. San Francisco/CA: Morgan Kaufmann.

Thibadeau, Robert, Marcel A. Just & Patricia A. Carpenter (1982). A model of the time course and content of reading. *Cognitive Science*, 6:157–203.

Uszkoreit, Hans (1991). Strategies for adding control information to declarative grammars. In *Proc. of the 29th Annual Meeting of the Association for Computational Linguistics. Berkeley, CA, 18–21 Jun 1991*, pp. 237–245.

Winograd, Terry (1983). *Language as a Cognitive Process*, Vol. 1, Syntax. Reading, MA: Addison-Wesley.

Woods, W. A. & J. G. Schmolze (1992). The KL-ONE family. *Computers & Mathematics with Applications*, 23(2–5):133–177.

PROBABILISTIC PARSE SELECTION BASED ON SEMANTIC COOCCURRENCES*

Eirik Hektoen
Computer Laboratory, University of Cambridge
Pembroke Street, Cambridge CB2 3QG, UK
Eirik.Hektoen@cl.cam.ac.uk
http://www.cl.cam.ac.uk/users/eh101

Abstract

This paper presents a new technique for selecting the correct parse of ambiguous sentences based on a probabilistic analysis of lexical cooccurrences in semantic forms. The method is called "Semco" (for semantic cooccurrence analysis) and is specifically targeted at the differential distribution of such cooccurrences in correct and incorrect parses. It uses Bayesian Estimation for the cooccurrence probabilities to achieve higher accuracy for sparse data than the more common Maximum Likelihood Estimation would. It has been tested on the *Wall Street Journal* corpus (in the PENN Treebank) and shown to find the correct parse of 60.9% of parseable sentences of 6–20 words.

1 Introduction

In recent years there have been many proposals for probabilistic natural language parsing techniques, that is, techniques which not only find the possible syntactic derivations for a sentence, but also attempt to determine the most likely parse according to some probabilistic model. A basic example is a probabilistic context free grammar (PCFG), in which each production rule is associated with the conditional probability of it being applied when the left-hand non-terminal occurs in the generation of a sentence (e.g., Baker, 1982; Kupiec, 1991; Pereira and Schabes, 1992). This effectively regards the derivation of a sentence as a top–down recursive stochastic process, starting with the sentence non-terminal and ending with a random sentence in the language being modelled. While a PCFG is a pleasingly simple model, the fact that it assumes the choice of production at each step is only conditional on the left-hand non-terminal is a limitation to its accuracy. Some variations have therefore been proposed, by which the probability of a rule (or more generally, a parse derivation step) is made conditional on an extended view of the preceding derivation. For example, Briscoe and Carroll (1993) associate probabilities with transitions in an LR(1) parse table (partly reflecting the left context and a one-word lookahead), while Black *et al.* (1993) present a model in which virtually any aspect of the partial parse at any point in a derivation may be taken into account.

Both Briscoe and Carroll and Black *et al.* tie the probabilistic analysis to the precise parsing algorithm being used, effectively shifting the emphasis from modelling sentence generation to parse selection as a goal in its own right. Others take this further by separating the parsing and the parse selection completely. Hindle and Rooth (1993), for example, propose a system for resolving PP attachment ambiguities by comparing the degree of statistical association between the possible verb/preposition/noun triples after an arbitrary parser has produced a set of syntactically possible parses. This approach is attractive because it is based on lexical cooccurrences, which may reflect the acceptability of the meaning of a parse, but is less comprehensive than the other methods mentioned, since only a particular kind of ambiguity is covered.

*I am very grateful to Ted Briscoe, John Carroll, Miles Osborne, John Daugman, Gerald Gazdar, Steve Young and the anonymous IWPT-97 reviewers for their valuable advice. The work was made possible by a generous grant from the Norwegian Research Council (*Norsk Forskningsråd*, ST.30.33.221752) and an Overseas Research Students Award from the Committee of Vice-Chancellors and Principals of the Universities of the United Kingdom (ORS/9109219).

Yet others have made the opposite move, and presented parsing methods where the probabilistic analysis is used as the main driving force for the parser independently of any linguistically motivated grammar. Both Magerman (1995) and Collins (1996) propose such systems, where complex statistical patterns extracted from a Treebank constitute both the (shallow) syntactic grammar and selectional criteria. The results are systems which are comprehensive in the types of ambiguities they can handle and designed to extract a highly detailed and wide ranging statistical data from the training corpus, but which do not take any advantage of the analytical syntactic rules encoded in a formal grammar, and which do not support the derivation of semantic forms.

The system presented in this paper is essentially a specialised parse selector based on semantic forms derived from the parses found by a separate parser, and can therefore be used with any grammar and parser supporting formal semantics. It is based on lexical cooccurrences in terms of the predicates in the semantic forms, but handles all predicates uniformly and is therefore generally comprehensive in the types of ambiguities covered. It uses a complex statistical analysis to extract a large set of probabilistic parameters from the training corpus, but does not abandon the use of a formal grammar. Significantly, the system uses *Bayesian Estimation*[1] rather than simple Maximum Likelihood Estimation (MLE) for determining cooccurrence probabilities. This appears to be a sufficient response to the high degree of sparseness in the lexical cooccurrence data without the blurring associated with smoothing and clustering techniques (generally required for MLE). It seems reasonable to expect that a parse selection system should benefit from being trained on the same form of data that it is to be applied to—that is, specifically on the selection of the correct parse in sets of possible parses for different sentences rather than the unconditional probability of correct parses in isolation. The focus of the training in this system is therefore the differential distribution of cooccurrences in correct versus incorrect parses. The system is called "Semco" (for semantic cooccurrence analysis), and has been trained and tested on (separate parts of) the *Wall Street Journal* corpus in the PENN Treebank.

2 Definitions

The aim of the Semco analysis is to model cooccurrences of lexical predicates in semantic forms for the purpose of parse selection. A semantic form is here assumed to be a logical expression or description (including unscoped or quasi logical forms) derived in a compositional manner from a syntactic parse tree, and thus in general representing one of several possible interpretations of a sentence. A *cooccurrence* should represent the variable semantic linking of the predicates (generally representing lexical items from the sentence) in such expressions, and is therefore defined as the coincidence of two predicates being applied to the same element (a quantified variable or a constant). More precisely, if the ith argument of the predicate Q and the jth argument of the predicate R are the same, the semantic form is said to include the cooccurrence $(Q.i, R.j)$ understood as an unordered pair.

To see how such cooccurrences can be used for sentence disambiguation, consider the following exchange (from Andy Warhol, 1975):

(1) B: *Is that a female impersonator?*
A: *Of what?*

The expression *female impersonator* is so commonly used as a noun–noun compound ("an impersonator of a female"), that it comes as a surprise here when A's reply requires an alternative reading in which *female* is used as an adjective (giving "an impersonator who is female"). The difference between the two readings is represented by the semantic predicates and cooccurrences shown in Table 1. Note that the cooccurrences are concise representations of the facts that *female* and *impersonator* form a noun compound in Reading 1, while *female* is used as an adjectival modifier of *impersonator* in Reading 2.

The probabilistic analysis treats cooccurrences as elementary, atomic units, so that for a given grammar and lexicon there is a finite set $\mathcal{C}$ of possible cooccurrences. As far as this analysis is concerned, a parse, or

[1] Note that *Bayesian Estimation* (following e.g. Freund, 1992) refers not merely to the use of Bayes's law, but to the particular technique of estimating unknown probabilities by integration over a continuous probability distribution applied in Section 4.

	Reading 1	Reading 2
Categories	$female_{\mathsf{N}}$ $impersonator_{\mathsf{N}}$	$female_{\mathsf{Adj}}$ $impersonator_{\mathsf{N}}$
Predicates	(female x) (impersonator y) (NCOMP y x)	(female x) (impersonator x)
Cooccurrences	(female.0 NCOMP.1) (impersonator.0 NCOMP.0)	(female.0 impersonator.0)

Table 1: Example predicates and cooccurrences

derivation, is regarded as a set of cooccurrences, with the set of all parses being $\mathcal{D} = \{d \mid d \subseteq \mathcal{C}\}$. Similarly, a sentence is regarded as a set of possible parses, such that the set of all possible sentences is $\mathcal{S} = \{s \mid s \subseteq \mathcal{D}\}$.

For each cooccurrence, parse and sentence there is a corresponding *event*—conventionally represented by the corresponding capital letter—referring to the status of a random sentence. More precisely:

- C is the event that the correct parse of the sentence includes the cooccurrence c.
- D is the event that the correct parse is d. It can be expressed as the conjunction of the cooccurrence events for all the cooccurrences in d and the negated cooccurrence events for all other cooccurrences:

$$D = \bigwedge_{c \in d} C \wedge \bigwedge_{c \notin d} \neg C. \tag{2}$$

- S is the event that the correct parse is an element of s, and is simply the disjunction of the corresponding parse events:

$$S = \bigvee_{d \in s} D. \tag{3}$$

3 The Event Space

Given the above definitions of cooccurrence, parse and sentence events, one's first reaction may be to regard the associated probabilities as the relative frequencies of the respective entities in the language—that is, in the correct analyses of the sentences in the training corpus. This is not the only possible definition, however, and would have serious disadvantages for the following analysis. For example, it would mean that the model should reflect the typical number of cooccurrences in an average sentence, making any assumption of independence between cooccurrences inappropriate (since the probabilities of any additional cooccurrence would diminish once the number has passed this average). It would also make it impossible for the training to be based on the differential distribution of cooccurrences in correct and other syntactically possible, but incorrect, parses of the training sentences, since such alternative parses would have no particular status in the model.

Alternative definitions of the basic probabilities are possible because all actual references to probabilities will be conditional on some given sentence. The only requirement of the probabilistic model is that it predicts the relative, conditional probabilities for different parses for any given sentence, while the unconditional probabilities of different sentences in the corpus are never directly relevant.

To derive a suitable definition of the basic event space here, we will take the presumed independence of all cooccurrence events as the starting point, based on the view that the model ought to satisfy the following basic assumption:

(4) **Basic Assumption:** The relative probabilities of any two parses depends only on the cooccurrences that distinguish between them, and not on any cooccurrence present in both of them, any cooccurrence absent in both of them, or any further possible parses of the same sentence.

The independence of the cooccurrence events follows by considering two parses, d_1 and d_2, that are only distinguished by a single cooccurrence c, say $c \in d_1$ and $d_2 = d_1 \setminus c$. By the basic assumption, the ratio $P(D_1|S) : P(D_2|S)$ is invariant for any such d_1 and d_2 and $s \supseteq \{d_1, d_2\}$, and this ratio must be $P(C) : P(\neg C)$. From the independence of the cooccurrence events and (2) it follows that the unconditional probability of a parse event D is given by

$$P(D) = \prod_{c \in d} P(C) \prod_{c \notin d} P(\neg C). \tag{5}$$

4 Cooccurrence Probability Estimation

The cooccurrence probabilities $P(C)$ represent the basic parameters in the analysis and need to be estimated from the training corpus. In many other probabilistic analyses of corpus data the basic parameters are estimated as the observed relative frequencies of sentences displaying the relevant characteristics, but such a simple approach is not possible here. Instead, Bayesian estimation (see e.g. Freund, 1992) is used, by which the estimate is defined in terms of the distribution of the probability to be estimated regarded as a continuous probabilistic variable.

More precisely, the unknown probability $P(C)$ for a given c is regarded as the continuous probabilistic variable Θ with a value θ in the interval $(0, 1)$. Let s_i for i in $0, \ldots, t$ be all the sentences in the training corpus, and let d_i be the correct parse of each s_i. Let also S_i be the event corresponding to s_i, and let C_i be the event associated with the cooccurrence c with respect to the sentence s_i. The overall status of the corpus with respect to c can then be expressed as the conjunction of the events $\hat{S} = \bigwedge S_i$ and

$$\hat{C} = \bigwedge_{c \in d_i} C_i \wedge \bigwedge_{c \notin d_i} \neg C_i. \tag{6}$$

The distribution of Θ given the observed status of the corpus can then be expressed as the probability density function $\phi(\theta|\hat{C}, \hat{S})$, which according to Bayes's law is given by

$$\phi(\theta|\hat{C}, \hat{S}) = P(\hat{C}|\theta, \hat{S}) \frac{h(\theta|\hat{S})}{P(\hat{C}|\hat{S})}. \tag{7}$$

Here $h(\theta|\hat{S})$ is the probability density function for the distribution of Θ given only the sentences in the corpus (i.e., not knowing the correct parses), and $P(\hat{C}|\hat{S})$ is the (discrete) probability of $\hat{C}$ not knowing θ. In other words, $h(\theta|\hat{S})$ is effectively (for our purposes) the *prior* distribution of Θ (regarding $\hat{S}$ as fixed), while $\phi(\theta|\hat{C}, \hat{S})$ is the *posterior* distribution of the same given the correct disambiguation of the sentences implied by $\hat{C}$.

The probability $P(\hat{C}|\theta, \hat{S})$, that is, the probability of the correct (observed) parse selections in the corpus in terms of the cooccurrence c and given $\theta = P(C)$, is given according to (6) and the fact that each sentence represents an independent draw in the event space by the product

$$P(\hat{C}|\theta, \hat{S}) = \prod_{c \in d_i} P(C_i|\theta, \hat{S}) \prod_{c \notin d_i} \left(1 - P(C_i|\theta, \hat{S})\right). \tag{8}$$

To determine $P(C_i|\theta, \hat{S})$, let n_i and m_i be the number of parses of s_i which do and do not, respectively, include c. Since the ratio of the probabilities of each of the former to each of the latter is $\theta : (1 - \theta)$, we get

$$P(C_i|\theta, \hat{S}) = \frac{n_i \theta}{n_i \theta + m_i(1 - \theta)}. \tag{9}$$

Returning to equation (7), the probability $P(\hat{C}|\hat{S})$ can now be determined from $P(\hat{C}|\theta, \hat{S})$ by the integral

$$P(\hat{C}|\hat{S}) = \int_0^1 P(\hat{C}|\theta, \hat{S}) h(\theta|\hat{S})\, d\theta. \tag{10}$$

This leaves only the prior distribution, $h(\theta|\hat{S})$, which cannot be determined analytically, but which may be estimated empirically from a preliminary estimation of all the cooccurrences in the corpus (e.g., using $h(\theta|\hat{S}) = 1$ as a temporary simplification without seriously affecting the overall distribution).

Having found the posterior distribution $\phi(\theta|\hat{C},\hat{S})$, a straightforward application of the Bayesian estimation method would be to estimate the unknown probability $P(C)$ as the *expected* value of Θ, that is

$$E(\Theta|\hat{C},\hat{S}) = \int_0^1 \phi(\theta|\hat{C},\hat{S})\,\theta\,d\theta. \tag{11}$$

A slight variation of this will be used here, however, based on the observation that the expected value operator in (11) essentially represents a continuous, arithmetic mean of the cooccurrence probability θ weighted by the distribution function $\phi(\theta|\hat{C},\hat{S})$. As such, it would represent a reasonable estimation of $P(C)$ if the result were to be used as a term in a sum of such results, but according to (5) the main use of a cooccurrence probability will be represented by a factor of either $P(C)$ or $1 - P(C)$ in a parse probability. As the net effect of the presence or absence of the cooccurrence in a parse thus is to either multiply or divide the parse probability by $\frac{\theta}{1-\theta}$, a more suitable estimate of $P(C)$ is found by applying the expected value operator to $\ln\frac{\theta}{1-\theta}$, that is, defining the estimate as $\tilde{p}_c$ given by the equation[2]

$$\ln\frac{\tilde{p}_c}{1-\tilde{p}_c} = E\Big(\ln\frac{\Theta}{1-\Theta}|\hat{C},\hat{S}\Big) = \int_0^1 \phi(\theta|\hat{C},\hat{S})\,\ln\frac{\theta}{1-\theta}\,d\theta. \tag{12}$$

From (7), (8), (10) and (12), noting that the integral in (10) does not depend on θ and can therefore be moved outside that in (12), we then get the following overall expression for the cooccurrence probability estimate:

$$\ln\frac{\tilde{p}_c}{1-\tilde{p}_c} = \frac{\int_0^1 P(\hat{C}|\theta,\hat{S})h(\theta|\hat{S})\ln\frac{\theta}{1-\theta}\,d\theta}{\int_0^1 P(\hat{C}|\theta,\hat{S})h(\theta|\hat{S})\,d\theta}. \tag{13}$$

To summarise, the effect of all this is that the posterior probabilistic distribution of the cooccurrence probability is determined from the prior distribution of such probabilities and the observations relating to this cooccurrence in the corpus. The posterior distribution represents the full knowledge we have of the likelihood of different possible values of the cooccurrence probability, and the final estimate is defined as that which represents the best approximation (for our purposes) of this as a fixed value.

5 Implementation Notes

The previous section derived (13) in conjunction with (8) and an empirical estimation of $h(\theta|\hat{S})$ as the main expression for the estimate $\tilde{p}_c$ of a cooccurrence probability $P(C)$. In practice, only sentences with at least one parse that includes c and one that doesn't, that is, for which $n_i, m_i > 0$, have any effect on the result. For any such sentence, moreover, it is enough to record the ratio

$$r_i = \frac{n_i}{m_i} \tag{14}$$

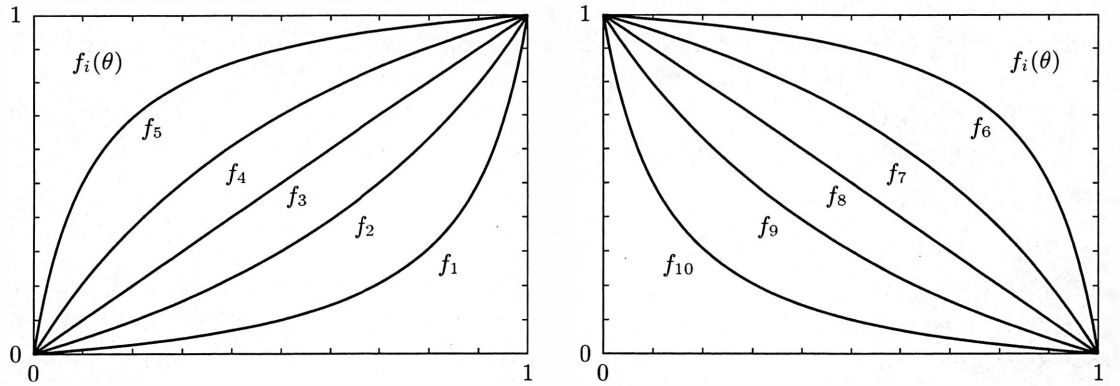
in the training data. Then, by defining

$$f(\theta) = \left[\prod_{n_i,m_i>0} f_i(\theta)\right] h(\theta|\hat{S}) \tag{15}$$

and

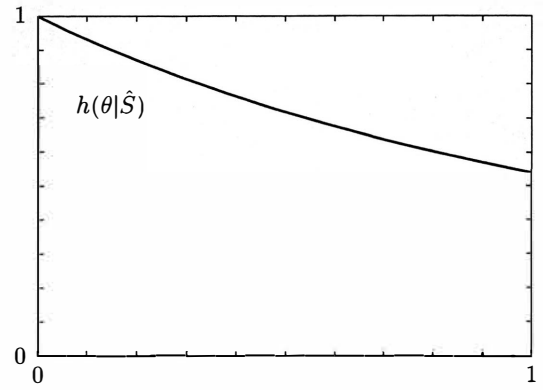
$$f_i(\theta) = \begin{cases} P(C_i|\theta,\hat{S}) &= r_i\theta/(r_i\theta+1-\theta) & \text{if } c \in d_i \\ P(\neg C_i|\theta,\hat{S}) &= (1-\theta)/(r_i\theta+1-\theta) & \text{otherwise,} \end{cases} \tag{16}$$

[2]The convergence of the integral in (12), although $\ln\frac{\theta}{1-\theta} \to \pm\infty$ for $\theta \to 0$ and $\theta \to 1$, follows from the well-known convergence of $\int_0^1 \ln\theta\,d\theta = -1$ and the fact that the relevant probability distribution functions are finite.

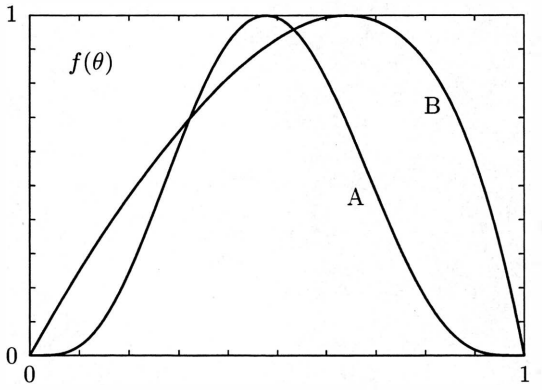


(a) examples of $f_i(\theta)$ for $r_i = \frac{1}{9}, \frac{3}{7}, \frac{5}{5}, \frac{7}{3}, \frac{9}{1}$ (for $i = 1, \ldots, 5$, resp.) and $c \in d_i$

(b) more examples of $f_i(\theta)$ for the same r_i (for $i = 6, \ldots, 10$, resp.) but with $c \notin d_i$



(c) the prior distribution $h(\theta|\hat{S}) = \frac{1.36}{(1+0.36\theta)^2}$ in the *Wall Street Journal* (found by manual curve fitting to the results of a preliminary analysis)



(d) possible posterior distributions based on the preceding functions:
A: $f(\theta) = \left[\prod_1^{10} f_i(\theta)\right] h(\theta|\hat{S})$,
B: $f(\theta) = \left[\prod_{2,7} f_i(\theta)\right] h(\theta|\hat{S})$

Figure 1: Cooccurrence probability density functions. All functions are scaled to give a maximum value of 1 with no significance for the cooccurrence probability estimation.

equation (13) is reduced to

$$\ln \frac{\tilde{p}_c}{1 - \tilde{p}_c} = \frac{\int_0^1 f(\theta) \ln \frac{\theta}{1-\theta} \, d\theta}{\int_0^1 f(\theta) \, d\theta}. \tag{17}$$

The functions $f_i(\theta)$ and $f(\theta)$ represent scaled probability density functions of Θ: $f_i(\theta)$ is (proportional to) the apparent distribution based only on sentence i; $f(\theta)$ is (proportional to) the posterior distribution based on all the relevant sentences as well as the prior distribution $h(\theta|\hat{S})$. Fig. 1 shows some typical examples of the $f_i(\theta)$ functions, the $h(\theta|\hat{S})$ found to represent the distribution of cooccurrence probabilities in the *Wall Street Journal* corpus, and two possible forms of $f(\theta)$ derived from them.

The actual computation of $\tilde{p}_c$ will have to be by numerical integration of (17) based on the r_i values extracted from the training corpus for each cooccurrence.[3] This may seem rather costly in computational terms, but is feasible in practice with a careful implementation. In particular, it is generally not necessary to compute $\tilde{p}_c$ for all the cooccurrences in the training corpus, since in most cases only a small fraction of them will ever be

[3]To avoid arithmetic overflow at or near $\theta = 0$ and $\theta = 1$, however, it is convenient to rewrite the nominator in (17) as $\int_0^1 \big([f(\theta) - f(0)] \ln \theta - [f(\theta) - f(1)] \ln(1 - \theta)\big) d\theta - f(0) + f(1)$, and omit the logarithms when the preceding factors are 0.

required. Instead, a practical implementation may compile indexed files with the r_i values extracted from the corpus for each cooccurrences and compute the small number of cooccurrence probabilities required for any sentence when the need arises.

6 Test Results

An implementation of the Semco analysis has been tested with a feature unification-based, medium-wide coverage, syntactic/semantic grammar and sentences from the bracketted form of the *Wall Street Journal* part of the PENN Treebank. Due to limitations in the grammar and parser, the corpus had to be restricted in different ways: It was first limited to sentences of 6–20 words (after the deletion of parenthetical material) which parsed successfully with at most 100 parses (as more ambiguous ones would add disproportionately to the overall cost of the computation). It was further reduced by rejecting sentences for which no parse achieved a minimum threshold similarity with the corpus bracketting, or for which more than 30% of the parses achieved the same, maximal such similarity.

The parse/bracketting similarity measure used was a weighted sum of different counts based on shared constituents (with or without matching labels), crossing brackets, and the overall length of the sentence (for normalising the result). The threshold was set to correspond, typically, to a parse including about three-quarters of the corpus constituents with no crossing brackets. Multiple parses achieving the same, maximal score above this threshold (within the limit mentioned above) were presumed "equally correct" for training and testing purposes. The number of sentences left in the corpus after each step in the selection process were:

Number of sentences of length 6–20 words	20155
... which parsed successfully	12162
... which had up to 100 parses	10852
... which met the similarity measure requirements	7887

The final set of sentences was shuffled and divided into five partitions, allowing a cycle of five training/testing iterations to be run with different splits of the corpus into $\frac{4}{5}$ for training and $\frac{1}{5}$ testing.

Table 2 shows a detailed overview of the main test statistics. The main part of the table refers to a particular division of the corpus, and shows a breakdown by sentence length and ambiguity. This is followed in the last row by the mean results of the five training/testing rounds based on different divisions of the corpus.

The Parseval measures in the table have become a frequently used, common standard for comparison of different systems, but are only partly relevant for a system such as this. The fact that they are taken at constituent level means that they primarily measure the ability of the system to reproduce the precise bracketting in the corpus rather than the correct selection of parses as such. As a result, fully correct parses receive reduced scores where the grammar used by the parser differs significantly from that assumed in the corpus, while incorrect parses are credited to the extent that some of their constituents coincide with those in the corpus. In this system the grammar generally produces much more detailed parses than the corpus (i.e., with many more constituents, e.g. at bar-1 level), meaning that the *precision* rate is severely reduced and effectively rendered meaningless, while the *recall* rate is not similarly badly affected. This systematic imbalance between the corpus bracketing and the parses also prompted the inclusion of two *crossing brackets* rates: where four words are bracketed as "(w_1 w_2) w_3 w_4" in the corpus but parsed as "w_1 ((w_2 w_3) w_4)", for example, it counts as one crossed bracket in the corpus ("xb/c") but two in the parse ("xb/p"). Again, the generally greater number of constituents in the parses than in the corpus means that the latter is artificially high and a poor basis for comparison with other systems.

The table includes three variant Parseval measures. The "non-crossing precision", which is like the standard precision rate except that any constituent in the parse that does not actually cross brackets with the corpus form is assumed to be correct, is intended as an illustration of a possible way to deal with the problem discussed above. The labelled precision and recall rates are the common variations of the Parseval measures where constituents are required to have matching syntactic labels.

	Parseval measures				Variants			Correct parse ranked in top n				
	prec	rec	xb/c	xb/p	nxpr	lpre	lrec	1	2	3	4	5
Results with the original partitioning of the corpus for training/testing:												
All sents	61.3	87.2	0.40	0.79	93.1	58.0	82.5	60.8	75.1	81.3	85.6	89.2
Length												
6–10	70.0	94.3	0.12	0.20	97.2	67.7	91.2	82.4	91.2	94.2	97.0	98.7
11–15	61.5	87.1	0.41	0.76	93.5	58.2	82.5	58.9	75.3	82.7	87.2	90.6
16–20	57.2	83.7	0.67	1.45	91.0	53.5	78.4	41.2	57.9	65.7	71.3	77.1
Ambiguity												
1–20	62.8	89.9	0.31	0.53	95.1	59.8	85.6	69.3	83.1	88.6	92.9	95.6
21–40	59.0	83.1	0.63	1.20	90.9	55.3	77.9	42.5	60.2	69.2	74.7	80.1
41–60	58.1	81.4	0.69	1.47	89.6	54.2	75.8	37.9	56.0	64.7	67.2	71.6
61–80	56.8	78.7	0.59	2.12	85.0	53.4	73.9	35.1	45.9	51.4	55.4	63.5
81–100	58.2	83.2	0.44	1.40	90.1	54.5	78.0	38.5	46.2	51.9	55.8	61.5
The mean results over a cycle of five different partitionings of the corpus:												
All sents	61.3	87.4	0.41	0.77	93.4	58.0	82.7	60.9	74.7	82.2	85.9	89.3

All entries are in %, except xb/c and xb/p which are per sentence. Definitions:

prec	*precision*, the proportion of the constituents in the parses also found in the corpus
rec	*recall*, the proportion of the constituents in the corpus also found in the parses
xb/c, xb/p	*crossing brackets rate*, counted in the corpus form or selected parse, resp.
nxpr	*non-crossing precision*, constituents in the parse that don't cross any brackets in the corpus
lpre, lrec	*labelled precision/recall*, where the syntactic labels in the parse and the corpus must match
1–5	*n best correct selection rate*, sentences for which the correct parse (i.e., the best match with the corpus bracketting) is ranked in the top n parses by the selection algorithm

Table 2: Parse selection accuracy test results

The measures that most directly reflect the practical accuracy of the Semco system in my opinion are those headed "Correct parse ranked in top n", as these show the relative frequencies of fully correct, sentence-level disambiguations ($n = 1$) and near-misses (n in 2–5). In the compilation of these figures, the "correct" parse of each sentence was identified by comparison with the corpus bracketting using the same similarity formula that was used for the training of the system.

The table shows, as one would expect, that the selection accuracy tends to diminish as sentences increase in length or ambiguity, but the differences in the three higher bands of ambiguities are relatively minor (indeed, for many measures the sentences with 81–100 parses do better than those of 41–60 parses). This indicates that the decision to omit sentences of more than 100 parses from the testing is not likely to have affected the overall performance of the system greatly.

7 Conclusions

The results in Table 2 show that the Semco technique achieves relatively high levels of parse selection accuracy, and that it therefore may represent a good practical method of sentence disambiguation. This is reflected in the Parseval recall rate of 87.4%, the average of only 0.41 crossing brackets per sentence (with respect to the corpus bracketting), and in the correct disambiguation of 60.9% of the sentences (with this figure increasing to 74.7% and 82.2% if near-misses ranked 2nd or 3rd are included).

Comparison with other published work is made difficult by the problems with the Parseval measures discussed

	Sentence lengths	Precision	Recall	Labelled precision	Labelled recall	Crossing brackets
Semco	6–20	61.3	87.4	58.0	82.7	0.41
	11–20	59.8	85.8	56.3	80.9	0.51
Magerman (1995)	10–20	90.8	90.3	89.0	88.5	0.49
(SPATTER)	4–40	86.3	85.8	84.5	84.0	1.33
Collins (1996)	1–40			86.3	85.8	1.14

Entries are in %, except the crossing brackets rate, which is per sentence. Note that the *precision* and *labelled precision* rates are poor measures of the Semco system's accuracy because of the more detailed parses produced by the grammar compared to the corpus annotations. The crossing brackets rate included for Semco is that counted against the corpus bracketting ("xb/c").

Table 3: Parseval measures of parse accuracy of different systems

above and by other incompatibilities between the different systems and the precise corpora used. Table 3 shows, however, the main figures from the Semco system and results published by Magerman (1995) (for his "SPATTER" system) and Collins (1996). Considering the strong relationship between sentence length and accuracy shown in Table 2, the most comparable figures in Table 3 are those for Semco with 11-20 words and Magerman's SPATTER system with 10–20 words. Disregarding the precision rates, which are severely biased against Semco, the table shows that while SPATTER performs somewhat better in terms of recall, the difference in the crossing brackets rates is insignificant. (Collins only includes results for sentence of 1–40 words, making any direct comparison with Semco dubious, but the results are broadly similar to Magerman's for 4–40 words.)

To be fair, the Semco results are based only on a subset of suitable, parseable sentences in the corpus (about 39% of the original sentences within the length range), but this is an inevitable consequence of the major differences between the systems. Magerman's and Collins's systems are both based on extracting local syntactic structures along with their statistical distribution directly from the corpus annotation, and are therefore independent of any linguistic analysis of the corresponding syntax. This robustness can be an important advantage—it is very hard to write a formal grammar with something approaching full coverage of naturally occurring languages—but has several disadvantages too. In particular, the lack of a linguistic analysis means that there is no guarantee that a parse generated by these systems represents a meaningful sentence-wide syntax, and the models cannot support interpretation through compositional semantics.

The Semco system is compatible with any parsing technique capable of supporting formal semantics, making it potentially much more useful in a wider, practical NLP system where any form of interpretation is required. The lack of robustness in the experimental set-up discussed in this paper is not a consequence of the Semco technique, but a reflection of the limited coverage of the grammar used. In future developments of this work, it is intended that the Semco system will be used with a fully wide-coverage natural language grammar and an n-best parser that includes mechanisms for handling undergeneration.

To conclude, the main novel aspect of the Semco analysis is the way the probabilistic model is based on the differential distribution of cooccurrences in correct and incorrect parses. This requires an analysis in which probabilities don't represent direct frequencies in the data, but rather correspond to a hypothetical event universe of which real sentences are only a small fragment. Simply put, the event universe includes sentences with any number of cooccurrences—whether they require a million words or ten—but this is no problem for the practical application of the system which is always conditional on a concrete, given sentence. There is no need for normalisation of the probabilities of parses with different numbers of cooccurrences in this analysis, sine the presence and absence of any cooccurrence are regarded as complementary events.

The use of Bayesian estimation for the basic cooccurrence probabilities is partly a requirement from the probabilistic model, but is also a means of dealing with the highly sparse data in a theoretically motivated manner. Where MLE tends to be highly inaccurate for very sparse data—requiring clustering or smoothing—

and directly inappropriate for unseen data, Bayesian estimation finds the result *theoretically expected* to represent the best approximation of the true probability based on a full analysis of the latter's continuous probability distribution. The result is unable to distinguish between different cooccurrences with the same observations (e.g., unseen), as smoothing or clustering might, but are statistically unbiased such that the random errors in the probability estimates for a set of cooccurrences will tend to cancel out and lead to improved accuracy in the parse probabilities. More detailed experiments by Hektoen (1997) show, moreover, that clustering the cooccurrence data in combination with Bayesian estimation reduces the accuracy of the parse selection.

References

Baker, J. 1982. Trainable Grammars for Speech Recognition. *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550.

Black, Ezra, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer and Salim Roukos. 1993. Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings, 31st Annual Meeting of the Association for Computational Linguistics*, pages 31–37, Columbus, Ohio, USA.

Briscoe, Ted and John Carroll. 1993. Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics*, 19(1):25–59.

Collins, Michael John. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings, 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Santa Cruz, California, USA.

Freund, John E. 1992. *Mathematical Statistics*, 5th edition. Prentice–Hall International, Inc., Englewood Cliffs, New Jersey, USA.

Hektoen, Eirik. 1997. *Statistical Parse Selection using Semantic Cooccurrences.* PhD thesis, Churchill College, Cambridge University, Cambridge, UK. Available from `http://www.cl.cam.ac.uk/users/eh101`.

Hindle, Donald and Mats Rooth. 1993. Structural Ambiguity and Lexical Relations. *Computational Linguistics*, 19(1):103–120.

Kupiec, Julian. 1991. A Trellis-Based Algorithm for Estimating the Parameters of a Hidden Stochastic Context-Free Grammar. *DARPA Speech and Natural Language Workshop*, Asilomar, California, USA.

Magerman, David M. 1995. Statistical Decision-Tree Models for Parsing. In *Proceedings, 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, Massachusetts, USA.

Pereira, Fernando and Yves Schabes. 1992. Inside–Outside Re-estimation for Partially Bracketed Corpora. In *Proceedings, 30th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Newark, Delaware, USA.

Schabes, Yves. 1992. Stochastic Lexicalised Tree-Adjoining Grammars. In *Proceedings of the fifteenth International Conference on Computational Linguistics: COLING-92*, volume 2, pages 426–432, Nantes, France.

Warhol, Andy. 1975. *The Philosophy of Andy Warhol.* Harcourt Brace Jovanovich, San Diego, California, USA.

A New Formalization of Probabilistic GLR Parsing

Kentaro Inui, Virach Sornlertlamvanich, Hozumi Tanaka, and Takenobu Tokunaga
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
2-12-1 O-okayama Meguro Tokyo 152, Japan
{inui,virach,take,tanaka}@cs.titech.ac.jp

Abstract

This paper presents a new formalization of probabilistic GLR language modeling for statistical parsing. Our model inherits its essential features from Briscoe and Carroll's generalized probabilistic LR model [3], which obtains context-sensitivity by assigning a probability to each LR parsing action according to its left and right context. Briscoe and Carroll's model, however, has a drawback in that it is not formalized in any probabilistically well-founded way, which may degrade its parsing performance. Our formulation overcomes this drawback with a few significant refinements, while maintaining all the advantages of Briscoe and Carroll's modeling.

1 Introduction

The increasing availability of text corpora has encouraged researchers to explore statistical approaches for various tasks in natural language processing. Statistical parsing is one of these approaches. In statistical parsing, one of the most straightforward methodologies is to generalize context-free grammars by associating a probability with each rule in producing probabilistic context-free grammars (PCFGs). However, as many researchers have already pointed out, PCFGs are not quite adequate for statistical parsing due to the lack of context-sensitivity. Probabilistic GLR parsing is one existing statistical parsing methodology which is more context-sensitive than PCFG-based parsing.

Several attempts have been made to incorporate probability into generalized LR (GLR) parsing [17]. For example, Wright and Wrigley proposed an algorithm to distribute probabilities originally associated with CFG rules to LR parsing actions, in such a way that the resulting model is equivalent to the original PCFG [19]. Perhaps, the most naive way of coupling a PCFG model with the GLR parsing framework would be to assign the probability associated with each CFG rule to the reduce actions for that rule. Wright and Wrigley expanded on this general methodology by distributing probabilities to shift actions as well as reduce actions, so that the parser can prune improbable parse derivations after shift actions as well as reduce actions. This can be advantageous particularly when one considers applying a GLR parser to, for example, continuous speech recognition. However, since their principal concern was in compiling PCFGs into the GLR parsing framework, their language model still failed to capture the context-sensitivity of languages.

Su et al. proposed a way of introducing probabilistic distribution into the shift-reduce parsing framework [16]. Unlike Wright and Wrigley's work, the goal of this research was the construction of a mildly context-sensitive model[1] that is effective for statistical parsing. Their model distributes probabilities to stack transitions between two shift actions, and associates a probability with each parse derivation, given by the product of the probability of each change included in the derivation. Further, they also described an algorithm to handle this model within the GLR parsing framework, gaining parse efficiency. However, since their probabilistic model in itself is not intimately coupled with the GLR parsing algorithm, their model needs an additional complex algorithm for training.

On the other hand, Briscoe and Carroll proposed the distribution of probabilities directly to each action in an LR table to realize mildly context-sensitive parsing [3]. Their model overcomes the drawback of context-insensitivity of PCFGs by estimating the probability of each LR parsing action according to its left (i.e. LR parse state) and right context (i.e. next input symbol). The probability of each parse derivation is computed as the product of the probability assigned to each action included in the derivation. Unlike the approach of

[1] By "a mildly context-sensitive model", we mean a model that is moderately more context-sensitive than context-free models such as PCFGs, but not a *fully* context-sensitive one, which would be intractable in both training and parsing.

Su et al., this makes it easy to implement context-sensitive probabilistic parsing by slightly extending GLR parsers, and the probabilistic parameters can be easily trained simply by counting the frequency of application of each action in parsing the training sentences. Furthermore, their model is expected to be able to allow the parser to prune improbable parse derivations at an equivalently fine-grained level as that of Wright and Wrigley's statistical parser, since it assigns probabilities to both shift and reduce actions. However, in as far as we have tested the performance of Briscoe and Carroll's model (B&C model, hereafter) in our preliminary experiments, it seems that, in many cases, it does not significantly improve on the performance of the PCFG model, and furthermore, in the worst case, it can be even less effective than the PCFG model. According to our analysis, these seem to be result, principally, of the method used for normalizing probabilities in their model, which may not be probabilistically well-founded. In fact, Briscoe and Carroll have not explicitly presented any formalization of their model.

This line of reasoning led us to consider a new formalization of probabilistic GLR (PGLR) parsing. In this paper, we propose a newly formalized PGLR language model for statistical parsing, which has the following advantages:

- It provides probabilistically well-founded distributions.
- It realizes mildly context-sensitive statistical parsing.
- It can be trained simply by counting the frequency of each LR parsing action.
- It allows the parser to prune improbable parse derivations, even after shift actions.

Although the performance of our model should be evaluated through large-scaled experiments, we are achieving promising results in our preliminary experiments. In what follows, we first present our new formalization of PGLR parsing (section 2). We then review B&C model according to our formalization, demonstrating that B&C model may not be probabilistically well-founded through the use of simple examples (section 3). We finally discuss how our refinement is expected to influence parsing performance through a further example (section 4).

2 A PGLR Language Model

Suppose we have a CFG and its corresponding LR table. Let $\boldsymbol{V}_n$ and $\boldsymbol{V}_t$ be the nonterminal and terminal alphabets, respectively, of the CFG. Further, let $\boldsymbol{S}$ and $\boldsymbol{A}$ be the sets of LR parse states and parsing actions appearing in the LR table, respectively. For each state $s \in \boldsymbol{S}$, the LR table specifies a set $La(s) \subseteq \boldsymbol{V}_t$ of possible next input symbols. Further, for each coupling of a state s and input symbol $l \in La(s)$, the table specifies a set of possible parsing actions: $Act(s, l) \subseteq \boldsymbol{A}$. Each action $a \in \boldsymbol{A}$ is either a shift action or reduce action. Let $\boldsymbol{A}_s$ and $\boldsymbol{A}_r$ be the set of shift and reduce actions, respectively, such that $\boldsymbol{A} = \boldsymbol{A}_s \cup \boldsymbol{A}_r \cup \{accept\}$ (*accept* is a special action denoting the completion of parsing).

As with most statistical parsing frameworks, given an input sentence, we rank the parse tree candidates according to the probabilities of the parse derivations that generate those trees. In LR parsing, each parse derivation can be regarded as a sequence of transitions between LR parse stacks, which we describe in detail below. Thus, in the following, we use the terms parse tree, parse derivation, and stack transition sequence interchangeably.

Given an input word sequence $W = \{w_1, \ldots, w_n\}$, we estimate the distribution over the parse tree candidates T as follows:

$$P(T|W) = \alpha \cdot P(T) \cdot P(W|T) \tag{1}$$

The first scaling factor α is a constant that is independent of T, and thus does not need to be considered in ranking parse trees. The second factor $P(T)$ is the distribution over all the possible trees that can be derived from a given grammar, such that, for $\boldsymbol{\mathcal{T}}$ being the infinite set of all possible trees:

$$\sum_{T \in \boldsymbol{\mathcal{T}}} P(T) = 1 \tag{2}$$

We estimate this syntactic distribution $P(T)$ using a PGLR model. The third factor $P(W|T)$ is the distribution of lexical derivations from T, where each terminal symbol of T is assumed to be a part of speech symbol. Most statistical parsing frameworks estimate this distribution by assuming that the probability of the i-th word

$w_i \in W$ depends only on its corresponding terminal symbol (i.e. part of speech) l_i. Since l_i is uniquely specified by T for each i, we obtain equation (3):

$$P(W|T) \approx \prod_{w_i \in W} P(w_i|l_i) \tag{3}$$

One could use a more context-sensitive model to estimate the lexical distribution $P(W|T)$; for example, one could introduce the statistics of word collocations. However, this is beyond the scope of this paper. For further discussion, see [7].

A stack transition sequence T can be described as (4):

$$\sigma_0 \overset{l_1,a_1}{\Longrightarrow} \sigma_1 \overset{l_2,a_2}{\Longrightarrow} \cdots \overset{l_{n-1},a_{n-1}}{\Longrightarrow} \sigma_{n-1} \overset{l_n,a_n}{\Longrightarrow} \sigma_n \tag{4}$$

where σ_i is the i-th stack, whose stack-top state is denoted by $top(\sigma_i)$, and $l_i \in La(top(\sigma_{i-1}))$ and $a_i \in Act(top(\sigma_{i-1}), l_i)$ are, respectively, an input symbol and a parsing action chosen at σ_{i-1}. It can be proven from the LR parsing algorithm that, given an input symbol $l_{i+1} \in La(top(\sigma_i))$ and an action $a_{i+1} \in Act(top(\sigma_i), l_{i+1})$, the next (derived) stack $next(\sigma_i, a_{i+1})$ $(= \sigma_{i+1})$ can always be uniquely determined as follows:

- If the current action a_{i+1} is a shift action for an input symbol l_{i+1}, then the parser consumes l_{i+1}, pushing l_{i+1} onto the stack, and then pushes the next state s_{i+1}, which is uniquely specified by the LR table, onto the stack.
- If the current action a_{i+1} is a reduction by a rule $A \rightarrow \beta$, the parser derives the next stack as follows. The parser first pops $|\beta|$ grammatical symbols together with $|\beta|$ state symbols off the stack, where $|\beta|$ is the length of β. In this way, the stack-top state s_j is exposed. The parser then pushes A and s_{i+1} onto the stack, with s_{i+1} being the entry specified in the LR goto table for s_j and A. All these operations are executed deterministically.

A parse derivation completes if $l_n = \$$ and $a_n = accept$. We say stack transition sequence T is complete if $l_n = \$$, $a_n = accept$, and $\sigma_n = final$, where *final* is a dummy symbol denoting the stack when parsing is completed. Hereafter, we consistently refer to an LR parse state as a *state* and an LR parse stack as a *stack*. And, unless defined explicitly, s_i denotes the stack-top state of the i-th stack σ_i, i.e. $s_i = top(\sigma_i)$.

The probability of a complete stack transition sequence T can be decomposed as:

$$\begin{aligned} P(T) &= P(\sigma_0, l_1, a_1, \sigma_1, \ldots, \sigma_{n-1}, l_n, a_n, \sigma_n) && (5) \\ &= P(\sigma_0) \cdot \prod_{i=1}^{n} P(l_i, a_i, \sigma_i | \sigma_0, l_1, a_1, \sigma_1, \ldots, l_{i-1}, a_{i-1}, \sigma_{i-1}) && (6) \end{aligned}$$

Here we assume that σ_i contains all the information of its preceding parse derivation that has any effect on the probability of the next transition, namely:

$$P(l_i, a_i, \sigma_i | \sigma_0, l_1, a_1, \sigma_1, \ldots, l_{i-1}, a_{i-1}, \sigma_{i-1}) \approx P(l_i, a_i, \sigma_i | \sigma_{i-1}) \tag{7}$$

This assumption simplifies equation (6) to:

$$P(T) = \prod_{i=1}^{n} P(l_i, a_i, \sigma_i | \sigma_{i-1}) \tag{8}$$

Now, we show how we estimate each transition probability $P(l_i, a_i, \sigma_i | \sigma_{i-1})$, which can be decomposed as in (9):

$$P(l_i, a_i, \sigma_i | \sigma_{i-1}) = P(l_i | \sigma_{i-1}) \cdot P(a_i | \sigma_{i-1}, l_i) \cdot P(\sigma_i | \sigma_{i-1}, a_i, l_i) \tag{9}$$

To begin with, we estimate the first term $P(l_i|\sigma_{i-1})$ as follows:

Case 1. $i = 1$:

$$P(l_1|\sigma_0) = P(l_1|s_0) \tag{10}$$

Case 2. The previous action a_{i-1} is a shift action, i.e. $a_{i-1} \in \boldsymbol{A}_s$. We assume that only the current stack-top state $s_{i-1} = top(\sigma_{i-1})$ has any effect on the probability of the next input symbol l_i. This means that:

$$P(l_i|\sigma_{i-1}) \approx P(l_i|s_{i-1}) \tag{11}$$

where

$$\sum_{l \in La(s)} P(l|s) = 1 \tag{12}$$

Case 3. The previous action a_{i-1} is a reduce action, i.e. $a_{i-1} \in \boldsymbol{A}_r$. Unlike Case 2, the input symbol does not get consumed for reduce actions, and thus the next input symbol l_i is always identical to l_{i-1}; namely, l_i can be deterministically predicted. Therefore,

$$P(l_i|\sigma_{i-1}) = 1 \tag{13}$$

Next, we estimate the second term $P(a_i|\sigma_{i-1}, l_i)$ relying on the analogous assumption that only the current stack-top state s_{i-1} and input symbol l_i have any effect on the probability of the next action a_i:

$$P(a_i|\sigma_{i-1}, l_i) \approx P(a_i|s_{i-1}, l_i) \tag{14}$$

where

$$\sum_{a \in Act(s,l)} P(a|s,l) = 1 \tag{15}$$

Finally, as mentioned above, given the current stack σ_{i-1} and action a_i, the next stack σ_i can be uniquely determined:

$$P(\sigma_i|\sigma_{i-1}, l_i, a_i) = 1 \tag{16}$$

As shown in equations (11) and (13), the probability $P(l_i|\sigma_{i-1})$ should be estimated differently depending on whether the previous action a_{i-1} is a shift action or a reduce action. Let $\boldsymbol{S}_s$ be the set containing s_0 and all the states reached immediately after applying a shift action, and $\boldsymbol{S}_r$ be the set of states reached immediately after applying a reduce action:

$$\boldsymbol{S}_s \stackrel{\text{def}}{=} \{s_0\} \cup \{s | \exists a \in \boldsymbol{A}_s, \sigma : \ s = top(next(\sigma, a))\} \tag{17}$$

$$\boldsymbol{S}_r \stackrel{\text{def}}{=} \{s | \exists a \in \boldsymbol{A}_r, \sigma : \ s = top(next(\sigma, a))\} \tag{18}$$

where s_0 is the initial state. Note that these two sets are mutually exclusive[2]:

$$\boldsymbol{S} = \boldsymbol{S}_s \cup \boldsymbol{S}_r \quad \text{and} \quad \boldsymbol{S}_s \cap \boldsymbol{S}_r = \emptyset \tag{19}$$

Equations (9) through (18) can be summarized as:

$$P(l_i, a_i, \sigma_i|\sigma_{i-1}) \approx \begin{cases} P(l_i, a_i|s_{i-1}) & (\text{for } s_{i-1} \in \boldsymbol{S}_s) \\ P(a_i|s_{i-1}, l_i) & (\text{for } s_{i-1} \in \boldsymbol{S}_r) \end{cases} \tag{20}$$

Since $\boldsymbol{S}_s$ and $\boldsymbol{S}_r$ are mutually exclusive, we can assign a single probabilistic parameter to each action in an LR table, according to equation (20). To be more specific, for each state $s \in \boldsymbol{S}_s$, we associate a probability $p(a)$ with each action $a \in Act(s,l)$ (for $l \in La(s)$), where $p(a) = P(l, a|s)$ such that:

$$\sum_{l \in La(s)} \sum_{a \in Act(s,l)} p(a) = 1 \quad (\text{for } s \in \boldsymbol{S}_s) \tag{21}$$

[2]It is obvious from the algorithm for generating an LR(1) goto graph[1] that, for each state s ($\neq s_0$), if there exist states s_i and s_j whose goto transitions on symbol X_i and X_j, respectively, both lead to s, then $X_i = X_j$. Namely, for any given state s, the symbol X required to reach s by way of a goto transition is always uniquely specified. On the other hand, if the current state is in S_s, then it should have been reached through a goto transition on a certain terminal symbol $X \in V_t$, whereas, if the current state is in S_r, then it should have been reached through a goto transition on a certain nonterminal symbol $X \in V_n$. Given these facts, it is obvious that S_s and S_r are mutually exclusive.

On the other hand, for each state $s \in \boldsymbol{S}_r$, we associate a probability $p(a)$ with each action $a \in Act(s,l)$ (for $l \in La(s)$), where $p(a) = P(a|s,l)$ such that:

$$\sum_{a \in Act(s,l)} p(a) = 1 \quad (\text{for } s \in \boldsymbol{S}_r) \tag{22}$$

Through assigning probabilities to actions in an LR table in this way, we can estimate the probability of a stack transition sequence T as given in (4) by computing the product of the probabilities associated with all the actions included in T:

$$P(T) = \prod_{i=1}^{n} p(a_i) \tag{23}$$

Before closing this section, we describe the advantages of our PGLR model. Our model inherits some of its advantages from B&C model. First, as in equation (14), the probabilistic distribution of each parsing action depends on both its left context (i.e. LR parse state) and right context (i.e. input symbol). This incorporates context-sensitivity into the model. We elaborate this through an example in section 4. Second, since the probability of each parse derivation can be estimated simply as the product of the probabilities associated with all the actions in that derivation, we can easily implement a probabilistic LR parser through a simple extension to the original LR parser. We can also easily train the model, as we need only count the frequency of application of each action in generating correct parse derivations for each entry in the training corpus. Third, both B&C model and our model are expected to be able to allow the parser to prune improbable parse derivations at an equivalently fine-grained level as that for Wright and Wrigley's statistical parser, since these two models assign probabilities to both shift and reduce actions. Furthermore, since our model assigns a single probabilistic parameter to each action in an LR table similarly to B&C model, the algorithm proposed by Carroll and Briscoe [4] for efficient unpacking of packed parse forests with probability annotations can be equally applicable to our model. Finally, although not explicitly pointed out by Briscoe and Carroll, it should also be noted that PCFGs give long-term preference over structures but do not sufficiently reflect short-term bigram statistics of terminal symbols, whereas both B&C model and our PGLR model reflect these types of preference simultaneously. $P(l_i|s_{i-1})$ in equation (11) is a model that predicts the next terminal symbol l_i for the current left context $s_{i-1} \in \boldsymbol{S}_s$. In this case of $s_{i-1} \in \boldsymbol{S}_s$, since s_{i-1} uniquely specifies the previous terminal symbol l_{i-1}, $P(l_i|s_{i-1}) = P(l_i|s_{i-1}, l_{i-1})$, which is a slightly more context-sensitive version of the bigram model of terminal symbols $P(l_i|l_{i-1})$. This feature is expected to be significant particularly when one attempts to integrate syntactic parsing with morphological analysis in the GLR parsing framework (e.g. [11]), since the bigram model of terminal symbols has been empirically proven to be effective in morphological analysis.

Besides these advantages, which are all shared with B&C model, our model overcomes the drawback of B&C model; namely, our model is based on a probabilistically well-founded formalization, which is expected to improve the parsing performance. We discuss this issue in the remaining sections.

3 Comparison with Briscoe and Carroll's Model

In this section, we briefly review B&C model, and make a qualitative comparison between their model and ours.

In our model, we consider the probabilities of transitions between stacks as given in equation (8), whereas Briscoe and Carroll consider the probabilities of transitions between *LR parse states* as below:

$$P(T) \approx \prod_{i=1}^{n} P(l_i, a_i, s_i|s_{i-1}) \tag{24}$$

$$= \prod_{i=1}^{n} P(l_i, a_i|s_{i-1}) \cdot P(s_i|s_{i-1}, l_i, a_i) \tag{25}$$

Briscoe and Carroll initially associate a probability $p(a)$ with each action $a \in Act(s,l)$ (for $s \in \boldsymbol{S}_s$, $l \in La(s)$) in an LR table, where $p(a)$ corresponds to $P(l_i, a_i|s_{i-1})$, the first term in (25):

$$p(a) = P(l, a|s) \tag{26}$$

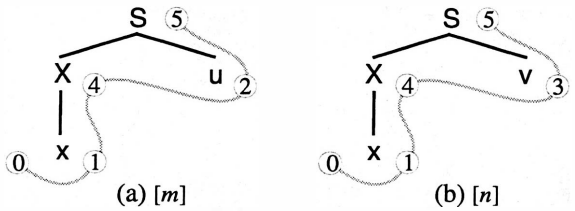


Figure 1. Parse trees derived from grammar $G1$ (The square-bracketed value below each tree denotes the number of occurrences of that tree.)

Table 1. LR table for grammar $G1$, with trained parameters (The numbers given in the middle and bottom of each row the parameters for B&C model and our model, respectively.)

state	action				goto	
	u	v	x	$	X	S
0			**sh1** $(m+n)$		4	5
(S_s)			1			
			1			
1	**re3** (m)	**re3** (n)				
(S_s)	$m/(m+n)$	$n/(m+n)$				
	$m/(m+n)$	$n/(m+n)$				
2				**re1** (m)		
(S_s)				1		
				1		
3				**re2** (n)		
(S_s)				1		
				1		
4	**sh2** (m)	**sh3** (n)				
(S_r)	$m/(m+n)$	$n/(m+n)$				
	1	1				
5				**acc** $(m+n)$		
(S_r)				1		
				1		

such that:

$$\forall s \in \boldsymbol{S}. \sum_{l \in La(s)} \sum_{a \in Act(s,l)} p(a) = 1 \tag{27}$$

In this model, the probability associated with each action is normalized in the same manner for any state. However, as discussed in the previous section, the probability assigned to an action should be normalized differently depending on whether the state associated with the action is of class $\boldsymbol{S}_s$ or $\boldsymbol{S}_r$ as in equations (21) and (22). Without this treatment, probability $P(l_i|s_{i-1})$ in equation (11) could be incorrectly duplicated for a single terminal symbol, which would make it difficult to give probabilistically well-founded semantics to the overall score. As a consequence, in B&C formulation, the probabilities of all the complete parse derivations may not sum up to one, which would be inconsistent with the definition of $P(T)$ (see equation (2)).

To illustrate this, let us consider grammar $G1$ as follows.

Grammar G1:

(1) S → X u
(2) S → X v
(3) X → x

This grammar allows only two derivations as shown in Figure 1. Suppose that we have tree (a) with frequency m, and (b) with frequency n in the training set. Training B&C model and our model, respectively, with these trees, we obtain the models as shown in Table 1, where, for each LR parse state, each bracketed value in the top row denotes the number of occurrences of the action associated with it, and the numbers in the middle and bottom of each row denote the probabilistic parameters of B&C model and our model, respectively.

Given this setting, the probability of each tree in Figure 1 is computed as follows (see Figure 1, where each circled number denotes the LR parse state reached after parsing has proceeded from the left-most corner to the

location of that number):

$$P_{\mathrm{B\&C}}(\mathrm{tree(a)}) = 1 \cdot \frac{m}{m+n} \cdot \frac{m}{m+n} \cdot 1 = \left(\frac{m}{m+n}\right)^2 \tag{28}$$

$$P_{\mathrm{B\&C}}(\mathrm{tree(b)}) = 1 \cdot \frac{n}{m+n} \cdot \frac{n}{m+n} \cdot 1 = \left(\frac{n}{m+n}\right)^2 \tag{29}$$

$$P_{\mathrm{PGLR}}(\mathrm{tree(a)}) = 1 \cdot \frac{m}{m+n} \cdot 1 \cdot 1 = \frac{m}{m+n} \tag{30}$$

$$P_{\mathrm{PGLR}}(\mathrm{tree(b)}) = 1 \cdot \frac{n}{m+n} \cdot 1 \cdot 1 = \frac{n}{m+n} \tag{31}$$

where B&C denotes B&C model and PGLR denotes our model. This computation shows that our model correctly fits the distribution of the training set, with the sum of the probabilities being one. In the case of B&C model, on the other hand, the sum of these two probabilities is smaller than one. The reason can be described as follows. After shifting the left-most x, which leads the process to state 1, the model predicts the next input symbol as either u or v, and chooses the reduce action in each case, reaching state 4. So far, both B&C model and our model behave in the same manner. In state 4, however, B&C model repredicts the next input symbol, despite it already having been determined in state 1. This duplication makes the probability of each tree smaller than what it should be. In our model, on the other hand, the probabilities in state 4, which is of class $\boldsymbol{S}_r$, are normalized for each input symbol, and thus the prediction of the input symbol u (or v) is not duplicated.

Briscoe and Carroll are also required to include the second factor $P(s_i|s_{i-1}, l_i, a_i)$ in (25) since this factor does not always compute to one. In fact, if we have only the information of the current state and apply a reduce action in that state, the next state is not always uniquely determined. For this reason, Briscoe and Carroll further subdivide probabilities assigned to reduce actions according to the stack-top states exposed immediately after the pop operations associated with those reduce action. Contrastively, in our model, given the current stack, the next stack after applying any action can be uniquely determined as in (16), and thus we do not need to subdivide the probability for any reduce action.

To illustrate this, let us take another simple example in grammar $G2$ as given below, with all the possible derivations shown in Figure 2. Further, the LR table is shown in Table 2.

Grammar G2:

(1) S → u X
(2) S → v X
(3) X → x

Let us compute again the probability of each tree for the two models:

$$P_{\mathrm{B\&C}}(\mathrm{tree(a)}) = \frac{m}{m+n} \cdot 1 \cdot \frac{m}{m+n} \cdot 1 = \left(\frac{m}{m+n}\right)^2 \tag{32}$$

$$P_{\mathrm{B\&C}}(\mathrm{tree(b)}) = \frac{n}{m+n} \cdot 1 \cdot \frac{n}{m+n} \cdot 1 = \left(\frac{n}{m+n}\right)^2 \tag{33}$$

$$P_{\mathrm{PGLR}}(\mathrm{tree(a)}) = \frac{m}{m+n} \cdot 1 \cdot 1 \cdot 1 = \frac{m}{m+n} \tag{34}$$

$$P_{\mathrm{PGLR}}(\mathrm{tree(b)}) = \frac{n}{m+n} \cdot 1 \cdot 1 \cdot 1 = \frac{n}{m+n} \tag{35}$$

In B&C model, the probability assigned to the reduce action in state 3 with the next input symbol being $ is subdivided according to whether the state reached after the stack-pop operation is state 1 or 2 (see Table 2). This makes the probability of each tree smaller than what it should be.

The above examples illustrate that, in B&C model, the probabilities of all the possible parse derivations may not necessarily sum up to one, due to the lack of probabilistically well-founded normalization. In our model, on the other hand, the probabilities of all the derivations are guaranteed to always sum to one[3]. This flaw in B&C model can be considered to be related to Briscoe and Carroll's claim that their model tends to

[3]Precisely speaking, this is the case if the model is based on a canonical LR (CLR) table. In the case of lookahead LR (LALR) tables, the probabilities of all the complete derivations may not sum up to one even for the case of our model, since some stack

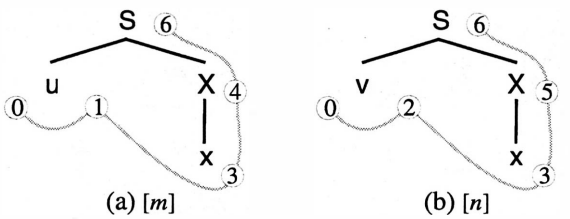


Figure 2. Parse trees derived from grammar $G2$ (The square-bracketed value below each tree denotes the number of occurrences of that tree.)

Table 2. LR table for grammar $G2$, with trained parameters (For each state, the numbers given in the middle and bottom of each row denote the parameters for B&C model and our model, respectively. Each middle bracketed number for state 3 denotes the state exposed by the stack-pop operation associated with the corresponding reduce action.)

state	action				goto	
	u	v	x	$	X	S
0	**sh1** (m)	**sh2** (n)				6
(S_s)	$m/(m+n)$	$n/(m+n)$				
	$m/(m+n)$	$n/(m+n)$				
1			**sh3** (m)		4	
(S_s)			1			
			1			
2			**sh3** (n)		5	
(S_s)			1			
			1			
3				**re3** $(m+n)$		
(S_s)				$^{(1)}m/(m+n)$; $^{(2)}n/(m+n)$		
				1		
4				**re1** (m)		
(S_r)				1		
				1		
5				**re2** (n)		
(S_r)				1		
				1		
6				**acc** $(m+n)$		
(S_r)				1		
				1		

favor parse trees involving fewer grammar rules, almost regardless of the training data. In B&C model, stack transition sequences involving more reduce actions tend to be assigned much lower probabilities for the two reasons mentioned above: (a) the probabilities assigned to actions following reduce actions tend to be lower than what they should be, since B&C model repredicts the next input symbols immediately after reduce actions, (b) the probabilities assigned to reduce actions tend to be lower than what they should be, since they are further subdivided according to the stack-top states exposed by the stack-pop operations. Therefore, given the fact that stack transition sequences involving fewer reduce actions correspond to parse trees involving fewer grammar rules, it is to be expected that B&C model tends to strongly prefer parse trees involving fewer grammar rules.

transitions may not be accepted (for details of CLR and LALR, see, for example, [1, 5]). However, this fact will never prevent our model from being applicable to LALR. Let $\mathcal{T}_{acc} \subset \mathcal{T}$ be the infinite set of all possible complete and *acceptable* stack transition sequences. The second factor $P(T)$ in equation (1) is a distribution over complete and acceptable stack transition sequences such that the probabilities of all the transition sequences in $\mathcal{T}_{acc}$ sum up to one. However, if one considers an LALR-based model, since there may be rejected transition sequences in $\mathcal{T}$, equation (1) should be replaced as follows:

$$P(T|W) = \alpha \cdot P(T|T \in \mathcal{T}_{acc}) \cdot P(W|T) = \alpha' \cdot P(T) \cdot P(W|T)$$

where α' is a constant that is independent of T, and $P(T)$ is a distribution over all the possible complete transition sequences, whether acceptable or not, such that $\sum_{T \in \mathcal{T}} P(T) = 1$. Thus, one can rank the parse tree candidates for any given input sentence according to $P(T)$ and $P(W|T)$, whether one bases the model on CLR, LALR, or even LR(0) (i.e. SLR). For further discussion, see [8].

To solve this problem, Briscoe and Carroll proposed calculating the geometric mean of the probabilities of the actions involved in each stack transition sequence. However, this solution makes their model even further removed from a probabilistically well-founded model. In our model, on the other hand, any bias toward shorter derivations is expected to be much weaker, and thus we do not require the calculation of the geometric mean.

One may wonder to what extent these differences matter for practical statistical parsing. Although this issue needs to be explored through large-scaled empirical evaluation, it must be still worthwhile to consider some likely cases where the difference discussed here will influence parsing performance. We discuss such a case through a further example in the next section.

4 Expected Impact on Parsing Performance

In this section, we first demonstrate through an example how B&C model and our model, which we class as GLR-based models here, realize mildly context-sensitive statistical parsing, as compared to the PCFG model. We then return to the issue raised at the end of the previous section.

Suppose we have grammar $G3$ as follows:

Grammar G3:

(1) S → u S
(2) S → v S
(3) S → x
(4) S → S S

Further, let us assume that we train the PCFG model, B&C model, and our PGLR model, respectively, using a training set as shown in Figure 3, where trees (a) and (b) are the parse trees for input sentence $W_1 = \{u, x, x\}$, and (c) and (d) are those for $W_2 = \{v, x, x\}$. Table 3 shows the LR table for grammar $G3$, with the trained

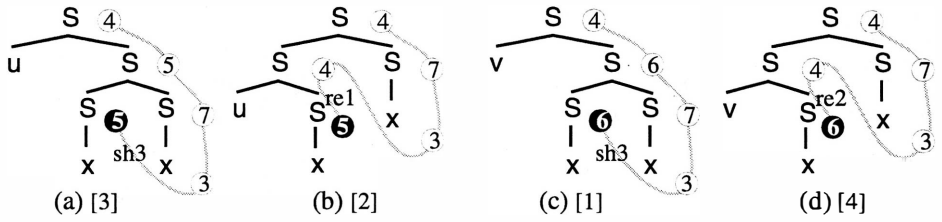


Figure 3. Training set (The square-bracketed number below each tree denotes the number of occurrences of that tree.)

parameters[4].

According to the training data in Figure 3, right branching (i.e. tree (a)) is preferred for input sentence W_1, whereas left branching (i.e. tree (d)) is preferred for input sentence W_2[5]. It is easy to see that the PCFG model does not successfully learn these preferences for either of the sentences, since all the parse trees produced for each sentence with involve the same set of grammar rules.

Unlike the PCFG model, both the GLR-based models can learn these preferences in the following way. In the LR parsing process for sentence W_1, the point where the parser must choose between parse trees (a) and (b) is in state 5, which is reached after the reduction of the left-most x into S (see Figure 3). In state 5, if the shift action is chosen, parse tree (a) is derived, while, if the reduce action is chosen, (b) is derived. Thus, the preference for (a) to (b) is reflected in the distribution over the shift-reduce conflict in this state. Table 3 shows that both B&C model and our model correctly prefer the shift action in state 5 with the next input symbol being x. For input sentence W_2, on the other hand, the left branching tree (d) is preferred. This preference is also reflected in the distribution over the shift-reduce conflict in the state reached after the reduction of the left-most x into S, but, this time, the state is state 6 instead of state 5. According to Table 3, state 6 with the next input symbol being x correctly prefers the reduce action, which derives the left-branching tree (d). In sum,

[4]In practical applications, when computing parameters, one would need to use some smoothing technique in order to avoid assigning zero to any parameter associated with an action that had never occurred in training.

[5]Such preferences are likely to be observed in real corpora. For example, in Japanese sentences, a conjunction at the beginning of a sentence tends to modify the rest of the sentence at the top-most level of the sentence. This corresponds to the preference for sentence W_1 with u equating to a conjunction. On the other hand, a postpositional phrase tends to be subordinated by its left-most verb, which corresponds to the preference for sentence W_2 with v and x being a postpositional phrase and a verb, respectively.

Table 3. LR table for grammar $G3$, with trained parameters (For each state, the numbers given in the middle and bottom of each row denote the parameters for B&C model and our model, respectively. Each middle bracketed number denotes the state exposed by the stack-pop operation associated with the reduce action corresponding to that number.)

state	action				goto
	x	y	z	$	S
0	**sh1**	**sh2**	**sh3**		4
(S_s)	.5	.5	0		
	.5	.5	0		
1	**sh1**	**sh2**	**sh3**		5
(S_s)	0	1	0		
	0	1	0		
2	**sh1**	**sh2**	**sh3**		6
(S_s)	0	0	1		
	0	0	1		
3	**re3**	**re3**	**re3**	**re3**	
(S_s)	0	0	$^{(1)}.25$; $^{(2)}.25$	$^{(4)}.3$; $^{(5)}.15$; $^{(6)}.05$	
	0	0	.5	.5	
4	**sh1**	**sh2**	**sh3**	**acc**	7
(S_r)	0	0	.23	.77	
	1	1	1	1	
5	**sh1/re1**	**sh2/re1**	**sh3/re1**	**re1**	7
(S_r)	0/0	0/0	$.38/^{(0)}.25$	.38	
	.5/.5	.5/.5	.6/.4	1	
6	**sh1/re2**	**sh2/re2**	**sh3/re2**	**re2**	7
(S_r)	0/0	0/0	$.17/^{(0)}.67$	.17	
	.5/.5	.5/.5	.2/.8	1	
7	**sh1/re4**	**sh2/re4**	**sh3/re4**	**re4**	7
(S_r)	0/0	0/0	0/0	$^{(0)}.6$; $^{(1)}.3$; $^{(2)}.1$	
	.5/.5	.5/.5	.5/.5	1	

the different preferences for W_1 and W_2 are reflected separately in the distributions assigned to the different states (i.e. states 5 and 6).

As illustrated in this example, for each parsing choice point, the LR parse state associated with it can provide a context for specifying the preference for that parse choice. This feature of the GLR-based models enables us to realize mildly context-sensitive parsing. Furthermore, although not explicitly demonstrated in the above example, it should also be noted that the GLR-based models are sensitive to the next input symbol as shown in (14) in section 2.

Now, let us see how the probabilities assigned to LR parsing actions are reflected in the probability of each parse tree. Table 4 shows the overall distributions provided by the PCFG model, B&C model, and our model, respectively, to the trees in Figure 3[6]. According to the table, our model accurately learns the distribution of the training data, whereas B&C model does not fit the training data very well. In particular, for sentence W_1, it goes as far as incorrectly preferring parse tree (b). This occurs due to the lack of well-founded normalization of probabilities as discussed in section 3. As mentioned above, B&C model correctly prefers the shift action in state 5, as does our model. However, for the rest of the parsing process, B&C model associates a considerably

[6]Although Briscoe and Carroll proposed to take the geometric mean of peripheral distributions as mentioned in section 3, we did not apply this operation when computing the probabilities in Table 4, to give the reader a sense of the difference between the probabilities given by B&C model and our model. Note that, in our example, since the number of state transitions involved in each parse tree is always the same for any given sentence, and given that the grammar generates only binary trees, taking the geometric mean would not change the preference order.

Table 4. Distributions over the parse trees from Figure 3 (trees (a) and (b) are the parse trees for input sentence $W_1 = \{u, x, x\}$, and (c) and (d) are those for $W_2 = \{v, x, x\}$)

	$P(\text{tree(a)}\|W_1)$	$P(\text{tree(b)}\|W_1)$	$P(\text{tree(c)}\|W_2)$	$P(\text{tree(d)}\|W_2)$
PCFG	.50	.50	.50	.50
B&C	.37	.63	.005	.995
PLR	.60	.40	.20	.80
training data	.60	.40	.20	.80

higher probability to the process from state 4 through 3 and 7 to 4, which derives tree (b), than the process from 3 through 7 and 5 to 4, which derives tree (a), since, in their model, the former process is inappropriately supported by the occurrence of tree (d). For example, in both parsing processes for (b) and (d), the pop operation associated with the reduction in state 3 exposes state 4, and B&C model thus assigns an inappropriately high probability to this reduction, compared to the reduction in state 3 for tree (a).

Of course, as far as various approximations are made in constructing a probabilistic model similar to both B&C model and our model, it is always the case that the model may not fit the training data precisely due to the insufficiency of the model's complexity. Analogous to B&C model, our model does not always fit the training data precisely due to the independence assumptions such as equations (7), (11), etc. However, it should be noted that, as illustrated by the above example, there is a likelihood that B&C model not fitting the training data is due not only to the insufficiency of complexity, but also the lack of well-founded normalization.

5 Conclusion

In this paper, we newly presented a formalization of probabilistic LR parsing. Our modeling inherits some of its features from B&C model. Namely, it is mildly context-sensitive, and naturally integrates short-term bigram statistics of terminal symbols and long-term preference over structures of parse trees. Furthermore, since the model is tightly coupled with GLR parsing, it can be easily implemented and trained. Inheriting these advantages, our formalization additionally overcomes an important drawback of B&C model: the lack of well-founded normalization of probabilities. We demonstrated through examples that this refinement is expected to improve parsing performance. Those examples may seem to be relatively artificial and forced. However, in our preliminary experiments, we are achieving some promising results, which support our claim (see [15] for a summary of preliminary results). We are now planning to conduct further large-scaled experiments.

It should also be noted that our modeling is equally applicable to both CLR tables and LALR tables. Since it is a highly empirical issue whether it is better to use CLR-based models or LALR-based models, it may be interesting to make experimental comparisons between these two types (for a qualitative comparison, see [8]).

Other approaches to statistical parsing using context-sensitive language models have also been proposed, such as [2, 9, 12, 14]. We need to make theoretical and empirical comparisons between these models and ours. The significance of introducing lexical sensitivity into language models should also not be underestimated. In fact, several attempts to use lexically sensitive models already exist: e.g. [6, 10, 13]. Our future research will also be directed towards this area [7].

Acknowledgements

The authors would like to thank the reviewers for their suggestive comments. They would also like to thank Mr. UEKI Masahiro and Mr. SHIRAI Kiyoaki (Tokyo Institute of Technology) for their fruitful discussion on the formalization of the proposed model. Finally, they would like to thank Mr. Timothy Baldwin (Tokyo Institute of Technology) for his help in writing this paper.

References

[1] A.V. Aho, S. Ravi, and J.D. Ullman. *Compilers, Principle, Techniques, and Tools.* Addision Wesely, 1986.

[2] E. Black, F. Jelinek, J. Lafferty, D. M. Magerman, R. Mercer, and S. Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pp. 31–37, 1993.

[3] T. Briscoe and J. Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, Vol. 19, No. 1, 1993.

[4] J. Carroll and E. Briscoe. Probabilistic normalization and unpacking of packed parse forests for unification-based grammars. In *Proceedings, AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pp. 33–38., 1992.

[5] N. P. Chapman. *LR Parsing — Theory and Practice.* Cambridge University Press, 1987.

[6] M. J. Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[7] K. Inui, K. Shirai, H. Tanaka, and T. Tokunaga. Integrated probabilistic language modeling for statistical parsing. Technical Report TR97-0005, Dept. of Computer Science, Tokyo Institute of Technology, 1997. Available from `http://www.cs.titech.ac.jp/tr.html`.

[8] K. Inui, V. Sornlartlamvanich, H. Tanaka, and T. Tokunaga. A new probabilistic LR language model for statistical parsing. Technical Report TR97-0004, Dept. of Computer Science, Tokyo Institute of Technology, 1997. Available from `http://www.cs.titech.ac.jp/tr.html`.

[9] K. Kita. Spoken sentence recognition based on HMM-LR with hybrid language modeling. *IEICE Trans. Inf. & Syst.*, Vol. E77-D, No. 2, 1994.

[10] H. Li. A probabilistic disambiguation method based on psycholinguistic principles. In *Proceedings of the Fourth Workshop on Very Large Corpora (WVLC-4)*, 1996. cmp-lg/9606016.

[11] H. Li and H. Tanaka. A method for integrating the connection constraints into an LR table. In *Proceedings of Natural Language Processing Pacific Rim Symposium '95*, pp. 703–708, 1995.

[12] Magerman. D. M. and Marcur. M. Pearl: A probabilistic chart parser. In *Proceedings of the 5th Conference of European Chapter of the Association for Computational Linguistics*, pp. 15–20, 1991.

[13] Y. Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, Vol. 2, pp. 425–432, 1992.

[14] S. Sekine and R. Grishman. A corpus-based probabilistic grammar with only two non-terminals. In *Proceedings of the International Workshop on Parsing Technologies '95*, 1995.

[15] V. Sornlertlamvanich, K. Inui, K. Shirai, H. Tanaka, T. Tokunaga, and T. Takezawa. Incorporating probabilistic parsing into an LR parser – LR table engineering (4) –. *Information Processing Sciety of Japan, SIG-NL-119*, 1997. Available from `http://tanaka-www.cs.titech.ac.jp/~inui/`.

[16] K.-Y. Su, J.-N. Wang, M.-H. Su, and J.-S. Chang. GLR parsing with scoring. In Tomita [18], chapter 7.

[17] M. Tomita. *An Efficient Parsing for Natural Languages.* Kluwer, Boston, Mass, 1986.

[18] M. Tomita, editor. *Generalised LR Parsing.* Kluwer Academic Publishers, 1991.

[19] J. H. Wright and E. N. Wrigley. GLR parsing with probability. In Tomita [18], chapter 8.

EFFICIENT PARSING FOR CCGS WITH GENERALIZED TYPE-RAISED CATEGORIES

Nobo Komagata *
Department of Computer and Information Science
University of Pennsylvania
komagata@linc.cis.upenn.edu

Abstract

A type of 'non-traditional constituents' motivates an extended class of Combinatory Categorial Grammars, CCGs with Generalized Type-Raised Categories (CCG-GTRC) involving variables. Although the class of standard CCGs is known to be polynomially parsable, unrestricted use of variables can destroy this essential requirement for a practical parser. This paper argues for polynomial parsability of CCG-GTRC from practical and theoretical points of view. First, we show that an experimental parser runs polynomially in practice on a realistic fragment of Japanese by eliminating spurious ambiguity and excluding genuine ambiguities. Then, we present a worst-case polynomial recognition algorithm for CCG-GTRC by extending the polynomial algorithm for the standard CCGs.

1 Introduction

In Japanese, as in other SOV languages, a sequence of NPs can form a conjunct as exemplified below.

(1) { John-ga / John-NOM } Mary-o / May-ACC , CONJ { Ken-ga / Ken-NOM } Naomi-o / Naomi-ACC tazuneta. / visited
"John visited Mary and Ken [visited] Naomi."

This type of 'non-traditional constituents' poses a problem to many grammar formalisms and parsers, including those specifically designed for Japanese, e.g., JPSG [Gunji, 1987] and JLE (based on finite-state syntax) [Kameyama, 1995]. Although these systems could be extended to cover the presented case, such extensions would not generalize to the wide range of non-traditional constituency.

Combinatory Categorial Grammar (CCG) has been proposed to account for non-traditional constituency in various areas of syntax [Ades and Steedman, 1982, Dowty, 1988, Steedman, 1985, Steedman, 1996] and also in the related areas of prosody, information structure, and quantifier scope [Prevost and Steedman, 1993, Hoffman, 1995, Park, 1995]. The mechanisms independently motivated to cover the wide range of non-traditional constituency can also provide an analysis for the NP-NP sequences in (1) as follows:

(2)

John-ga	Mary-o	
NP	NP	
$\Downarrow$	$\Downarrow$	type raising
$S/(S\backslash NP)$	$\underline{(S\backslash NP)}/((S\backslash NP)\backslash NP)$	functional composition (underlined portions are cancelled)
$S/((S\backslash NP)\backslash NP)$		

Informally, the NPs are assigned *higher-order function* categories associated with the basic category NP, and these functions can compose to derive another function category representing the NP-NP sequence. The two instances of such a category can then be coordinated and take the transitive verb category, $(S\backslash NP)\backslash NP$, as the argument to derive the category S.

A similar type of constituents can also be formed of NPs extracted from different levels of embedding as in the following example:

*I am grateful to Mark Steedman for his numerous suggestions and comments. I would also like to thank Jason Eisner, B. Srinivas, David Weir, K. Vijay-Shanker, and the anonymous reviewers for their comments. The research was supported in part by NSF Grant Nos. IRI95-04372, STC-SBR-8920230, ARPA Grant No. N66001-94-C6043, and ARO Grant No. DAAH04-94-G0426.

(3) Japanese: Rinyouzai-wa natoriumu-ni, β syadanzai-wa koukan sinkei kei-ni,
kankei-no aru kouketuatu-no hito-ni kikimasu.
Gloss: {Diuretic-TOP sodium-DAT} & {β blocker-TOP sympathetic nervous system-DAT}
relevance-GEN exist hypertension-GEN person-DAT effective.
Translation: "Diuretic is effective for the person with hypertension related to sodium, and β blocker
[is for the person with hypertension related] to sympathetic nervous system."

This is a simplified version of the sentence taken from the text actually parsed in the experiment discussed below. The original sentence includes three conjuncts, a few adverbs, and another level of embedding. The underlined part is another instance of non-traditional constituent, which includes an extraction from the relative clause. Its structure is schematically shown as follows:

(4) [t_1 hypertension$_2$-GEN person-DAT effective.]
[t_2 t_3 relevance-GEN exist]

Due to functional composition, the category $(S\backslash NP)\backslash NP$, identical to the transitive verb category, can be derived for the above phrase with two extraction sites t_1 and t_3.[1] As in the first example, the NP-NP sequences at the sentence-initial position receive the category $S/((S\backslash NP)\backslash NP)$ and then the sentence category is derived after the combination with the category $(S\backslash NP)\backslash NP$.

Assuming that the competence grammar does not place a bound on the levels of embedding [Miller and Chomsky, 1963], we may have unboundedly-many extractions [cf. Becker et al., 1991]. Since no systematic constraint has been identified for the bound on the composition of such extracted constituents, we also assume that these constituents can compose without a limit, potentially resulting in an unboundedly-long NP sequence.[2] As in the case of embedding, the degraded acceptability of long sequences can be attributed to performance issues. These assumptions calls for an infinite set of type-raised categories such as $(S\backslash X_n...\backslash X_1)/((S\backslash X_n...\backslash X_1)\backslash NP)$ associated with NP. We capture this polymorphic situation by using variables as in $T/(T\backslash NP)$.[3] Although CCGs have also been motivated by psychological aspects, we will be concerned only with syntactic aspects in this paper.

Formal properties of the standard CCGs (CCG-Std) not involving variables are relatively well-studied. CCG-Std is proved weakly-equivalentto Head Grammars (HG), Linear Index Grammars (LIG), and Tree Adjoining Grammars (TAG), known as "mildly context-sensitive grammars" [Vijay-Shanker and Weir, 1994]. CCG-Std is also shown to be polynomially parsable [Vijay-Shanker and Weir, 1990, Vijay-Shanker and Weir, 1991, Vijay-Shanker and Weir, 1993].[4] But, unrestricted use of variables can destroy these properties. For example, Hoffman [1993] showed that a grammar involving categories of the form $(T\backslash x)/(T\backslash y)$ can generate a language $a^n b^n c^n d^n e^n$, which is no longer equivalent to LIGs. The use of variables in the coordination schema "x^+ **conj** $x \rightarrow x$" is also believed to generate a language beyond LIG's power, namely $(wc)^n$ [Weir, 1988]. This paper is concerned with the other aspect: polynomial parsability of the CCGs with a generalized class of type-raised categories (CCG-GTRC) involving variables, from both practical and theoretical points of view. Related questions about the generative power of CCG-GTRC are addressed in [Komagata, 1997b, Komagata, 1997c].

For the practical side, we demonstrate that our CKY-style parser for Japanese appears to run in polynomial time once *spurious ambiguities* (multiple derivations for the equivalent semantics) are eliminated by equivalence check and *genuine ambiguities* (e.g., attachment ambiguity) are excluded. For the theoretical side, we present a worst-case polynomial recognition algorithm, by extending the polynomial algorithm proposed for CCG-Std by Vijay-Shanker and Weir [1990]. We will also observe that practical and theoretical polynomial results are due to distinct factors and thus the theory and practice do not directly correspond for the present case.

This paper is organized as follows: Section 2 introduces Generalized Type-Raised Categories and defines CCG-GTRC. Section 3 analyzes the performance of the practical parser through an experiment after a discussion about spurious ambiguity elimination. Section 4 presents the theoretical worst-case polynomial recognition algorithm for CCG-GTRC.

[1] The use of trace t is for illustration purposes only. The current approach does not assume the notion of gap or movement as the theories which employ trace.

[2] This assumption needs to be examined more carefully and is left for future research [cf. Joshi et al., 1994]. Since this paper assumes that the scope of coordination and adverb modificatioin is bounded, the unbounded composition of type-raised category actually calls for a motivation involving information structure [Komagata, 1997a].

[3] It is also possible to dynamically type-raise categories as needed. We take lexical approach to avoid the procedural aspects associated with dynamic type raising.

[4] This situation can be contrasted with Lambek calculus. For example, König [1994] showed an exponential parsing algorithm for the calculus, which can be improved to a polynomial one by setting a bound on the number of extractions.

2 CCGs with Generalized Type-Raised Categories

CCG-GTRC involves the class of *constant categories* (Const) and the class of *Generalized Type-Raised Categories* (GTRC).

A constant (derivable) category c can always be represented as $F|a_n...|a_1$ where F is an atomic *target* category and a_i's with their directionality are *arguments*. We will use "$A,...,Z$" for atomic, constant categories, "$a,...,z$" for possibly complex, constant categories, and '|' as a meta-variable for directional slashes $\{/,\backslash\}$. Categories are in the "result-leftmost" representation and associate left. Thus we usually write $F|a_n...|a_1$ for $(...(F|a_n)...|a_1)$. We will call "$|a_i...|a_j$" a *sequence* (of arguments). The length of a sequence is defined as $\left||a_i...|a_1\right| = i$ while the nil sequence is defined to have the length 0. Thus an atomic constant category is considered as a category with the target category with the nil sequence. We may also use the term 'sequence' to represent an ordered set of categories such as "$c_1,...,c_2$" but these two uses can be distinguished by the context. The standard CCGs (CCG-Std) solely utilize the class of Const.

GTRC is a generalization of *Lexical Type-Raised Category* (LTRC) which has the form $\overset{T}{\mathsf{T}}\left\langle\begin{smallmatrix}/\\\backslash\end{smallmatrix}\left(\mathsf{T}\left\rangle\begin{smallmatrix}\backslash\\/\end{smallmatrix}a\right)\right.|b_i...|b_1$ associated with a lexical category $a|b_i...|b_1$ where $\overset{T}{\mathsf{T}}$ is a variable over categories with the atomic target category T. The target indication may be dropped when it is not crucial or all the atomic categories are allowed for the target. We assume the order-preserving form of LTRC using the following notation. '$\begin{smallmatrix}/\\\backslash\end{smallmatrix}$' and '$\begin{smallmatrix}\backslash\\/\end{smallmatrix}$' indicate that either set of slashes in the upper or the lower tier can be chosen but a mixture such as '/' and '/' is prohibited.[5] GTRC is defined as having the form of $\mathsf{T}\left\langle\begin{smallmatrix}/\\\backslash\end{smallmatrix}(\mathsf{T}\underbrace{|a_m...|a_2\left\rangle\begin{smallmatrix}\backslash\\/\end{smallmatrix}a_1\right.}_{\textit{inner sequence}})\underbrace{|b_n...|b_1}_{\textit{outer sequence}}$ resulting from compositions of LTRCs where $m \geq 1$, $n \geq 0$, and the directional constraint is carried over from the involved LTRCs. When the directionality is not critical, we may simply write a GTRC as $\mathsf{T}|(\mathsf{T}|a_m...|a_2|a_1)|b_n...|b_1$. For $gtrc = \mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_1$, we define $|gtrc| = n+1$, ignoring the underspecified valency of the variable. Note that the introduction of LTRCs in the lexicon is non-recursive and thus does not suffer from the problem of the overgeneration discussed in [Carpenter, 1991].

These categories can be combined by combinatory rule schemata. Rules of (forward) "generalized functional composition" have the following form:

(5) $\underset{\text{functor category}}{x/\underline{y}} \quad \blacktriangleright^k \quad \underset{\text{input category}}{\underline{y}|z_k...|z_1} \quad \longrightarrow \quad \underset{\text{result category}}{x|z_k...|z_1}$

The integer 'k' in this schema is bounded by k_{max}, specific to the grammar, as in CCG-Std. Rules of functional application, "$x/y \blacktriangleright^0 y \rightarrow x$", can be considered as a special case of (5) where the sequence z_i's is nil. The index k may be dropped when no confusion occurs. We say " '$x/\underline{y} \blacktriangleright \underline{y}|z_k...|z_1$' *derives* $x|z_k...|z_1$", and "$x|z_k...|z_1$ *generates* the string of nonterminals '$x/y, y|z_k...|z_1$' or the string of terminals '$\mathtt{ab}$' " where the terminals $\mathtt{a}$ and $\mathtt{b}$ are associated with x/y and $y|z_k...|z_1$, respectively. The case with backward rules involving '$\blacktriangleleft$' is analogous.

Inclusion of GTRCs calls for thorough examination of each combinatory case depending on the involved category classes. A summary is given in Table 1. The categorial representations for a few relevant cases will be shown below, but the details are left to the accompanying technical report [Komagata, 1997a].

The three cases indicated by '$*$' in Table 1 introduce categories which are neither Const nor GTRC due to the residual variables. For example, observe Case 3c below.

(6) Combination type of "Const$\blacktriangleright$GTRC" where $k > |\text{input}|$ (and $k > 2$):

$$a/\underline{b} \blacktriangleright^k \underline{\mathsf{T}_0}\overbrace{|\mathsf{T}_1|(\mathsf{T}_0|\mathsf{T}_1|c_m...|c_1)|d_{k-2}...|d_1}^{k} \longrightarrow a\underset{\substack{\uparrow\\\text{residual}}}{|\mathsf{T}_1}|(b|\mathsf{T}_1|c_m...|c_1)|d_{k-2}...|d_1$$ [6]

This is an unintended, accidental use of functional composition. The closure of the system must be maintained by excluding these cases.[7] We are now in the position to define the class of CCG-GTRC:

Definition 1 A CCG-GTRC is a six tuple $(V_N, V_T, S, \mathcal{T}, f, R)$ where

- V_N is a finite set of nonterminals (atomic categories)
- V_T is a finite set of terminals (lexical items, written as $\mathtt{a},...,\mathtt{z}$)

[5] For a related discussion, see [Steedman, 1991].

[6] T could also be decomposed into $\mathsf{T}_0|\mathsf{T}_k...|\mathsf{T}_1$ for a larger k but all these share the same characteristics with the above scheme.

[7] This condition is particularly important for implementation since the residual variables can behave beyond our imagination.

	Case	Functor cat		Input cat		Result cat		
		Class	Outer seq	Class	$k \lessgtr \lvert\text{input}\rvert$	Class	Residual variable	Unbounded const argument
	1	Const	-	Const	$\leq$	Const	no	no
	2*a*	GTRC	yes	Const	$\leq$	GTRC	no	no
	2*b*	GTRC	no	Const	$\leq$	Const	no	no
	3*a*	Const	-	GTRC	$<$	Const	no	no
	3*b*	Const	-	GTRC	$=$	Const	no	possible
*	3*c*	Const	-	GTRC	$>$	neither	yes	possible
	4*a*	GTRC	yes	GTRC	$<$	GTRC	no	no
	4*b*	GTRC	yes	GTRC	$=$	GTRC	no	possible
*	4*c*	GTRC	yes	GTRC	$>$	neither	yes	possible
	4*d*i	GTRC	no	GTRC	$<$	GTRC	no	no
	4*d*ii	GTRC	no	GTRC	$<$	Const	no	no
	4*e*	GTRC	no	GTRC	$=$	GTRC	no	no
*	4*f*	GTRC	no	GTRC	$>$	neither	yes	possible

Table 1: Combinatory Cases for CCG-GTRC

- S is a distinguished member of V_N
- $\mathcal{T}$ is a countable set of variables[8]
- f is a function that maps elements of V_T to finite subsets of "Const $\cup$ LTRC"
- R is a finite set of rule instances summarized in Table 1 (except for those with '$*$').[9]

3 Progress Towards a Practical Parser for CCG-GTRC

This section investigates the performance of the experimental parser and demonstrates that it runs polynomially in practice. Since any practical parser for CCGs (or categorial grammars in general) requires some kind of spurious ambiguity elimination, we start with a discussion about spurious ambiguity elimination techniques.

3.1 Spurious Ambiguity Elimination

Let us first define different types of ambiguities referred to in this paper:

(7) *a.* Categorial ambiguity: Availability of multiple categories (lexical/derivational), e.g., $\{NP, S\backslash NP\}$.

b. Spurious ambiguity: Multiple derivations of the same category which are semantically *equivalent* [Wittenburg, 1986], e.g., "John visited Bill." has two derivations of S (left and right branching) in CCG with the identical semantics '(*visited bill john*)'.

c. Genuine ambiguity:

(i) Lexico-semantic ambiguity: Multiple semantic assignments to a single lexical category:

(ii) Attachment ambiguity: Multiple derivations of the same category with *distinct* semantics. E.g., PP attachment.

Note that by 'semantics', we assume predicate-argument structure (a kind of logical form) as the input to the semantic module.

Since spurious ambiguity is often accused of as the source of inefficient parsing, CCG parsers must implement some means of spurious ambiguity elimination. We review three classes of approaches: (i) syntactic, (ii) semantic, and (iii) those which do not belong to the previous two.

The syntactic approaches eliminate 'spurious derivations' which are not 'the normal form'. Hendriks [1993] and König [1994] worked on Lambek calculus which does not have functional composition as a primitive rule. Hepple and Morrill

[8] Each instance of GTRC must be assigned a new variable when the GTRC is instantiated at a particular string position in order to avoid unintended variable binding.

[9] Due to the introduction of GTRC, the rule instances may involve variables even at the first argument of the functor category and at the input category.

[1989] cover a subset of the current formalism but do not have the mixed instances of function composition nor type raising. Eisner [1996] covers a wider range of CCGs but the case including type raising remains to be shown correct. Labeled deduction [Morrill, 1994] has a means to interpret semantics syntactically but normal form deduction must be adjusted.

Karttunen [1986] proposed the following semantic method. For each new derivation, discard it if its semantics is *equivalent to* (or mutually subsumes) that of some entry with the same category already in the cell.[10] This directly enforces the definition of spurious ambiguity and does not depend on the syntax. Note that 'equivalence' depends on the semantic representation [Thompson, 1991]. For the case where the semantics is represented in λ-calculus, equivalence is not computable [Paulson, 1991]. For the case of feature structure, equivalence is defined as alphabetic variants and characterized by the isomorphism between the structures [Carpenter, 1992]. Our case corresponds to the latter although the equivalence check on predicate-argument structures are term unification rather than graph unification for the feature structure.

Among the third type, Pareschi and Steedman [1987] employed a control-based mechanism but the published algorithm has been shown to be incomplete [Hepple, 1987]. Wittenburg and Wall [1991] use a compilation scheme but the crucial notion of flexible constituency is compromised.

To be precise, the definition of spurious ambiguity for syntactic methods is not the same as our definition given above. Since syntactic methods are insensitive to semantics, they cannot distinguish genuine (attachment) and spurious ambiguities and eliminate both of these. With semantic equivalence check, we can adjust the way the genuine ambiguity is handled. While equivalence check in its original form is sensitive to genuine ambiguity, equivalence check applied only to category (ignoring semantics) is insensitive to genuine ambiguity much like syntactic methods. We refer to these two cases as (i) *category+semantics* and (ii) *category-only*. Both cases are explored in the experiment in the following subsection.

Among the presented methods, the only proposal immediately compatible with the present formalism is Karttunen's semantic equivalence check. Let us review some arguments against this approach. Eisner [1996] argued that a sequence of categories exemplified by "X/X ... X ... $X\backslash X$" can slow down the parser exponentially. Note that we assume that X/X and $X\backslash X$ are 'modifiers' of X with distinct semantics. But this is an instance of genuine ambiguity, a problem to any parser. Note that for syntactic methods, this appears as spurious ambiguity and the semantic processing is simply left for a later stage. Wittenburg [1987] objected to the cost of equivalence check. But an equivalence check (for our semantics) is inherently easier than the general case of subsumption check, the latter requiring the *occurs check* for soundness [Pereira and Shieber, 1987]. Hepple and Morrill [1989] have raised another objection. While syntactic methods detect spurious ambiguities before deriving a result, equivalence checking needs to compare the derived result with every entry in the current table cell. However, the cost associated with the semantic method depends on how many genuinely ambiguous entries are in the cell but not on the number of spuriously-ambiguous entries (they are eliminated as soon as they are derived and do not accumulate). This does not introduce additional complexity specific to spurious ambiguity check. Further, once semantics is involved, it is not possible to distinguish between spurious and genuine ambiguities unless we actually check the involved semantics.

3.2 Experiment

We now look at the results of a pilot experiment done on Sun Ultra E4000 2x167MHz Ultraspacs with 320MB memory running SunOS 5.5.1. The program (100KB approx., about a half is the grammar) was written in Sicstus Prolog Ver. 3 and CPU time was measured by Sicstus' built-in predicate `statistics`.

We parsed 22 contiguous sentences (6 paragraphs) in Japanese in a section arbitrarily chosen from "Anata-no Byouin-no Kusuri (Your Hospital Drugs) 1996" by Tadami Kumazawa and Ko-ichi Ushiro. Japanese is chosen because the current work is a part of a larger project of medical database interface using this language. The romanized sentences are partially-segmented to the word level but the verb inflection and suffixes are considered as a part of the word. The average number of words in a sentence is 20 and the longest sentence contains 41 words. The sentences are realistically difficult, and include complex clauses (relative and complement), coordination (up to 4 conjuncts), nominal/verbal modifications (adjectives/adverbs), scrambling, and verb argument dropping.

The parser is based on a CKY algorithm [Aho and Ullman, 1972] equipped with Karttunen's equivalence check for spurious ambiguity elimination but without the worst-case polynomial algorithm introduced in the next section.[11] LTRCs are assigned to words by lexical rules and GTRCs are restricted to unidirectional forms. Coordination is handled by

[10] Shieber [1986] contains a detailed discussion of subsumption.

[11] Earlier applications of a CKY-style algorithm to CCG parsing include [Pareschi and Steedman, 1987].

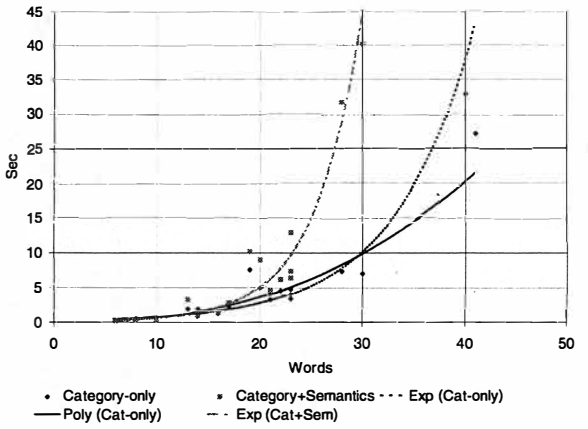


Figure 1: Basic Data Set (linear scale)

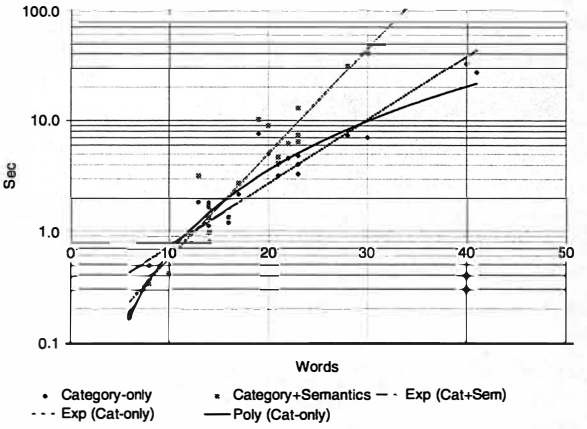


Figure 2: Basic Data Set (log scale)

special trinomial rules [Steedman, 1996] with a few categorial features added to limit the coordination involving multiple constituents only to the left-branching structure. Verb argument dropping is handled by lexical rules which change the verb valency. Morphological analysis is a complete substring match and the results are dynamically 'asserted' among the code. About 200 lexical entries are asserted after parsing the 22 sentences. Currently, morphological analysis takes about 0.2 seconds per word on average and needs improvement. The output of a parse is an enumeration of the final result categories associated with the features and the semantics as seen below.

(8)
```
itumo 95mmHg o koeru baai_wa tiryou ga hituyoudesu.
always num(95mmHg) -ACC exceed in_case treatment [-NOM,-CONJv] necessary
  Cat:
    SS: s
    Fs: []
    PA: (in_case always((exceed num(95mmHg) $1)) (necessary treatment))
  Cat:
    SS: s
    Fs: []
    PA: always((in_case (exceed num(95mmHg) $2) (necessary treatment)))
CPU time: 280 ms  Elapsed: 320 ms  Words: 8  Solutions: 2
```

The unresolved pronoun is shown as '$\$n$' where $n \in \mathbb{N}$. The ambiguity regarding adverbial modification is left unresolved. The implementation has a simplified treatment of quantifiers and scope ambiguity too is left unresolved.

We consider the following two cases: (i) category-only and (ii) category+semantics. As we have discussed in the previous subsection, the application of equivalence check to the category-only case not only eliminates spurious ambiguities but also provide a result without genuine ambiguities.

Let us start with the analysis of the category-only case.[12] This case corresponds to the situation involving syntactic methods and also the polynomial algorithms introduced in the next section. The results are shown in Figure 1 (linear scale) and 2 (log scale). Both exponential ($y = 0.1963 \times 1.14^n$) and polynomial ($y = 0.002 \times n^{2.496}$) regression lines calculated by Microsoft Excel are provided. Although it is often easier to fit either an exponential or a polynomial curve on a log-scale graph, the data do not seem to be enough for such a conclusion. To see how the experiment might extend to the case with words longer than 45 words, we parsed pseudo-long sentences. That is, some of the test sentences are conjoined to form long sentences. Although these are semi-fabricated data, most long sentences are in fact the results of coordination. Thus natural data are expected to behave similarly rather than differently from our pseudo-long sentences. The results are shown in Figures 3 and 4. The polynomial curves ($y = 0.0017 \times n^{2.5616}$) seem to represent the data better than the exponential curve ($y = 0.4375 \times 1.09^n$), especially on the log-scale graph. Since the data is sparse, we do not attempt to obtain a significant statistic analysis for these and simply eye-fit the data. With these qualifications, we conclude that the performance appears no worse than n^3. The result also shows that categorial ambiguities still present in the parses

[12] Category-only is the case also corresponding to the spurious ambiguity check of syntactic methods.

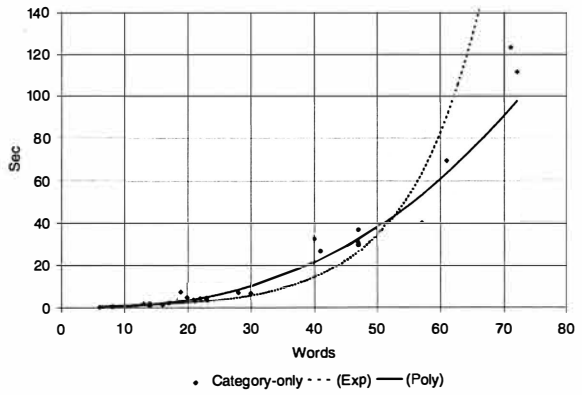


Figure 3: Extended Data Set (linear scale)

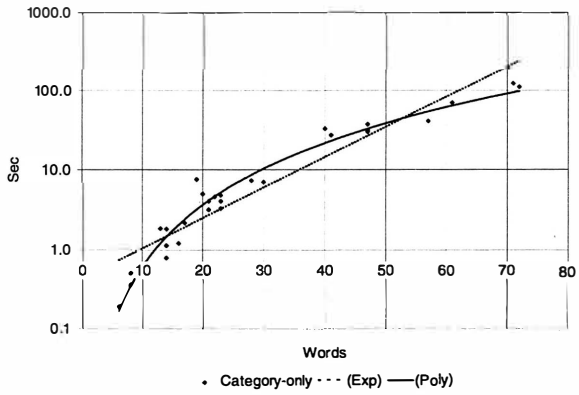


Figure 4: Extended Data Set (log scale)

are in practice within this bound.

A few remarks are in order. We compared our results with the following experiment to see how the figures stand. Tanaka and Ueki [1995] report that the LR-based syntactic analysis of a 19-word sentence in Japanese took 3.240sec.[13] The range of CPU times for our sentences with 19-23 words is between 3 to 8sec (category-only case). The performance of our parser seems to be within a comparable range.

Another point is that the effect of spurious ambiguity check is immediate. Without the check, only the sentences with 10 or fewer words were parsed. Under this condition, the maximum number of cell entries easily exceeds 300 for longer sentences, which resulted in out-of-space errors. We thus confirmed that the exponential effect of spurious ambiguity is well controlled by semantic equivalence check.

Since one of the advantages of CCG parsers is the ability to derive semantics along with syntactic structure, the results of the category+semantics case is of special interest. The situation naturally looks quite different. The exponential regression line $t = 0.0638 \times 1.24^n$ (Figures 1 and 2) seems to fit the data closely. In fact, the two longest sentences with 40 and 41 words results in out-of-space errors. Since spurious ambiguities are eliminated by equivalence check and categorial ambiguity is only polynomial as shown in the category-only experiment, the exponential slow down is due exclusively to genuine ambiguity. Genuine ambiguity is a major problem for our parser as it is for any parser.

As a potential solution to genuine ambiguity problem, Dörre [1997] recently reported a polynomial algorithm to build shared semantic structures from shared syntactic structures. Unfortunately, this technique may not be directly applicable to our parser because semantic equivalence check is now required to traverse the shared structures. It is not clear if the traversal can be done in polynomial time. This concern is shared by the situation of applying structure sharing technique to conceptual dependency [Bröker et al., 1994].

4 Worst-Case Polynomial Recognition Algorithm

Although the implemented parser runs efficiently (the category-only case), its worst-case performance is still exponential due to categorial ambiguity. This section presents a worst-case polynomial recognition algorithm for a subclass of CCG-GTRC (Poly-GTRC) by extending the polynomial algorithm of Vijay-Shanker and Weir [1990] for CCG-Std (Poly-Std). We will observe below that the crucial property of CCG-Std employed by Poly-Std can be extended to the subclass of CCG-GTRC with an additional condition. Let us start with a brief review of the intuition behind Poly-Std and then move on to Poly-GTRC. Note that Poly-Std has the second stage of structure building but we concentrate on the more critical part of recognition.

[13]The word count is based on our criteria. Other details are ignored for now.

4.1 Polynomial Algorithm for CCG-Std

First, observe the following properties of CCG categories:

(9) *a*. The length of a category in a cell can grow proportionally to the input size.

b. The number of categories in a cell may grow exponentially to the input size.

For example, consider a lexicon $f = \{(\mathtt{a}, S/NP/S),\ (\mathtt{a}, S/PP/S)\}$. Then, for the input "$\underbrace{\mathtt{a}....\mathtt{a}}_{n}$", the top CKY-cell includes 2^n combinations of categories like $S \underbrace{\left\{ \begin{smallmatrix} /NP \\ /PP \end{smallmatrix} \right\} ... \left\{ \begin{smallmatrix} /NP \\ /PP \end{smallmatrix} \right\}}_{n} /S$ derived by functional composition. Thus, we have exponential worst-case performance with respect to the input size.

The idea behind Poly-Std is to store categories as if they were some kind of linked list. Informally, a long category $F|a_n...|a_2|a_1$ is stored as 'F this portion is *linked* in a cell $[p,q]$ with *index* $|a_2$ $|a_1$'. A crucial point here is that the instances of target category F and arguments a_2 and a_1 are *bounded*. We will come back to this point in the next subsection. The pair $[p,q]$ can be represented as a n^2 matrix. Thus by setting up n^2 subcells in each CKY-table cell, we can represent a category in a finite manner.

The effectiveness of such a representation comes from the fact that CCG rule application does not depend on the entire category. Namely, in order to verify "$F \underbrace{|a_n...|a_2}_{\bigstar}\ \underline{|a_1} \blacktriangleright^k \underline{b_0}|b_k...|b_1 \rightarrow F \underbrace{|a_n...|a_2}_{\bigstar}\ |b_k...|b_1$", the sequence marked by '$\bigstar$' does not need to be examined. Thus, for the functor category, we only need to check F and a_1 available in the current cell. In addition, since b_0 must be unified with a bounded a_1, and k is also bounded by k_{max}, the entire input category is bounded and thus can be stored in the current cell. Therefore, the proposed representation does not slow down this type of process. When the result category exceeds a certain limit, we leave the excessive portion right in the original cell and set up a link to it.

One complication is that when an argument (e.g., a_1 in the above example) is canceled, we may have to restore a portion of the category from the linked cell (as the 'index' for the cell is required). We need to scan the linked cells and find the categories with the same index from n^2 subcells. Even though there may be multiple such categories, all of them can be restored in one of n^2 subcells associated with the result category. This case dominates the computational complexity but can be done in $O(n^3)$. Since this is inside i, j, k of CKY-style loop, the overall complexity is $O(n^6)$.

4.2 Polynomial Algorithm for CCG-GTRC

We first note that there are cases where a crucial property of CCG-Std cannot be maintained in CCG-GTRC. The property is that arguments of derived categories are bounded. Although there might be a polynomial algorithm for CCG-GTRC which does not depend on this property, we pursue a straightforward extension of Poly-Std with an additional condition on the rules.[14]

The problematic cases are 3*b* and 4*b* in Table 1. As we have motivated in Introduction, the inner sequence of GTRC must be allowed to grow unboundedly long (otherwise, the resulting grammar becomes equivalent to CCG-Std). Then an argument can grow unboundedly long as exemplified by Case 3*b* below.

(10) Combination type of "Const▶GTRC" where $k = |\text{input}|$ (and $k \geq 1$):

$$a/\underline{b} \blacktriangleright^k \underline{\top} \overbrace{|(\top|c_m...|c_1)|d_{k-1}...|d_1}^{k} \longrightarrow a|(b \underbrace{|c_m...|c_1)}_{\text{unbounded}}|d_{k-1}...|d_1$$

But we do not want to exclude Cases 3*b* and 4*b* entirely since a case involving a sentential adverb may be possible as follows: "$S/S \blacktriangleright \top|(\top|A|B)$". The present approach is to place the following condition on the rule application:

(11) **Bounded Argument Condition**: The input GTRCs in the cases 3*b* and 4*b* must be instantiated.

This is in a sense analogous to placing a bound k_{max} on functional composition instantiating the input category. By this condition, we have the following property:

(12) The set of arguments of constant categories and the set of arguments of the inner and outer sequences of GTRC are all finite.

[14] The length of the argument is still bounded by $O(n)$ in CCG-GTRC since the only source of unboundedness is GTRC inner sequences. If every argument can be represented in some finite manner with link information similar to the one used for the Poly-Std, polynomial recognition might be possible.

That is, by considering the inner sequence at the same level as the outer sequence of a GTRC or the arguments of a constant category, we can maintain the property analogous to the one found for CCG-Std. Note that the Bound Argument Condition is not implemented in our parser. The inner sequence of GTRC does not grow unboundedly in practice and thus the arguments of categories do not grow unboundedly either.

Now, consider the subclass of CCG-GTRC constrained by the Bounded Argument Condition. In the rest of this section, we will concentrate on this subclass.

Poly-GTRC is an extension to Poly-Std. The basic organization of the algorithm is analogous to Poly-Std. We use the same $n^2 \times n^2$ CKY-style table and a similar representation for constant categories. But we need to deal with GTRCs in polynomial time as well. First, let us examine two representative cases of rule applications since this reveals the necessary conditions for polynomial parsing.

The inner sequence of GTRC can grow as a result of Case 4*e* shown below:

(13) $\mathsf{T} \underset{\uparrow}{/} \underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underline{\mathsf{U}}|(\mathsf{U}|c_p...|c_1)|d_{k-1}...|d_1 \longrightarrow \mathsf{T}|(\mathsf{T}|a_m...|a_1|c_p...|c_1)|d_{k-1}...|d_1 \quad (k \geq 1)$

The only information needed to determine if the rule is applicable is the directionality of the slash indicated by '↑'. Thus we do not actually need to know the inner sequence of the functor or input categories. The inner sequence of the result GTRC can thus be represented as two links to the functor and input categories. This link information virtually encodes a kind of grammar for deriving the inner sequence and is thus considered as an application of structure sharing [Billot and Lang, 1989, Dymetman, 1997]. The outer sequence can be represented in a way similar to the argument of constant category. Although there may be exponentially-many GTRCs associated with each CKY cell, the number of cell entries is bounded by the link destinations of the inner sequence and the finite representation for the outer sequence.

Next, consider Case 2*b*.

(14) $\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underset{\underline{c_0|c_m...|c_1}}{\overset{c}{=}} |d_k...|d_1 \longrightarrow c_0|d_k...|d_1$

We need to show that the unification process of the underlined portions can be done in polynomial time. As the first approximation, consider this process as an iteration of (backward) functional application of the form "$\underline{a_i} \blacktriangleleft c_0|c_m...|\underline{c_i}$" for $i = 1$ to m where a_i and c_i are canceled. But recall that in general, we only store a finite portion of both the functor and the input categories in the current cell and the remaining information must be restored through the links. The restoration of the information could cost exponential time since there may be multiple links to lower locations at any point. Therefore it is crucial that we proceed from $i = 1$ to m so that no enumeration of all the instances of $c_i,...,c_1$ and $a_i,...,a_1$ in (14) is actually generated. The traversal of the link from c_1 and a_1 may introduce sets of categories C_i and $\mathcal{A}_i$ for each position of $i \geq 2$, as schematically shown below.

(15)
$$\begin{array}{ccccc} C_m & \cdots & C_2 & & \{c_1\} \\ \updownarrow & & \updownarrow & \longleftarrow & \updownarrow \\ \mathcal{A}_m & \cdots & \mathcal{A}_2 & & \{a_1\} \end{array}$$

Note that each set C_i and $\mathcal{A}_i$ are bounded. This is the crucial point we needed the Bounded Argument Condition. Now, suppose that an element in C_i is canceled with some elements in $\mathcal{A}_i$. We can proceed to the next set C_{i+1} where the elements in C_{i+1} are obtained by traversing the links from the canceled elements in C_i. Once we move from C_i to C_{i+1}, the history of cancellation can be forgotten, as in the case of iterative functional application in Poly-Std. Thus even though we have potentially exponential instances of $c_i,...,c_1$, the traversal of this side can be done step-by-step without suffering the exponential effect.

The traversal of $\mathcal{A}_i$'s is more challenging. The availability of a_i for cancellation with some c_i depends on the history of the cancellation of $a_{i-1},...,a_1$. Actually, it depends on the *tree structure* exactly encoded by the structure sharing technique. The 'GTRC recovery algorithm' will be introduced below to handle this situation in polynomial time.

The other cases are included in the technical report along with a more detailed description and a worked example of recovering GTRCs [Komagata, 1997a]. Through the examination, we conclude that the polynomial parsability of CCG-GTRC depends on recovery of GTRCs. We present the polynomial GTRC recovery algorithm in Figure 5.

The GTRC recovery algorithm takes advantage of the encoded shared structure, and utilizes an additional n^2 GTRC recovery table to restore possibly ambiguous GTRC derivations in polynomial time. The first stage (*table setup*) is to represent the derivational structure available in the CKY table in a slightly different way. Suppose the following partial CKY table starting from a GTRC in question:

Initialization:
- Create an n^2 GTRC recovery table, R

Table setup (Stage 1):
- For each cell (top-down) $O(n^2)$
  - For each entry (depending on the midpoint) $O(n)$
    - Restore the derivation info from CKY table and store the children in the appropriate cells $O(n^2)$

Recovery (Stage 2):
- For each cell in the bottom row (right-to-left) $O(n)$
  - For each entry $O(n)$
    - If there is a matching category in the target category set
      - Mark the current entry as 'success'

      Otherwise
      - Mark the current entry as 'fail'
    - Do **status percolation**

Status percolation (subprocedure):
- For each cell (bottom-up) $O(n^2)$
  - For each entry $O(n)$
    - For each parent $O(n^2)$
      - If the parent is marked as 'fail'
        - Mark the current entry as 'fail'

        Otherwise
        - If the current entry is the right branch *and* marked as 'fail' *and* all the right branch siblings are marked as 'fail,
          - Mark the parent as 'fail'
        - If the current entry is the left branch *and* marked as 'success'
          - Mark the parent as 'success'

Figure 5: GTRC Recovery Algorithm

(16) Partial CKY table:

	1	2	3	4	5
5	$T/(T\ldots\backslash B)_{[1,2]\blacktriangleright[3,5]}$				
4					
3			$T/(T\ldots\backslash B)_{[3,3]\blacktriangleleft[4,5]}$		
2	$T/(T\backslash A)_{[1,1]\blacktriangleright[2,2]}$			$T\backslash(T/C)_{[4,4]\blacktriangleleft[5,5]}$	
1	$T/(T\backslash A)/D$	D	$T/(T\backslash B)$	E	$T\backslash(T/C)\backslash E$

$T/(T\ldots\backslash B)_{[1,2]\blacktriangleright[3,5]}$ represents the derivation "$T/(T\backslash A)_{[1,1]\blacktriangleright[2,2]} \blacktriangleright T/(T\ldots\backslash B)_{[3,3]\blacktriangleleft[4,5]} \rightarrow T/(T\ldots\backslash B)$" at the designated string positions. A GTRC recovery table can be used to store the same derivational structure with the bottom row corresponding to *the order of the inner sequence* of the GTRC rather than the string position. This is the order to process the inner sequence for C_i-$\mathcal{A}_i$ comparison shown in (15). Since GTRC recovery process only concerns with the inner sequence of the GTRCs, the recovery table may have a dimension smaller than the corresponding portion of the CKY table as seen in the following example:

(17) GTRC recovery table:

	3	2	1
3	$T/(T\ldots\backslash B)_{[1,1]\blacktriangleright[2,3]}$		
2		$T/(T\ldots\backslash B)_{[2,2]\blacktriangleright[3,3]}$	
1	$T/(T\backslash A)$	$T\backslash(T/C)$	$T/(T\backslash B)$

The categorial ambiguities originally aligned at string positions are now aligned in the order of processing.

In the second stage (*recovery stage*), the comparison with the target categories is done while the above-mentioned dependency among LTRCs in the bottom row is checked. The comparison proceeds from right to left in the bottom row. The decision on the cancellation of the argument under consideration, a_i, depends on (i) if it is unifiable with some target category (in C_i) *and* (ii) if the corresponding sequence to the right of a_i was successfully canceled. This latter condition

can be checked by observing the status of the first right branch from the current position since all the processes up to that point must have been completed. For the later processing (for the positions to the left), the success/failure status of the current category must also be percolated to the relevant higher nodes (*status percolation*). The total complexity turns out to be a rather daunting $O(n^{10}) = O(\underset{i,j,k}{n^3} \times \underset{\text{recovery}}{n^2} \times \underset{\text{percolation}}{n^5})$.

The extremely-high cost of GTRC recovery process seems to be related to the following conjecture: the generative power of this subclass of CCG-GTRC is greater than CCG-Std. A more restricted version of CCG-GTRC has been shown to be weakly equivalent to CCG-Std [Komagata, 1997b, Komagata, 1997c]. This subclass restricts the directionality of GTRC to the unidirectional case and only deals with GTRCs without outer sequence, thus allowing only the GTRCs of the form $\mathrm{T}\left\langle (\mathrm{T}\right\rangle a_m \ldots \left.\right\rangle a_1)$. In this case, the inner sequence and the string positions agree on the number and the order of the arguments and no GTRC recovery algorithm is needed. Realistically, most languages seem to restrict the use of GTRCs in some sense. For example, the Japanese grammar used in our experiment restricts GTRCs to the unidirectional variety but still with outer sequence.

5 Conclusion

The practical and the theoretical polynomial results are due to distinct factors. The former comes from a practical bound on the number of cell entries after spurious ambiguity elimination, showing CFG-like behavior. The latter (for both Poly-Std and Poly-GTRC) is achieved by efficiently representing and processing the potentially exponentially-many entries in a cell. This is possible even with the presence of spurious/genuine ambiguities. But it turns out that what the polynomial algorithms do is to eliminate a possibility which in practice rarely occurs. The additional cost for Poly-Std/GTRC of managing n^2 subcells and links to cover all the cases of exponential factors including spurious/genuine ambiguities is thus considered as overkill for the practical case. Although it may be possible to add spurious ambiguity check to Poly-Std/GTRC, we are better off with a simple CKY-style parser with equivalence check, without the overhead of the Poly-Std/GTRC.

The above conclusion naturally remains qualified by the small scale of the experiment reported here. But, the test sentences are reasonably representative and relatively challenging. They vary in sentence length and complexity and span the space we may typically encounter. With additional data, it is reasonable to expect that the missing points will be filled and statistic significance will be obtained. It is also reasonable to believe that the experiment with pseudo-long sentences characterizes the kind of complexity that will be found in natural data.

Few grammars or parsers designed for a configurational language such as English perform well on realistic fragments of a language like Japanese. One of the reasons may be abundance of non-traditional constituents found in SOV languages. Our parser, in contrast, demonstrates a reasonable performance although the grammar is basically an instantiation of the same framework applied for English [Steedman, 1996]. This strongly suggests that a framework which can analyze non-traditional constituents as a part of the competence property has an advantage in dealing with cross-linguistic data.

References

Anthony Ades and Mark J. Steedman. 1982. On the Order of Words. *Linguistics and Philosophy*, 4.

Alfred V. Aho and J. D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling. Vol. 1: Parsing.* Prentice-Hall.

Tilman Becker, Aravind Joshi, and Owen Rambow. 1991. Long-Distance Scrambling and Tree Adjoining Grammars. In *EACL5*, 1991.

Sylvie Billot and Bernard Lang. 1989. The Structure of Shared Forests in Ambiguous Parsing. In *ACL27*, 1989.

Norbert Bröker, Udo Hahn, and Susanne Schacht. 1994. Concurrent Lexicalized Dependency Parsing: The ParseTalk Model. In *COLING-94*, 1994.

Bob Carpenter. 1991. The Generative Power of Categorial Grammars and Head-Driven Phrase Structure Grammars with Lexical Rules. *Computational Linguistics*, 17.

Robert L. Carpenter. 1992. *The Logic of Typed Feature Structures.* Cambridge University Press.

Jochen Dörre. 1997. Efficient Construction of Underspecified Semantics under Massive Ambiguity. In *ACL35/EACL8*, 1997.

David Dowty. 1988. Type Raising, Functional Composition, and Non-Constituent Conjunction. In Richard Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures.* D. Reidel.

Marc Dymetman. 1997. Charts, Interaction-Free Grammars, and the Compact Representation of Ambiguity. In *IJCAI-97*, 1997.

Jason Eisner. 1996. Efficient Normal-Form Parsing for Combinatory Categorial Grammar. In *ACL 34*, 1996.

Takao Gunji. 1987. *Japanese Phrase Structure Grammar: A Unification-Based Approach.* D. Reidel.

Herman Hendriks. 1993. *Studied Flexibility: Categories and Types in Syntax and Semantics.* ILLC Dissertation Series.

Mark Hepple and Glyn Morrill. 1989. Parsing and Derivational Equivalence. In *EACL 4*, 1989.

Mark Hepple. 1987. Methods for Parsing Combinatory Grammars and the Spurious Ambiguity Problem.

Beryl Hoffman. 1993. The Formal Consequences of Using Variables in CCG Categories. In *ACL31*, 1993.

Beryl Hoffman. 1995. *The Computational Analysis of the Syntax and Interpretation of "Free" Word Order in Turkish.* PhD thesis, University of Pennsylvania.

Aravind K. Joshi, Tilman Becker, and Owen Rambow. 1994. Complexity of Scrambling: A New Twist to the Competence - Performance Distinction. In *3e Colloque International sur les grammaires d'Arbres Adjoints*, 1994.

Megumi Kameyama. 1995. The Syntax and Semantics of the Japanese Language Engine. In Reiko Mazuka and Noriko Nagai, editors, *Japanese Sentence Processing.* Lawrence Erlbaum.

Lauri Karttunen. 1986. Radical Lexicalism. Technical report, CSLI.

Nobo Komagata. 1997a. Efficient Parsing for CCGs with Generalized Type-Raised Categories. Technical report, University of Pennsylvania.

Nobo Komagata. 1997b. Generative Power of CCGs with Generalized Type-Raised Categories. In *ACL35/EACL8 (Student Session)*, 1997b.

Nobo Komagata. 1997c. Generative Power of CCGs with Generalized Type-Raised Categories. Technical report, University of Pennsylvania.

Esther König. 1994. A Hypothetical Reasoning Algorithm for Linguistic Analysis. *J. Logic and Computation*, 4(1):1–19.

George Miller and Noam Chomsky. 1963. Finitary Models of Language Users. In R.R. Luce et al., editors, *Handbook of Mathematical Psychology, Vol. II.* John Wiley and Sons.

Glyn V. Morrill. 1994. *Type logical grammar: categorial logic of signs.* Kluwer.

Remo Pareschi and Mark Steedman. 1987. A Lazy Way to Chart-Parse with Categorial Grammars. In *ACL25*, 1987.

Jong C. Park. 1995. Quantifier Scope and Constituency. In *ACL33*, 1995.

Lawrence C. Paulson. 1991. *ML for the Working Programmer.* Cambridge University Press.

Fernando C.N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural-Language Analysis.* CSLI.

Scott Prevost and Mark Steedman. 1993. Generating Contextually Appropriate Intonation. In *EACL6*, 1993.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammar.* CSLI.

Mark J. Steedman. 1985. Dependency and Coordination in the Grammar of Dutch and English. *Language*, 61:523–56.

Mark Steedman. 1991. Type-Raising and Directionality in Combinatory Grammar. In *ACL29*, 1991.

Mark Steedman. 1996. *Surface Structure and Interpretation.* MIT Press.

Hozumi Tanaka and Masahiro Ueki. 1995. Japanese Parsing System using EDR Dictionary. `http://www.icot.or.jp/AITEC/PUBLICATIONS/Itaku/95/catalogue18-E.html`

Simon Thompson. 1991. *Type Theory and Functional Programming.* Addison-Wesley.

K. Vijay-Shanker and David J. Weir. 1990. Polynomial Time Parsing of Combinatory Categorial Grammars. In *ACL28*, 1990.

K. Vijay-Shanker and David J. Weir. 1991. Polynomial Parsing of Extentions of Context-Free Grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology.* Kluwer.

K. Vijay-Shanker and David J. Weir. 1993. Parsing Constrained Grammar Formalisms. *Computational Linguistics*, 19(4).

K. Vijay-Shanker and D. J. Weir. 1994. The Equivalence of Four Extensions of Context-Free Grammars. *Mathematical Systems Theory*, 27:511–546.

David Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD thesis, University of Pennsylvania.

Kent Wittenburg and Robert E. Wall. 1991. Parsing with Categorial Grammar in Predictive Normal Form. In Masaru Tomita, editor, *Current Issues in Parsing Technology.* Kluwer.

Kent Barrows Wittenburg. 1986. *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification-Based Formalism.* PhD thesis, University of Texas, at Austin.

Kent Wittenburg. 1987. Predictive Combinators: A Method for Efficient Processing of Combinatory Categorial Grammars. In *ACL25*, 1987.

PROBABILISTIC PARSING USING LEFT CORNER LANGUAGE MODELS

Christopher D. Manning
Linguistics F12
University of Sydney NSW 2006
Australia
cmanning@mail.usyd.edu.au

Bob Carpenter
Lucent Technologies Bell Labs
600 Mountain Avenue, Room 2D-329
Murray Hill NJ 07974
carp@research.bell-labs.com

Abstract

We introduce a novel parser based on a probabilistic version of a left-corner parser. The left-corner strategy is attractive because rule probabilities can be conditioned on both top-down goals and bottom-up derivations. We develop the underlying theory and explain how a grammar can be induced from analyzed data. We show that the left-corner approach provides an advantage over simple top-down probabilistic context-free grammars in parsing the *Wall Street Journal* using a grammar induced from the Penn Treebank. We also conclude that the Penn Treebank provides a fairly weak testbed due to the flatness of its bracketings and to the obvious overgeneration and undergeneration of its induced grammar.

1 Introduction

For context-free grammars (CFGs), there is a well-known standard probabilistic version, Probabilistic Context-Free Grammars (PCFGs), which have been thoroughly investigated [Suppes, 1970, Sankoff, 1971, Baker, 1979, Lari and Young, 1990, Kupiec, 1991, Jelinek et al., 1992, Charniak, 1993]. Under this model, one assigns probabilities for different rewrites of a non-terminal. Or in other words, one is giving the probability of a local subtree given the mother node. So, for example, we might have:

$$\begin{aligned} P(NP \rightarrow Det\ N) &= 0.2 \\ P(NP \rightarrow Pron) &= 0.1 \end{aligned}$$

where in general, $\forall$ nonterminals A, $\sum_\gamma P(A \rightarrow \gamma) = 1$.

But standard PCFGs are only one way to make a probabilistic version of CFGs. If we think in parsing terms, a PCFG corresponds to a probabilistic version of top down parsing, since at each stage we are trying to predict the child nodes given knowledge only of the parent node. Other parsing methods lend themselves to different models of probabilistic conditioning. Usually, such conditioning is a mixture of top-down and bottom-up information. This paper discusses some initial results from another point in this parameter space where the conditioning reflects a left-corner parsing strategy, yielding what we will call probabilistic left-corner grammars (PLCGs).[1] Left-corner parsers simultaneously work top-down from a goal category and bottom-up from the

[1]This name may appear strange since the symbolic part of the grammar is unchanged and still context-free. But if we regard the probabilistic conditioning as part of the grammar, then we do have a different kind of grammar. We should then perhaps call the result a LCPG, but we place the P in initial position for reasons of tradition.

Expansion	% as Subj	% as Obj
NP → PRP	13.7%	2.1%
NP → NNP	3.5%	0.9%
NP → DT NN	5.6%	4.6%
NP → NN	1.4%	2.8%
NP → NP SBAR	0.5%	2.6%
NP → NP PP	5.6%	14.1%

Table 1: Selected common expansions of NP as Subject vs. Object

left corner of a particular rule. For instance, a rule such as $S \rightarrow NP\ VP$ has the left-corner NP and will be fired whenever an NP has been derived and an S would help toward the eventual goal category. In this paper, we present algorithms for PLCG parsing, present some results comparing PLCG parsing with PCFG parsing, and discuss some mechanisms for improving results.

Why might one want to employ PLCGs? While the main perceived weakness of PCFGs is their lack of lexicalization, they are also deficient on purely structural grounds [Briscoe and Carroll, 1993]. Inherent to the idea of a PCFG is that probabilities are context-free: for instance, that the probability of a noun phrase expanding in a certain way is independent of where the NP is in the tree. Even if we in some way lexicalize PCFGs to remove the other deficiency, this assumption of structural context-freeness remains. But this context-free assumption is actually quite wrong. For example, Table 1 shows how the probabilities of expanding an NP node (in the Penn Treebank) differ wildly between subject position and object position. Pronouns, proper names and definite NPs appear more commonly in subject position while NPs containing post-head modifiers and bare nouns occur more commonly in object position (this reflects the fact that the subject normally expresses the sentence-internal topic [Manning, 1996]).

Another advantage of PLCGs is that parse probabilities are straightforwardly calculated from left to right, which is convenient for online processing and integration with other linear probabilistic models.[2]

2 Probabilistic Left Corner Grammars

Left corner parsers [Rosenkrantz and Lewis II, 1970, Demers, 1977] work by a combination of bottom-up and top-down processing. One begins with a goal category (the root of what is currently being constructed), and then looks at the left corner of the string (i.e., one shifts the next terminal). If the left corner is the same category as the goal category, then one can stop. Otherwise, one projects a possible local tree from the left corner. The remaining children of this projected local tree then become goal categories and one recursively does left corner parsing of each. When this local tree is finished, one again recursively does left-corner parsing with this subtree as the left corner, and the same goal category. To make this description more precise, a Prolog version of a simple left corner recognizer is shown in Figure 1. This particular parser assumes a rule format for `rule/2` that allows lexical material to appear on the right-hand side of a rule.[3]

A common formulation of left corner parsers is in terms of a stack of found and sought constituents, the latter being represented as minus categories on the stack (and represented as `m(Cat)` in the Prolog code). A left corner parser that uses a stack is shown in Figure 2. Shifting is now an explicit option on a par with projecting and attaching, but note that when to shift remains deterministic. If the thing on top of the stack is a predicted `m(Cat)`, then one must shift, and one can never successfully shift at other times. This second version of the parser more transparently corresponds to the probabilistic language model we employ.

To produce a language model that reflects the operation of a left corner parser, we have to provide probabilities for the different operations (the clauses of `process` in Figure 2). For each step, we need to decide the probabilities of deciding to shift, attach, or project. The only interesting choice here is deciding whether to attach in cases where the left corner category and the goal category are the same. For the other two operations of the parser, we need to model the probability of shifting different terminals, and the probability of building a

[2]Note however, that while the obvious way of calculating PCFG probabilities does not allow incremental processing, incremental calculation is possible, as discussed by [Jelinek et al., 1992].

[3]In general, empty categories can be accommodated by allowing a category to be introduced for completion without popping a word off the input stack.

```
% lc(List_of_words_to_parse)
lc(Ws) :- start(C),
    complete_list([C], Ws, []).

complete(C, C, Ws, Ws).           % attach
complete(W, C, Ws, NewWs) :-      % project lc
    rule(LHS, [W|Rest]),          % lex / phrase
    complete_list(Rest, Ws, Ws2),
    complete(LHS, C, Ws2, NewWs).

complete_list([], Ws, Ws).
complete_list([C|Cs], [W|Ws], NewWs) :-
    complete(W, C, Ws, Ws2),   % shift
    complete_list(Cs, Ws2, NewWs).
```

Figure 1: A Prolog LC parser

```
slc(Ws) :- start(C),
    slc(Ws, [m(C)]).

slc([], []).
slc(L0, Stack0) :-
    process(Stack0, Stack, L0, L),
    slc(L, Stack).

process([A, m(A)|Stack], Stack, L, L). % attach
process([Item|Items], Stack, L, L) :-  % project LC
    rule(LHS, [Item|Rest]),
    predict(Rest, [LHS|Items], Stack).
process(Stack, [L|Stack], [L|Ls], Ls). % shift

predict([], L, L).
predict([L|Ls], L2, [m(L)|NewLs]) :-
    predict(Ls, L2, NewLs).
```

Figure 2: A Prolog LC stack parser

certain local tree given the left corner (lc) and the goal category (gc). Under this model, we have probabilities for this last operation like this:

$$\begin{aligned} P(SBar \rightarrow P\ S | lc = P, gc = S) &= 0.25 \\ P(PP \rightarrow P\ NP | lc = P, gc = S) &= 0.55 \end{aligned}$$

How to make probabilities out of the above choices is made precise in the next section.

2.1 The LC probability of a parse

In this section, we provide probabilities for left-corner derivations. These form the basis for a language model that assigns probabilities to sentences. For a sentence s, we have that the probability of a sentence according to a grammar G is:

$$\begin{aligned} P(s|G) &= \sum_t P(s,t|G), \quad t \text{ a parse tree of } s \\ &= \sum_{\{t:\ \text{yield}(t)=s\}} P(t|G) \end{aligned}$$

The last line follows since the parse tree determines the terminal yield. It is therefore sufficient to be able to calculate the probability of a (parse) tree. Below we suppress the conditioning of the probability according to

the grammar.

Now following the intuition of our model having been inspired by left corner parsing, we can express the probability of a parse tree in terms of the probabilities of left corner derivations of that parse tree:

$$P(t) = \sum_{d \text{ a LC derivation of } t} P(d)$$

But under left corner parsing, each parse tree has a unique derivation and so the summation sign can be dropped from this equation.

Now, without any assumptions, the probability of a derivation can be expressed as a product in terms of the probabilities of each of the individual operations in the derivation. Suppose that $\{C_1, \ldots, C_m\}$ is the sequence of operations in the LC parse derivation d of t. Then, by the chain rule, we have:

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

In practice, we cannot condition the probability of each parse decision on the entire history. The simplest model, which we will explore for the rest of this section, is to assume that the probability of each parse decision is largely independent of the parse history, and just depends on the state of the parser. In particular, we will assume that it depends simply on the left corner and top goal categories of the parse stack. This drastic assumption nevertheless gives us a slightly richer probabilistic model than a PCFG, because elementary left-corner parsing actions are conditioned by the goal category, rather than simply being the probability of a local tree. For instance, the probability of a certain expansion of NP may be different in subject position and object position, because the goal category is different.

Each elementary operation of a left corner parser is either a shift, an attach or a left corner projection. Under the independence assumptions mentioned above, the probability of a shift will simply be the probability of a certain left corner daughter (lc) being shifted given the current goal category (gc), which we will model by P_{shift}. Note that when to shift is deterministic. If a goal (i.e., minus) category is on top of the stack (and hence there is no left corner category), then one must shift. Otherwise one cannot. If one is not shifting, one must choose to attach or project, which we model by P_{att}. Attaching only has a non-zero probability if the left corner and the goal category are the same, but we define it for all pairs. If we do not attach, we project a constituent based on the left corner with probability P_{lc}. Thus the probability of each elementary operation C_i can be expressed in terms of probability distributions P_{shift}, P_{att}, and P_{lc} as follows:

$$P(C_i = \text{shift } lc) = \begin{cases} P_{shift}(lc|gc) & \text{if top of the stack is } gc \\ 0 & \text{otherwise} \end{cases}$$

$$P(C_i = \text{attach}) = \begin{cases} P_{att}(lc, gc) & \text{if top of the stack is not } gc \\ 0 & \text{otherwise} \end{cases}$$

$$P(C_i = \text{project } A \to \gamma) = \begin{cases} (1 - P_{att}(lc, gc)) P_{lc}(A \to \gamma | lc, gc) & \text{if top of the stack is not } gc \\ 0 & \text{otherwise} \end{cases}$$

Where these operations obey the following:

$$\sum_{lc} P_{shift}(lc|gc) = 1$$

$$\text{If } lc \neq gc,\ P_{att}(lc, gc) = 0$$

$$\sum_{\{A \to \gamma : \gamma = lc, \ldots\}} P(A \to \gamma | lc, gc) = 1$$

From the above we note that the probabilities of the choice of projections sums to one, and hence, since other probabilities are complements of each other, the probabilities of the actions available for each elementary operation sum to one. There are also no dead ends in a derivation, because unless A is a possible left corner constituent of gc, $P(A \to \gamma | lc, gc) = 0$. Thus we have shown that these probabilities define a language model.[4] That is, $\sum_s P(s|G) = 1$. It is possible to extend the PLCG model in various ways to include more probabilistic conditioning, as we discuss briefly later, but our current results reflect this model.

[4]Subject to showing that the probability mass accumulates in finite trees.

Rule			Freq.	PCFG Prob.	Rule			Freq.	PCFG Prob.
PP	→	IN NP	76617	0.81	NP	→	PRP	17323	0.06
NP	→	NP PP	34965	0.11	ADVP	→	RB	14228	0.72
NP	→	DT NN	29351	0.09	NP	→	NN	13586	0.04
S	→	NP VP	28292	0.30	NP	→	NNS	13318	0.04
S	→	VP	23559	0.25	VP	→	TO VP	12900	0.09
S	→	NP VP .	17703	0.19	NP	→	NNP	12575	0.04

Table 2: Highest frequency CFG rules in Penn Treebank

2–12 word sentences	this paper	Charniak
Grammar (rules)	14 971	10 605
% sent. length < cutoff	16.6%	
Test set size (sentences)	401	
Average Length (words)	8.3	8.7
Precision	89.8%	88.6
Recall	90.7%	91.7
Labelled Precision	83.5%	
Labelled Recall	82.9%	
Labelled Precision +1	87.1%	
Labelled Recall +1	85.2%	
Average CBs	0.27	
Non-crossing accuracy	95.8%	97.9%
Sentences with 0 CBs	84.5%	

Table 3: PCFG results

3 Parsing experiments

3.1 PCFG Experiment

Training and testing were done on Release 2 of the Penn Treebank [Marcus et al., 1993], published in 1995. As in other recent work [Magerman, 1995, Collins, 1996], training was done on sections 02–21 of the Wall Street Journal portion of the treebank (approximately 40,000 sentences, 780,153 local trees) and final testing was done on section 23, which contains 2416 sentences. Counts of how often each local tree occurred in the treebank were made, and these were used directly to give probabilities for rewriting each nonterminal. The highest frequency rules are given in Table 2. Local trees were considered down to the level of preterminals (i.e., part of speech tags); lexical information was ignored.[5] Every tree was given a new root symbol 'ROOT', attached by a unary branch to the root in the treebank. Empty nodes (of which there are several kinds in the treebank) were ignored, and nonterminals above them that dominated no pronounced words were also deleted.[6] No attempt was made to do any smoothing. While in a few cases this would clearly be useful (e.g., the training data allows a compound noun to be modified by four adjectives, but not a simple noun), in practice the induced treebank grammar is hugely ambiguous, and greatly overgenerates. Thus, while the lack of smoothing in principle disallows some correct parses from being generated, the treebank grammar can always produce some parse for a sentence [Charniak, 1996] and adding unseen rules with low probabilities is unlikely to improve bottom line performance, because these added rules are unlikely to appear in the maximum probability parse. A better solution would be to use a covering grammar with fewer rules and a more deeply nested structure.

Testing was done by chart-parsing the part of speech tags of the sentences (i.e., ambiguities in part of speech assignment were assumed to be successfully resolved). An exhaustive chartparse was done and the highest probability (Viterbi) parse was selected, in the standard way [Charniak, 1993]. Results from such parsing are shown in Table 3 together with results from [Charniak, 1996]. The measures shown have been used

[5] Of course, we could easily integrate our model with a tagging model.

[6] Simply eliminating empties in the treebank is dangerous because they are the only trace of unbounded dependency constructions. This leads to ridiculous rules like $S \rightarrow VP$ (with 23559 appearances in the treebank) stemming from $S \rightarrow NP\ VP$ where there is a trace subject NP. A purely context-free solution would be to introduce slash percolation.

2/4–40 word sentences	PCFG	Magerman	Collins
% sent. length < cutoff	92.9%		
Test set size (sentences)		1759	2416
Average Length (words)	21.9	22.3	
Precision	78.8%	86.3%	
Recall	80.4%	85.8%	
Labelled Precision		84.5%	86.3%
Labelled Recall		84.0%	85.8%
Average CBs		1.33	1.14
Non-crossing accuracy	87.7%		
Sentences with 0 CBs		55.4%	57.2%

Table 4: PCFG [Charniak, 1996] vs. [Magerman, 1995]/[Collins, 1996] comparison

in various earlier works, and generally draw from the PARSEVAL measures [Black et al., 1991]. Precision is how many brackets in the parse match those in the correct tree (perhaps also examining labels), recall measures how many of the brackets in the correct tree are in the parse. The unmarked measures ignore unary constituents, the ones marked +1 include unary constituents.[7] Crossing brackets (CBs) measures record how many brackets in the parse cross bracketings in the correct tree, with the non-crossing accuracy measuring the percent of brackets that are not CBs. The '% sent. length < cutoff' says what percentage of sentences within the 2416 sentence test section were shorter than the cutoff and thus used in the test set for the current experiment. Our results are not directly comparable to Charniak's since he was using an earlier release of the Penn Treebank. Further, he used two strategies that aimed at increasing performance: recoding auxiliaries from their Penn tag (which is undistinguished from other verbs) to special auxiliary tags, and adding a (crude) correction factor to the PCFG model so that it uniformly favored right-branching trees rather than being context free. Whether the former change was shown to be beneficial is not discussed, but the later correction factor improved results by about two percent. The fact that the results are mixed between the two systems suggests that the quality of the Penn Treebank has improved in the second release, and these gains roughly match the gains from these factors. It is useful that the results are roughly comparable since we can then use Charniak's results as a rough benchmark for PCFG performance on longer sentences, which we have not obtained.[8]

Charniak's central contention is that purely structural parsing like this using treebank grammars works much better than community lore would have you believe. Indeed, as the comparison in Table 4 suggests, it does not work much worse than [Collins, 1996], a leading recent parser that includes lexical content. That is, it seems one can score well in the PARSEVAL measures using purely structural factors, and that the use of lexical factors in other models is at present only adding a little to their performance.[9] This is in part because the Penn treebank does not represent certain semantic "attachment" decisions, and the structure of the trees minimizes the penalty for other "attachment" errors, as we discuss in the last section of this paper.

3.2 LC parsing results

The probabilistic left corner parser is implemented as a beam parser in C. As a k-best beam parser, it is not guaranteed to find the best parse, unless the beam is effectively infinite. Space requirements depend on the size of the beam, and the length of the sentence, but are considerably more reasonable than those for the chart parser used above. Nevertheless, the branching factor in the search space is very high because there are many

[7] The original PARSEVAL measures (because they were designed for comparing different people's parsers that used different theories of grammar) ignored node labels entirely, discarded unary brackets, and performed other forms of tree normalization to give special treatment to certain cases such as verbal auxiliaries. While such peculiarities made some sense in terms of the purpose for which the measures were originally developed, it is not clear they are appropriate for the use of these measures within the statistical NLP community. Thus people often report labelled measures, but it is often unclear whether the other rules and transformations of the standard have been employed (but this affects the results reported). In this paper, unary nodes are deleted in measures except those marked +1, but none of the special case tree transformations in the PARSEVAL standard are applied. All punctuation and all constituent labels (but not functional tags) are also retained.

[8] Our PCFG parser builds a complete chart, which leads to unviable space requirements for long sentences.

[9] Again, results are not strictly comparable. The comparison is unfair to Magerman and Collins' systems since they are also doing part of speech tagging, whereas the PCFG is not. But on the other hand, Magerman and Collins' parsers conflate the ADVP and PRT labels and ignore all punctuation, which improves their reported results.

	Parser results			2–12 Error
Sentence Lengths:	2–12	2–16	2–25	Reduction
Beam size	50 000	50 000	40 000	
% sent. length < cutoff	16.6%	28.1%	60.2%	
Test set (sentences)	401	680	1454	
Average length (words)	8.3	10.9	16.3	
Precision	92.0%	90.1%	84.6%	21.6%
Recall	92.3%	89.5%	83.2%	17.2%
Labelled Precision	87.1%	86.0%	81.1%	21.8%
Labelled Recall	86.7%	84.9%	79.6%	22.2%
Labelled Precision +1	88.6%	87.7%	83.5%	11.6%
Labelled Recall +1	88.3%	86.3%	81.5%	20.9%
Average CBs	0.21	0.43	1.25	22.2%
Non-crossing accuracy	96.8%	94.7%	89.6%	23.8%
Sentences with 0 CBs	87.5%	76.0%	52.0%	19.4%

Table 5: Left Corner Parser results from n-ary grammar.

rules possible with a certain left corner and goal category (especially for a grammar induced from the Penn Treebank in the manner we have described). Therefore, a huge beam is needed for good results to be obtained. A way of addressing this problem by binarizing the grammar is discussed in the next section.

The parser maintains two beams, one containing partial LC parses of the first i words, and another in which is built a beam of partial LC parses of the first $i+1$ words. The partial parses are maintained as pointers to positions in trie data structures that represent the list of parser moves to this point and the current parse stack. At the end of parsing, the lists of parser moves can be easily turned into parse trees for the n-best parses in the beam.

Results are shown in Table 5. They reflect the same training and testing data as described above. The results show a small increase in performance for PLCGs. This is shown more dramatically in the right-hand column of Table 5, which shows that the extra information provided by the Left Corner goal category reduces parsing errors by about 20% over our PCFG results on 2–12 word sentences.

4 Binarization

For the parser above, use of a beam search is quite inefficient because, for a given left corner and goal category, there are often hundreds of possible local trees that could be projected, and little information is available at the time this decision is made, since the decision mainly depends on words that have not yet been shifted. Therefore the beam must be large for good results to be obtained, and, at any rate, the branching factor of the search space is extremely high, which slows parsing. One could imagine using various heuristics to improve the search, but the way we have investigated combatting this problem is by binarizing the grammar.

The necessary step for binarization is to eliminate productions with three or more daughters. We carried this out by merging the tails, so that a rule such as $NP \rightarrow Det\ JJ\ NN$ is replaced by two rules, $NP \rightarrow Det\ NP_{Det}$ and $NP_{Det} \rightarrow JJ\ NN$. This is carried out recursively until only binary rules remain. As a result of this choice of binarization, n-ary rules that share the same mother and left corner category all reduce to a single rule. This greatly cuts the branching factor of the search space and allows decisions to be put off during parsing, until more of the input has been seen, at which point alternative continuations can be better evaluated. Furthermore, the weights for such rules all combine into a larger weight for the combined rule.[10]

It is important to note that the resulting model is *not* equivalent to our original model. While the straightforward way of binarizing a PCFG yields the same probability estimates for trees as the n-ary grammar, this is not true for our PLCG model since we are now introducing new estimates for shifting terminals for each of our newly created non-terminals. Slightly different probability estimates result, and further work is needed to investigate what relationship exists between them and the probability estimates of the original grammar.

[10] If we were to do this even for rules that start out binary, and eliminate unary rules downward rather than upward, then for every left corner C and mother G there will be a unique rule $G \rightarrow C\ G_C$.

	Parser Results				2–25 Error
Sentence Lengths:	2–12	2–16	2–25	2-40	Reduction
Beam size	40 000	40 000	40 000	40 000	
% sent. length < cutoff	16.3%	28.1%	60.2%	91.8%	
Test set (sentences)		680	1454	2216	
Average length (words)	8.3	10.9	16.3	21.6	
Precision	93.5%	91.4%	86.6%	83.0%	13.0%
Recall	92.8%	89.9%	84.3%	80.7%	6.5%
Labelled Precision	89.8%	88.1%	83.4%	79.9%	12.2%
Labelled Recall	88.4%	86.3%	81.0%	77.6%	6.9%
Labelled Precision +1	90.0%	89.0%	85.1%	81.9%	9.7%
Labelled Recall +1	89.5%	87.2%	82.6%	79.5%	5.9%
Average CBs	0.17	0.39	1.09	1.99	12.8%
Non-crossing accuracy	97.2%	95.2%	90.9%	87.6%	12.5%
Sentences with 0 CBs	89.8%	78.2%	55.8%	41.5%	7.9%

Table 6: Left Corner Parser results with binarized grammar

Prior to the above binarization step, one might also wish to eliminate unary productions, much as we earlier eliminated empty categories. This can be done in two ways. One way is to fold them upwards. This preserves lexical tagging. That is, if there is a category A dominating only a tree rooted at B, then the category A is eliminated and the tree rooted at B moved upwards. This may cause the number of rules to increase, because a local tree that started with a daughter A will now show up with daughter B in the same place. The alternative is to eliminate B and replace it with A. This can also create a new local tree instance because the daughters of B now show up with a new mother A. In this way, lexical tags can be changed. For instance, consider a rule $NP \rightarrow NNP$ for a noun phrase rewriting as a proper noun. In the context of eliminating empty categories upward, we get a new rule $S \rightarrow NNP\ VP$, whereas by eliminating empty categories downward we would have produced a new lexical entry $NP \rightarrow Jones$. Our current results do not reflect the elimination of unary rules, but we hypothesize that doing this would further improve the measures that do not consider unary nodes, while probably harming the results on measures that do include unary nodes.

Table 6 shows that binarization brings a further modest improvement in results. The righthand column shows the percent error reduction on 2–25 word sentences between the n-ary grammar and the binary grammar.

5 Extended Left Corner Models

More sophisticated PLCG parsing models will naturally provide greater conditioning of the probability of an elementary operation based on the parse history. There are a number of ways that one could then proceed. From the background of work on LC parsing, a natural factor to consider is the size of the parse stack, and we will briefly investigate incorporating this factor.

For left corner parsing, the stack size is particularly interesting. [Stabler, 1994] notes that in contrast to bottom-up and top-down parsing methods, left-corner parsers can handle arbitrary left-branching and right-branching structures with a finitely bounded stack size. Furthermore, left-corner parses of center embedded constructions have stack lengths proportional to the amount of embedding. This is not actually true for the stack-based LC parser presented earlier which has the stack growing without bound for rightward-branching trees. To gain the desirable property that Stabler notes, one needs to do stack composition by deciding immediately whether to attach whenever one projects a category that matches the current goal category, rather than delaying the attachment until after the left corner constituent is complete. If one decides to attach, the goal minus category and the mother category are immediately removed from the stack. This is implemented by replacing the predicate `process` in Figure 2 by the code in Figure 3, where composition is done by the predicate `compose`. All else being equal, this change makes no difference to the probabilistic model presented earlier. In practice though, this formulation makes a beam search less effective since we are bringing forward the decision of whether to attach or not, and often both alternatives must be tried which fills out the beam unnecessarily.

Given human intolerance for center embeddings and ease of parsing left and right branching structures, we would expect the stack sizes to stay low. The general prediction (and empirical fact) is that the probability

```
process([Item|Items], Stack, L, L) :- % lc
    rule(LHS, [Item|Rest]),
    compose(LHS, Items, Stack1),
    predict(Rest, Stack1, Stack).
process(Stack, [L|Stack], [L|Ls], Ls). % shift

compose(A, [m(A)|L], L). % attach
compose(A, L, [A|L]).    % don't
```

Figure 3: Altered code for a Prolog LC parser that does stack composition

Stack Size	Total	Stack size change −1	0	+1
8	23	20 (87%)	3 (13%)	0
7	160	86 (54%)	54 (34%)	20 (13%)
6	1291	987 (76%)	218 (17%)	86 (7%)
5	3745	1393 (37%)	1365 (36%)	987 (26%)
4	17000	12116 (71%)	3491 (21%)	1393 (8%)
3	39105	12241 (31%)	14748 (38%)	12116 (31%)
2	108544	63160 (58%)	33143 (31%)	12241 (11%)
1	71681	8521 (12%)	0	63160 (88%)

Table 7: Changes in stack size for Penn Treebank

of not attaching (composing) decreases slightly with the size of the stack. The counts for the Penn Treebank, after binarization (including removal of unary rules), are given in Table 7. Once one looks beyond the odd-even effect in the data, the decreasing probability of not attaching can be clearly seen.

To incorporate stack length into the model, we wish to more accurately predict: $P(C_i|C_1,\ldots,C_{i-1})$. Previously, histories of parse steps were equivalenced according to just the goal category and the left corner (if present). Now, we are going to additionally differentiate parse histories depending on ℓ, the length of the stack after C_{i-1}. As before, if the top of the stack is a predicted category, we will shift; otherwise we cannot shift. In the latter case, we will predict $P_\delta(\delta|\ell, gc, lc)$, the probability of various changes in the stack size, based on the stack size, the goal category, and the left corner.

Let us assume that rules are binary, as discussed above. Then the possible change in the stack length from a single elementary operation is between -2 and $+1$. Given that we are not shifting, we have the following:

Stack Delta	Rule Type
-2	unary left corner and attach
-1	binary left corner and attach
0	unary left corner (no attach)
$+1$	binary left corner (no attach)

The probability of each elementary operation will then be the probability of a certain stack delta (given the stack size, left corner and goal category) times the probability of a certain rule, given the left corner, the goal category, and the stack delta. Whether we attach (compose) or not is deterministic given the stack delta, and so we no longer need to model the P_{att} distribution. Under this model, the probability of different projection operations (given that we are not in a position to shift) becomes:

$$\begin{aligned}
P(C_i = \text{project } A \to lc, \text{ attach}) &= P_\delta(-2|\ell, lc, gc) P_{lc'}(A \to lc | lc, gc, \ell, \delta) \\
P(C_i = \text{project } A \to lc, \text{ do not attach}) &= P_\delta(0|\ell, lc, gc) P_{lc'}(A \to lc | lc, gc, \ell, \delta) \\
P(C_i = \text{project } A \to lc\ c_2, \text{ attach}) &= P_\delta(-1|\ell, lc, gc) P_{lc'}(A \to lc\ c_2 | lc, gc, \ell, \delta) \\
P(C_i = \text{project } A \to lc\ c_2, \text{ do not attach}) &= P_\delta(+1|\ell, lc, gc) P_{lc'}(A \to lc\ c_2 | lc, gc, \ell, \delta)
\end{aligned}$$

A variety of other extended models are possible. Note that the model does not need to be uniform, and that we can estimate different classes of elementary operations using different probabilistic submodels. In particular,

at the level of preterminals, we can incorporate a tagging model. Given the structure of the Penn Treebank, a terminal w_j is always dominated by a unary rule giving the terminal's part of speech p_j. In line with our basic model above, the choice of part of speech for a word (where the word now counts as the left corner) will certainly depend on the current goal category. However, we can also condition it on other preceding parse decisions, in particular on the part of speech of the preceding two words, or, perhaps, in certain circumstances, on the particular word that preceded. Taking the former possibility, we can say, for cases where C_i involves predicting a preterminal (through a left corner projection step),

$$\begin{aligned} P(C_i|C_1,\ldots,C_{i-1}) &= P(p_j|w_j,gc,C_1,\ldots,C_{i-1}) \\ &\approx P(p_j|w_j,gc,p_{j-2},p_{j-1}) \end{aligned}$$

Assuming – perhaps rashly – independence between the conditioning variables that we have been using before (lc, gc) and the new ones (p_{j-1}, p_{j-2}), then we have that:

$$P(p_j|w_j,gc,p_{j-2},p_{j-1}) \approx \frac{P(p_j|w_j,gc)P(p_j|p_{j-2},p_{j-1})}{P(p_j)}$$

This is nice in part because we can calculate it using just the statistics previously gathered and a simple trigram model over POS tags. (An incidental nice property is that the probability is for the POS given the word, and not the other way round, as in the 'confusing' $P(w|t)$ term of standard Markov model POS taggers.)[11]

6 Comparison with previous work

Previous work on non-lexicalized parsing of Penn Treebank data includes [Charniak, 1996] and [Schabes et al., 1993], but the work perhaps most relevant to our own is that of [Briscoe and Carroll, 1993] and [Carroll and Briscoe, 1996], which also seeks to address the context-freeness assumption of PCFGs. They approach the problem by using a probabilistic model based on LR parse tables. Unfortunately, many differences of approach make meaningful comparisons difficult, and a comparable study of using PLCGs versus Probabilistic LR parsing remains to be done. Briscoe and Carroll use their Probabilistic LR grammar to guide the actions of a unification-based parser which uses a hand-built grammar. While we are sympathetic with their desire to use a more knowledge-based approach, this means that: their language model is deficient, since probability mass is given to derivations which are ruled out because of unification failures; the coverage of their parser is quite limited because of limitations of the grammar used; and much time needs to be expended in developing the grammar, whereas our grammar is acquired automatically (and quickly) from the treebank. Moreover, while results are not directly comparable, our parsers seems to do rather better on precision and recall than the parser described in [Carroll and Briscoe, 1996], while performing somewhat worse on crossing brackets measures. However, Carroll and Briscoe's inferior results probably reflect the fact that the parse trees of their grammar do not match those of their treebank more than anything else.

7 Observations on why parsing the Penn Treebank is easy

How is it that the purely structural – and context free even in structural terms – PCFG parser manages to perform so well? An important observation is that the measures of precision and recall (labelled or not) and crossing brackets are actually quite easy measures to do well on. It is important to notice that they are measuring success at the level of individual decisions – and normally what makes NLP hard is that you have to make many consecutive decisions correctly to succeed. The overall success rate is then the n^{th} power of the individual decision success rate – a number that easily becomes small.

But beyond this, there are a number of features particular to the structure of the Penn Treebank that make these measures particularly easy. Success on crossing brackets is helped by the fact that Penn Treebank trees

[11] Below is a derivation of this equation, written in a simplified form with three variables. We assume b and c are independent, and the result then follows by using Bayes' rule followed by the definition of conditional probability:

$$P(a|b,c) = \frac{P(a)P(b,c|a)}{P(b,c)} = \frac{P(a)P(b|a)P(c|a,b)}{P(b)P(c|b)} = \frac{P(a)P(b|a)P(c|a)}{P(b)P(c)} = \frac{P(a)P(b,a)P(c,a)}{P(a)^2P(b)P(c)} = \frac{P(a|b)P(a|c)}{P(a)}$$

Penn VP attach	(VP saw (NP the man) (PP with (NP a telescope)))
Penn NP attach	(VP saw (NP (NP the man) (PP with (NP a telescope))))
Another VP attach	(VP saw (NP the (N′ man)) (PP with (NP a (N′ telescope)))))
Another NP attach	(VP saw (NP the (N′ man (PP with (NP a (N′ telescope))))))

Table 8: Penn trees versus other trees

are quite flat. To the extent that sentences have very few brackets in them, the number of crossing brackets is likely to be small. Identifying troublesome brackets that would lower precision and recall measures is also avoided. As a concrete instance of this, one difficulty in parsing is deciding the structure of noun compounds [Lauer, 1995]. Noun compounds of three or more words in length can display any combination of left- or right-branching structure, as in [[cable modem] manufacturer] vs. [computer [power supply]]. But such fine points are finessed by the Penn Treebank, which gives a completely flat structure to a noun compound (and any other pre-head modifiers) as shown below (note that the first example also illustrates the rather questionable Penn Treebank practice of tagging hyphenated non-final portions of noun compounds as adjectives!).

(NP a/DT stock-index/JJ arbitrage/NN sell/NN program/NN)
(NP a/DT joint/JJ venture/NN advertising/NN agency/NN)

Another case where peculiarities of the Penn Treebank help is the (somewhat nonstandard) adjunction structures given to post noun-head modifiers, of the general form (NP (NP the man) (PP in (NP the moon))). A well-known parsing ambiguity is whether PPs attach to a preceding NP or VP – or even to a higher preceding node – and this is one for which lexical or contextual information is clearly much more important than structural factors [Hindle and Rooth, 1993]. Note now that the use of the above adjunction structure reduces the penalty for making this decision wrongly. Compare Penn Treebank style structures, and another common structure in the examples in Table 8. Note the difference in the results:

	Error	Errors assessed		
		Prec.	Rec.	CBs
Penn	VP instead of NP	0	1	0
	NP instead of VP	1	0	0
Another	VP instead of NP	1	2	1
	NP instead of VP	2	1	1

The forgivingness of the Penn Treebank scheme is manifest. One can get the attachment wrong and not have any crossing brackets.[12]

8 Conclusions

This paper explores a new class of probabilistic parsing algorithms for context-free grammars, probabilistic left corner grammars. The ability of left corner parsers to support left-to-right online parsing makes them initially promising for many tasks. The different conditioning model is slightly richer than that of standard PCFGs, and this was shown to bring worthwhile performance improvements over a standard PCFG when used to parse Penn Treebank sentences. Beyond this, the model can be extended in various ways, an avenue that we have only just begun exploring. Because the left-corner component of the grammar is purely structural, it can be combined with other models that include lexical attachment preferences and preferences for basic phrasal chunking (both incorporated into Collins' parser).

References

[Baker, 1979] Baker, J. K. (1979). Trainable grammars for speech recognition. In Klatt, D. H. and Wolf, J. J., editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550.

[12] If one includes unary brackets (recall footnote 7), then the contrast becomes even more marked, since there would be 2 precision and recall errors each under the alternative parse trees.

[Black et al., 1991] Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings, Speech and Natural Language Workshop, Pacific Grove, CA*, pages 306–311. DARPA.

[Briscoe and Carroll, 1993] Briscoe, T. and Carroll, J. (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based methods. *Computational Linguistics*, 19:25–59.

[Carroll and Briscoe, 1996] Carroll, J. and Briscoe, T. (1996). Apportioning development effort in a probabilistic LR parsing system through evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-96)*, pages 92–100, University of Pennsylvania.

[Charniak, 1993] Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, MA.

[Charniak, 1996] Charniak, E. (1996). Tree-bank grammars. Technical Report Technical Report CS-96-02, Dept of Computer Science, Brown University.

[Collins, 1996] Collins, M. J. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191.

[Demers, 1977] Demers, A. (1977). Generalized left corner parsing. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 170–181.

[Hindle and Rooth, 1993] Hindle, D. and Rooth, M. (1993). Structural ambiguity and lexical relations. *Computational Linguistics*, 19:103–120.

[Jelinek et al., 1992] Jelinek, F., Lafferty, J. D., and Mercer, R. L. (1992). Basic methods of probabilistic context free grammars. In Laface, P. and De Mori, R., editors, *Speech Recognition and Understanding: Recent Advances, Trends, and Applications*, volume 75 of *Series F: Computer and Systems Sciences*. Springer Verlag.

[Kupiec, 1991] Kupiec, J. (1991). A trellis-based algorithm for estimating the parameters of a hidden stochastic context-free grammar. In *Proceedings of the Speech and Natural Language Workshop*, pages 241–246. DARPA.

[Lari and Young, 1990] Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

[Lauer, 1995] Lauer, M. (1995). *Designing Statistical Language Learners: Experiments on Noun Compounds*. PhD thesis, Macquarie University, Sydney, Australia.

[Magerman, 1995] Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33st Annual Meeting of the Association for Computational Linguistics*, pages 276–283.

[Manning, 1996] Manning, C. D. (1996). *Ergativity: Argument Structure and Grammatical Relations*. CSLI.

[Marcus et al., 1993] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

[Rosenkrantz and Lewis II, 1970] Rosenkrantz, S. J. and Lewis II, P. M. (1970). Deterministic left corner parser. In *IEEE Conference Record of the 11th Annual Syposium on Switching and Automata*, pages 139–152.

[Sankoff, 1971] Sankoff, D. (1971). Branching processes with terminal types: applications to context-free grammars. *Journal of Applied Probability*, 8:233–240.

[Schabes et al., 1993] Schabes, Y., Roth, M., and Osborne, R. (1993). Parsing the Wall Street Journal with the Inside-Outside algorithm. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 341–347, University of Utrecht.

[Stabler, 1994] Stabler, E. P. (1994). The finite connectivity of linguistic structure. In Clifton, C., Frazier, L., and Rayner, K., editors, *Perspectives on Sentence Processing*, pages 303–336. Lawrence Erlbaum, Hillsdale, NJ.

[Suppes, 1970] Suppes, P. (1970). Probabilistic grammars for natural languages. *Synthese*, 22:95–116.

REGULAR APPROXIMATIONS OF CFLS: A GRAMMATICAL VIEW

Mark-Jan Nederhof
Dept. of Humanities Computing, University of Groningen
P.O. Box 716, NL-9700 AS Groningen, The Netherlands
Email: markjan@let.rug.nl

Abstract

We show that for each context-free grammar a new grammar can be constructed that generates a regular language. This construction differs from existing methods of approximation in that use of a pushdown automaton is avoided. This allows better insight into how the generated language is affected. The new method is also more attractive from a computational viewpoint.

1 Introduction

In [Pereira and Wright, 1991] a method was presented that allows the construction of a finite automaton from a context-free grammar. The (regular) language accepted by the finite automaton includes the strings generated by the original context-free grammar, plus a set of additional strings. In this sense, the context-free grammar is *approximated* by the finite automaton.

As intermediate result, an LR automaton is constructed from the context-free grammar. LR automata, which are special cases of pushdown automata, are representations of context-free languages. If one eliminates the pushdown stack of an LR automaton save the top-most element, a finite automaton results. For obtaining a better approximation, this stage can be preceded by encoding into stack symbols some bounded amount of information concerning the stack lying underneath; this information is given by a shorter stack resulting from omitting parts between multiple occurrences of LR states. This process is called *unfolding*.

In the context of error handling for programming languages, the approach from [Pereira and Wright, 1991], without the concept of unfolding, has been conceived independently by [Heckert, 1994].

Some aspects of this method have however not been clarified:

- Why are LR automata used, as opposed to any other kind of pushdown automaton? (See for example [Nederhof, 1994b] for a whole range of different kinds of pushdown automaton.)
- What happens to the language in the approximating process?

The second issue is partly a consequence of the first issue: the structure of a grammar and the structure of the corresponding LR automaton are two very different things, and how modifications to the automaton affect the language in terms of the grammar may be difficult to see.

In this paper, we will somewhat clarify these two issues. Concerning the first issue, we simply state that any kind of pushdown automaton constructed from a grammar allows a finite automaton to be derived that represents a regular approximation of the original language; in other words, LR automata do not have any special properties in this respect. However, in this communication, we will avoid the use of pushdown automata altogether, and perform the process of approximation on the level of the context-free grammar. This also helps to clarify the second issue above, viz. how to monitor the approximating process.

Our method is the following. We define a condition on context-free grammars that is a sufficient condition for a grammar to generate a regular language. We then give a transformation that turns an arbitrary grammar into another grammar that satisfies this condition. This transformation is obviously not language-preserving; it adds strings to the language generated by the original grammar, in such a way that the language becomes regular.

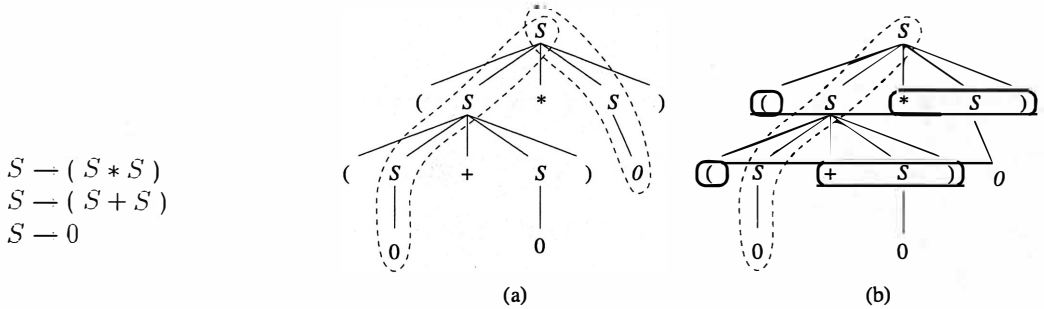


Figure 1: (a) two spines in a parse tree, (b) the grammar symbols to the left and right of a spine immediately dominated by nodes on the spine.

The structure of the paper is as follows. In Section 2 we recall some standard definitions from language theory. Section 3 investigates a sufficient condition for a context-free grammar to generate a regular language. An algorithm to transform a grammar such that this condition is satisfied is given in Section 4. As Section 5 shows, some aspects of our method are undecidable. A refinement of the approach for obtaining more precise approximations is presented in Section 6. Section 7 compares the new method to existing methods.

2 Preliminaries

A *context-free grammar* G is a 4-tuple (Σ, N, P, S), where Σ and N are two finite disjoint sets of terminal and nonterminal symbols, respectively, $S \in N$ is the start symbol, and P is a finite set of rules. Each rule has the form $A \rightarrow \alpha$ with $A \in N$ and $\alpha \in V^*$, where V denotes $N \cup \Sigma$. The relation $\rightarrow$ on $N \times V^*$ is extended to a relation on $V^* \times V^*$ as usual. The transitive and reflexive closure of $\rightarrow$ is denoted by $\rightarrow^*$.

The language *generated* by a context-free grammar is given by the set $\{w \in \Sigma^* \mid S \rightarrow^* w\}$. By definition, such a set is a *context-free* language. By *reduction* of a grammar we mean the elimination from P of all rules $A \rightarrow \gamma$ such that $S \rightarrow^* \alpha A \beta \rightarrow \alpha\gamma\beta \rightarrow^* w$ does not hold for any $\alpha, \beta \in V^*$ and $w \in \Sigma^*$.

We generally use symbols $A, B, C, \ldots$ to range over N, symbols $a, b, c, \ldots$ to range over Σ, symbols X, Y, Z to range over V, symbols $\alpha, \beta, \gamma, \ldots$ to range over V^*, and symbols $v, w, x, \ldots$ to range over Σ^*. We write ϵ to denote the empty string.

A rule of the form $A \rightarrow B$ is called a *unit* rule. A grammar is called *cyclic* if $A \rightarrow^* A$, for some A.

A (nondeterministic) *finite automaton* $\mathcal{F}$ is a 5-tuple $(K, \Sigma, \Delta, s, F)$, where K is a finite set of *states*, of which s is the *initial state* and those in $F \subseteq K$ are the *final states*, Σ is the input alphabet, and Δ is a finite subset of $K \times \Sigma^* \times K$.

We define a *configuration* to be an element of $K \times \Sigma^*$. We define the binary relation $\vdash$ between configurations as: $(q, vw) \vdash (q', w)$ if and only if $(q, v, q') \in \Delta$. The transitive and reflexive closure of $\vdash$ is denoted by $\vdash^*$.

Some input v is *recognized* if $(s, v) \vdash^* (q, \epsilon)$, for some $q \in F$. The language *accepted* by $\mathcal{F}$ is defined to be the set of all strings v that are recognized. By definition, a language accepted by a finite automaton is called a *regular* language.

3 The Structure of Parse Trees

We define a *spine* in a parse tree to be a path that runs from the root down to some leaf. Figure 1 (a) indicates two spines in a parse tree for the string $((0 + 0) * 0)$, according to a simple grammar.

Our main interest in spines lies in the sequences of grammar symbols at nodes bordering on spines. Figure 1 (b) gives an example: to the left of the spine we find the sequence "((" and to the right we find "$+S) * S)$". The way such pairs of sequences may relate to each other determines the strength of context-free grammars, and as we will see later, by restricting this relationship we may reduce the generative power of context-free grammars to the regular languages.

A simpler example is the set of parse trees such as the one in Figure 2 (a), for a 3-line grammar of palindromes. It is intuitively clear that the language is not regular: the grammar symbols to the left of the spine from the

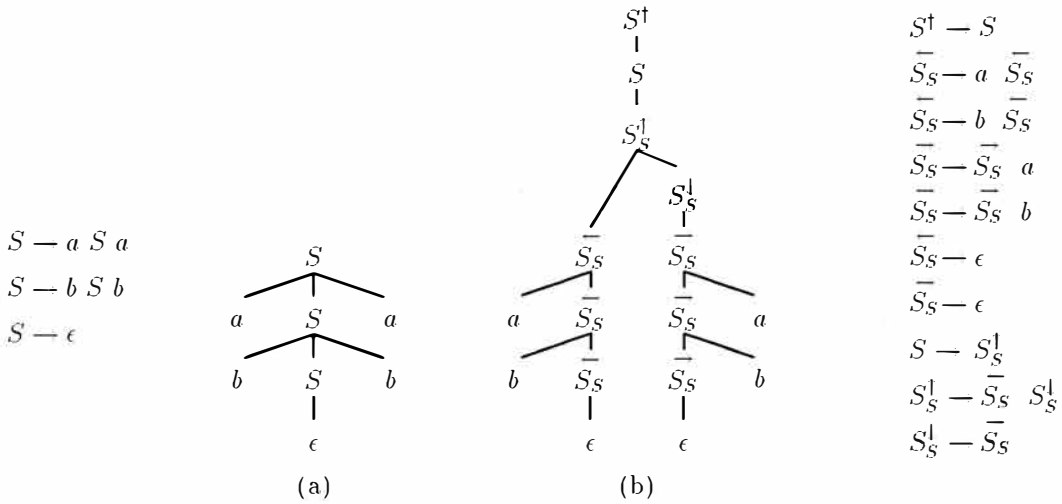


Figure 2: Parse trees for a palindrome: (a) original grammar, (b) transformed grammar (Section 4).

root to ϵ "communicate" with those to the right of the spine. More precisely, the prefix of the input up to the point where it meets the final node ϵ of the spine determines the suffix after that point, in a way that an unbounded collection of symbols from the prefix need to be taken into account.

A formal explanation for why the grammar may not generate a regular language relies on the following definition, due to [Chomsky, 1959b]:

Definition 1 *A grammar is* self-embedding *if there is some* $A \in N$*, such that* $A \Rightarrow^* \alpha A \beta$*, for some* $\alpha \neq \epsilon$ *and* $\beta \neq \epsilon$*.*

In order to avoid the somewhat unfortunate term *nonself-embedding* (or *noncenter-embedding*, as in [Langendoen, 1975]) we define a *strongly regular* grammar to be a grammar that is not self-embedding. Strong regularity informally means that when a section of a spine in a parse tree repeats itself, then either no grammar symbols occur to the left of that section of the spine, or no grammar symbols occur to the right. This prevents the "unbounded communication" between the two sides of the spine exemplified by the palindrome grammar.

Obviously, right linear and left linear grammars (as known from standard literature such as [Harrison, 1978]) are strongly regular. That right linear and left linear grammars generate regular languages is easy to show. That strongly regular grammars also generate regular languages will be proved shortly.

First, for an arbitrary grammar, we define the set of *recursive* nonterminals as:

$$\overline{N} = \{A \in N \mid \exists \alpha, \beta [A \Rightarrow^* \alpha A \beta]\}$$

We determine the partition $\mathcal{N}$ of $\overline{N}$ consisting of subsets $N_1, N_2, \ldots, N_m$, for some $m \geq 0$, of *mutually recursive* nonterminals:

$$\mathcal{N} = \{N_1, N_2, \ldots, N_m\}$$
$$N_1 \cup N_2 \cup \ldots \cup N_m = \overline{N}$$
$$N_i \cap N_j = \emptyset, \text{ for all } i, j \text{ such that } 1 \leq i < j \leq m$$
$$\exists i [A \in N_i \wedge B \in N_i] \quad \Leftrightarrow \quad \exists \alpha_1, \beta_1, \alpha_2, \beta_2 [A \Rightarrow^* \alpha_1 B \beta_1 \wedge B \Rightarrow^* \alpha_2 A \beta_2], \text{ for all } A, B \in \overline{N}$$

We now define the function *recursive* from $\mathcal{N}$ to the set $\{left, right, self, cyclic\}$:

$$\begin{aligned}
recursive(N_i) &= left, && if \quad \neg LeftGenerating(N_i) \wedge RightGenerating(N_i) \\
&= right, && if \quad LeftGenerating(N_i) \wedge \neg RightGenerating(N_i) \\
&= self, && if \quad LeftGenerating(N_i) \wedge RightGenerating(N_i) \\
&= cyclic, && if \quad \neg LeftGenerating(N_i) \wedge \neg RightGenerating(N_i)
\end{aligned}$$

where

$$\begin{aligned}
LeftGenerating(N_i) &= \exists (A \rightarrow \alpha B \beta) \in P [A \in N_i \wedge B \in N_i \wedge \alpha \neq \epsilon] \\
RightGenerating(N_i) &= \exists (A \rightarrow \alpha B \beta) \in P [A \in N_i \wedge B \in N_i \wedge \beta \neq \epsilon]
\end{aligned}$$

When $recursive(N_i) = left$, N_i consist of only left-recursive nonterminals, which does not mean it cannot also contain right-recursive nonterminals, but in that case right recursion amounts to application of unit rules. When $recursive(N_i) = cyclic$, it is *only* such unit rules that take part in the recursion.

That $recursive(N_i) = self$, for some i, is a sufficient and necessary condition for the grammar to be self-embedding. We only prove this in one direction: Suppose we have two rules $A_1 \rightarrow \alpha_1 B_1 \beta_1$, $A_1, B_1 \in N_i$, and $A_2 \rightarrow \alpha_2 B_2 \beta_2$, $A_2, B_2 \in N_i$, such that $\alpha_1 \neq \epsilon$ and $\beta_2 \neq \epsilon$. This means that $A_1 \rightarrow \alpha_1 B_1 \beta_1 \rightarrow^* \alpha_1 \alpha'_1 A_2 \beta'_1 \beta_1 \rightarrow \alpha_1 \alpha'_1 \alpha_2 B_2 \beta_2 \beta'_1 \beta_1 \rightarrow^* \alpha_1 \alpha'_1 \alpha_2 \alpha'_2 A_1 \beta'_2 \beta_2 \beta'_1 \beta_1$, for some $\alpha'_1, \beta'_1, \alpha'_2, \beta'_2$, making use of the assumption that B_1 and A_2, and then B_2 and A_1 are in the same subset N_i of mutually recursive nonterminals. In the final sentential form we have $\alpha_1 \alpha'_1 \alpha_2 \alpha'_2 \neq \epsilon$ and $\beta'_2 \beta_2 \beta'_1 \beta_1 \neq \epsilon$, and therefore the grammar is self-embedding.

A set N_i such that $recursive(N_i) = self$ thus provides an isolated aspect of the grammar that causes self-embedding, and therefore making the grammar strongly regular will depend on a solution for how to transform the part of the grammar in which nonterminals from N_i occur.

We now prove that a grammar that is strongly regular (or in other words, for all i, $recursive(N_i) \in \{left, right, cyclic\}$) generates a regular language. Our proof differs from a proof of the same fact in [Chomsky, 1959a] in that it is fully constructive: Figure 3 presents an algorithm for creating a finite automaton which accepts the language generated by the grammar.

The process is initiated at the start symbol, and from there the process descends the grammar in all ways until terminals are encountered, and then transitions are created labelled with those terminals. Descending the grammar is straightforward in the case of rules of which the left-hand side is not a recursive nonterminal: the groups of transitions found recursively for members in the right-hand side will be connected. In the case of recursive nonterminals, the process depends on whether the nonterminals in the corresponding set from $\mathcal{N}$ are mutually left-recursive or right-recursive; if they are both, which means they are cyclic, then either subprocess can be applied; in the code in Figure 3 cyclic and right-recursive subsets N_i are treated uniformly.

We discuss the case that the nonterminals are left-recursive. (The converse case is left to the imagination of the reader.) One new state is created for each nonterminal in the set. The transitions that are created for terminals and nonterminals not in N_i are connected in a way that is reminiscent of the construction of left-corner parsers [Rosenkrantz and Lewis II, 1970], and specifically of one construction that focuses on groups of mutually recursive nonterminals [Nederhof, 1994a, Section 5.8].

An example is given in Figure 4. Four states have been labelled according to the names they are given in procedure *make_fa*. There are two states that are labelled q_B. This can be explained by the fact that nonterminal B can be reached by descending the grammar from S in two essentially distinct ways.

4 Approximating a Context-Free Language

Now that we know what makes a context-free grammar violate a sufficient condition for the generated language to be regular, we have a good starting point to investigate how we should change a grammar in order to obtain a regular language. The intuition is that the "unbounded communication" between the left and right sides of spines is broken.

We concentrate on the sets N_i with $recursive(N_i) = self$. For each set separately, we apply the transformation in Figure 5. After this approximation algorithm, the grammar will be strongly regular. We will explain the transformation by means of two examples.

The first example deals with the special case that each nonterminal can lead to at most one recursive call of itself.[1] Consider the grammar of palindromes in the left half of Figure 2. The approximation algorithm leads to the grammar in the right half. Figure 2 (b) shows the effect on the structure of parse trees. Note that the left sides of former spines are treated by the new nonterminal $\overline{S}_S$ and the right sides by the new nonterminal $\overline{\overline{S}}_S$.

The general case is more complicated. A nonterminal A may lead to several recursive occurrences: $A \rightarrow^* \alpha A \beta A \gamma$. As before, our approach is to approximate the language by separating the left and right sides of spines, but in this case, several spines in a single parse tree may need to be taken care of at once.

As a presentation of this case in a pictorial way, Figure 6 (a) suggests a part of a parse tree in which all (labels of the) nodes belong to the same set N_i, where $recursive(N_i) = self$. Other nodes in the direct vicinity of the depicted part of the parse tree we assume not to be in N_i; the triangles $\triangle$, for example, denote a mother node in N_i and a number of daughter nodes *not* in N_i. The dotted lines labelled $p1, p3, p5, p7$ represent paths

[1] This is the case for *linear* context-free grammars [Hopcroft and Ullman, 1979].

```
let K = ∅, s = fresh_state, f = fresh_state, F = {f};
make_fa(s, S, f).

procedure make_fa(q0, α, q1):
  if α = ε
  then let Δ = Δ ∪ {(q0, ε, q1)}
  elseif α = a, some a ∈ Σ
  then let Δ = Δ ∪ {(q0, a, q1)}
  elseif α = Xβ, some X ∈ V, β ∈ V* such that |β| > 0
  then let q = fresh_state;
       make_fa(q0, X, q);
       make_fa(q, β, q1)
  else let A = α;  (* α must consist of a single nonterminal *)
       if A ∈ Ni, some i
       then for each B ∈ Ni do let qB = fresh_state end;
            if recursive(Ni) = left
            then for each (C — X1 ... Xm) ∈ P such that C ∈ Ni ∧ X1, ..., Xm ∉ Ni
                 do make_fa(q0, X1 ... Xm, qC)
                 end;
                 for each (C — D X1 ... Xm) ∈ P such that C, D ∈ Ni ∧ X1, ..., Xm ∉ Ni
                 do make_fa(qD, X1 ... Xm, qC)
                 end;
                 let Δ = Δ ∪ {(qA, ε, q1)}
            else "the converse of the then-part"  (* recursive(Ni) ∈ {right, cyclic} *)
            end
       else for each (A — β) ∈ P do make_fa(q0, β, q1) end  (* A is not recursive *)
       end
  end
end.

procedure fresh_state():
  create some fresh object q;
  let K = K ∪ {q};
  return q
end.
```

Figure 3: Transformation from a strongly regular grammar $G = (\Sigma, N, P, S)$ into an equivalent finite automaton $\mathcal{F} = (K, \Sigma, \Delta, s, F)$.

along nodes in N_i such that the nodes to the left of the visited nodes are not in N_i. In the case of $p2, p4, p6, p8$ the nodes to the right of the visited nodes are not in N_i.

The effect of our transformation on the structure of the parse tree is suggested in Figure 6 (b). We see that the left and right sides of spines (e.g. $p1$ and $p2$) are disconnected, in the same way as "unbounded communication" between the two sides of spines was broken in our earlier example of the palindrome grammar.

For example, consider the following grammar for mathematical expressions:

$$\begin{aligned} S &\ — \ A * B \\ A &\ — \ (A + B) \ \mid \ a \\ B &\ — \ [A] \ \mid \ b \end{aligned}$$

We have $\overline{N} = \{A, B\}$, $\mathcal{N} = \{N_1\}$, $N_1 = \{A, B\}$. Note that $recursive(N_1) = self$. After applying the approximation algorithm to N_1 we obtain:

$$
\begin{array}{lcl}
S & \rightarrow & Aa \\
A & \rightarrow & SB \\
A & \rightarrow & Bb \\
B & \rightarrow & Bc \\
B & \rightarrow & d
\end{array}
\qquad
\begin{array}{l}
\overline{N} = \{S, A, B\} \\
\mathcal{N} = \{N_1, N_2\} \\
N_1 = \{S, A\} \\
N_2 = \{B\}
\end{array}
$$

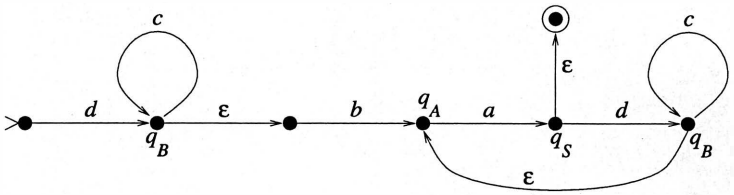


Figure 4: Application of the code from Figure 3 on a small grammar.

Assume the grammar is $G = (\Sigma, N, P, S)$. The following is to be performed for some fixed set $N_i \in \mathcal{N}$ such that $recursive(N_i) = self$.

1. If $S \in N_i$, then augment the grammar with new nonterminal $S^\dagger$ and rule $S^\dagger \rightarrow S$ and choose $S^\dagger$ to be the new start symbol of the grammar.
2. Add the following nonterminals to N: $A^\uparrow_B$, $A^\downarrow_B$, $\overleftarrow{A}_B$ and $\overrightarrow{A}_B$ for all $A, B \in N_i$.
3. Add the following rules to P, for all $A, B, C, D, E \in N_i$:
   - $\overleftarrow{A}_B \rightarrow X_1 \ldots X_m \overleftarrow{C}_B$, for all $(A \rightarrow X_1 \ldots X_m C\beta) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $\overrightarrow{A}_B \rightarrow \overrightarrow{C}_B X_1 \ldots X_m$, for all $(A \rightarrow \alpha C X_1 \ldots X_m) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $\overleftarrow{A}_A \rightarrow \epsilon$;
   - $\overrightarrow{A}_A \rightarrow \epsilon$;
   - $A \rightarrow A^\uparrow_A$;
   - $A^\uparrow_B \rightarrow \overleftarrow{A}_C X_1 \ldots X_m C^\downarrow_B$, for all $(C \rightarrow X_1 \ldots X_m) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $A^\downarrow_B \rightarrow \overrightarrow{C}_A X_1 \ldots X_m E^\uparrow_B$, for all $(D \rightarrow \alpha C X_1 \ldots X_m E\beta) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $A^\downarrow_B \rightarrow \overrightarrow{B}_A$.
4. Remove from P the old rules of the form $A \rightarrow \alpha$, where $A \in N_i$.
5. Reduce the grammar.

Figure 5: Approximation by transforming the grammar, given a set N_i.

$$
\begin{array}{lcl}
S & \rightarrow & A * B \\
\overleftarrow{A}_A & \rightarrow & (\overleftarrow{A}_A \\
\overleftarrow{B}_A & \rightarrow & [\overleftarrow{A}_A \\
\overrightarrow{A}_A & \rightarrow & \overrightarrow{B}_A) \\
\overrightarrow{A}_B & \rightarrow & \overrightarrow{B}_B) \\
\overrightarrow{B}_A & \rightarrow & \overrightarrow{A}_A] \\
\overrightarrow{B}_B & \rightarrow & \overrightarrow{A}_B]
\end{array}
\qquad
\begin{array}{lcl}
\overleftarrow{A}_A & \rightarrow & \epsilon \\
\overleftarrow{B}_B & \rightarrow & \epsilon \\
\overrightarrow{A}_A & \rightarrow & \epsilon \\
\overrightarrow{B}_B & \rightarrow & \epsilon \\
A & \rightarrow & A^\uparrow_A \\
B & \rightarrow & B^\uparrow_B
\end{array}
\qquad
\begin{array}{lcl}
A^\uparrow_A & \rightarrow & \overleftarrow{A}_A \; a \; A^\downarrow_A \\
B^\uparrow_A & \rightarrow & \overleftarrow{B}_A \; a \; A^\downarrow_A \\
B^\uparrow_A & \rightarrow & \overleftarrow{B}_B \; b \; B^\downarrow_A \\
B^\uparrow_B & \rightarrow & \overleftarrow{B}_A \; a \; A^\downarrow_B \\
B^\uparrow_B & \rightarrow & \overleftarrow{B}_B \; b \; B^\downarrow_B
\end{array}
\qquad
\begin{array}{lcl}
A^\downarrow_A & \rightarrow & \overrightarrow{A}_A + B^\uparrow_A \\
B^\downarrow_A & \rightarrow & \overrightarrow{A}_B + B^\uparrow_A \\
A^\downarrow_B & \rightarrow & \overrightarrow{A}_A + B^\downarrow_B \\
B^\downarrow_B & \rightarrow & \overrightarrow{A}_B + B^\uparrow_B \\
A^\downarrow_A & \rightarrow & \overrightarrow{A}_A \\
B^\downarrow_A & \rightarrow & \overrightarrow{A}_B \\
A^\downarrow_B & \rightarrow & \overrightarrow{B}_A \\
B^\downarrow_B & \rightarrow & \overrightarrow{B}_B
\end{array}
$$

If we compare this example to the general picture in Figure 6 (b), we conclude that a nonterminal such as $\overleftarrow{B}_A$ derives paths such as $p1, p3, p5$ or $p7$, where B was the top label at the path in the original parse tree and A occurred at the bottom. A similar fact holds for nonterminals such as $\overrightarrow{B}_A$. Nonterminals such as $B^\uparrow_A$ and

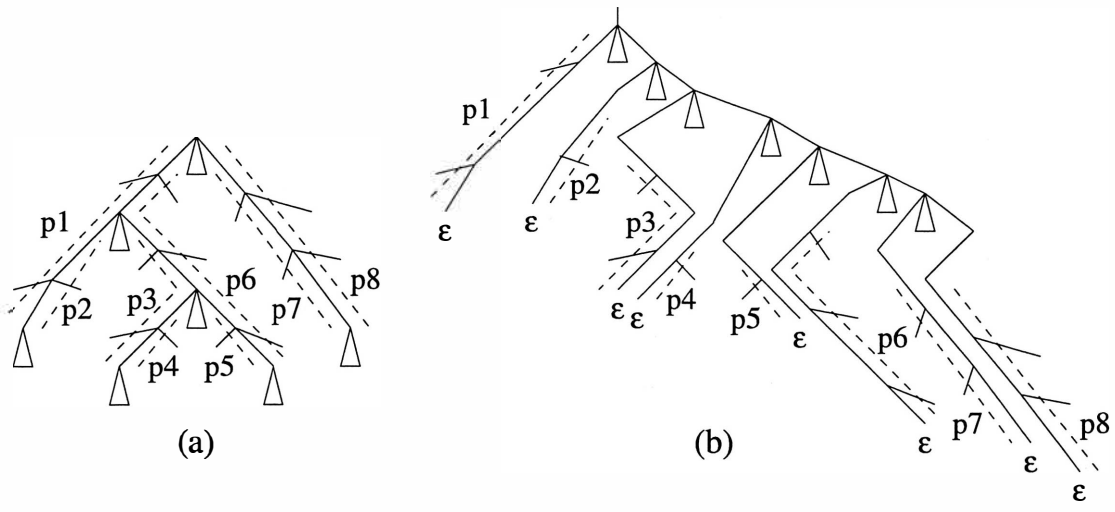


Figure 6: The general effect of the transformation on the structure of parse trees.

$B^{\downarrow}_{A}$ indicate that the root of the complete subtree was labelled A, and that the last node of the tree that was treated is labelled B; in the case of $B^{\downarrow}_{A}$ that node is at the top of a path such as $p1, p3, p5$ or $p7$ in the original tree, in the case of $B^{\downarrow}_{A}$ that node is at the bottom of a path such as $p2, p4, p6$ or $p8$.

5 Limitations

It is undecidable whether the language generated by a context-free grammar is regular [Harrison, 1978]. Consequently, the condition of strong regularity, which is decidable and is a sufficient condition for the language to be regular, cannot also be a necessary condition. This is demonstrated by the following grammar:

$$\begin{array}{lcl} S & \rightarrow & a\ A \mid B\ a \mid C \\ A & \rightarrow & a\ A \mid C \\ B & \rightarrow & B\ a \mid C \\ C & \rightarrow & a\ C\ a \mid c \end{array}$$

This (non-ambiguous) grammar generates the regular language a^*ca^*. Yet it is not strongly regular, due to the cycle corresponding to the rule $C \rightarrow a\ C\ a$. Fortunately, our algorithm transforms this grammar into a strongly regular one which generates the same language a^*ca^*.

However, in some cases a grammar which is not strongly regular and generates a regular language may be transformed into one which generates a strictly larger language. This pertains to a claim made by [Pereira and Wright, 1991] that their method is always language-preserving when the original grammar already generates a regular language, which was refuted by a revised paper by the same authors.[2] It turns out that transforming a context-free grammar generating a regular language into a finite automaton accepting the same language is an unsolvable problem [Ullian, 1967], and consequently, the method from [Pereira and Wright, 1991] cannot satisfy this property, nor can our new method.

A simple example where our method has this undesirable behaviour is the following:

$$S \rightarrow aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

This grammar generates the regular language of all strings over $\{a, b\}$ of even length. Our approximation however results in the exact same grammar as in the example of palindromes. This grammar generates the regular language of *all* strings over $\{a, b\}$ — not only those of even length.[3]

[2] Posted on the Computation and Language E-Print Archive, as number 9603002.

[3] The method from [Pereira and Wright, 1991] does a little better: of the strings of odd length it excludes those of length 1; yet it does allow all strings of length 3, 5, ... The two methods are compared more closely in Section 7.

If we represent context-free languages by means of pushdown automata, we can define a subclass for which regularity is decidable, namely those that allow a deterministic pushdown automaton. If such a deterministic language is regular, we can furthermore construct an equivalent deterministic finite automaton [Stearns, 1967]. It turns out that even for this restricted class of context-free languages, the construction of corresponding finite automata is quite complex: The (deterministic) finite automata may require a number of states that is a double exponential function in the size of the original deterministic pushdown automata [Meyer and Fischer, 1971]; [Valiant, 1975] has shown that the *upper* bound to the number of states is also a double exponential function.

For arbitrary context-free grammars that describe regular languages, no recursive function in the size of grammars exists that provides an upper bound to the number of states of equivalent finite automata [Meyer and Fischer, 1971].[4]

6 Refinement

Our approximation algorithm is such that the two sides of spines are disconnected for all nonterminals that are involved in self-embedding (i.e. those in some fixed N_i with $recursive(N_i) = self$). One can however retain a finite amount of self-embedding by unfolding j levels of applications of nonterminals from N_i before the approximation algorithm is applied. This is done in such a way that in those j levels no recursion takes place and precision thus remains unaffected.

More precisely, for each nonterminal $A \in N_i$ we introduce j fresh nonterminals $A[1], \ldots, A[j]$, and we change the rules of the (augmented) grammar as follows.

For each $A \rightarrow X_1 \cdots X_m$ in P such that $A \in N_i$, and h such that $1 \leq h \leq j$, we add $A[h] \rightarrow X'_1 \cdots X'_m$ to P, where

$$\begin{aligned} X'_k &= X_k[h+1], \text{ if } X_k \in N_i \wedge h < j \\ &= X_k, \text{ otherwise} \end{aligned}$$

Further, we replace all rules $A \rightarrow X_1 \cdots X_m$ such that $A \notin N_i$ by $A \rightarrow X'_1 \cdots X'_m$, where

$$\begin{aligned} X'_k &= X_k[1], \text{ if } X_k \in N_i \\ &= X_k, \text{ otherwise} \end{aligned}$$

If we take $j = 3$, the (augmented) palindrome grammar becomes:

$$\begin{array}{rcl} S^{\dagger} & \rightarrow & S[1] \\ S[1] & \rightarrow & a\ S[2]\ a \mid b\ S[2]\ b \mid \epsilon \\ S[2] & \rightarrow & a\ S[3]\ a \mid b\ S[3]\ b \mid \epsilon \\ S[3] & \rightarrow & a\ S\ a \mid b\ S\ b \mid \epsilon \\ S & \rightarrow & a\ S\ a \mid b\ S\ b \mid \epsilon \end{array}$$

After applying the approximation algorithm, all generated strings up to length 6 are palindromes. Only generated strings longer than 6 may not be palindromes: these are of the form $wvv'w^R$, for some $w \in \{a,b\}^3$ and $v, v' \in \{a,b\}^*$, where w^R indicates the mirror imagine of w. Thus the outer 3 symbols left and right do match, but not the innermost symbols in both "halves". In general, by choosing j high enough we can obtain approximations that are language-preserving up to a certain string length, provided the grammar is not cyclic.[5] A more complicated way of achieving such approximations, using refinement of the method from [Pereira and Wright, 1991], was discussed in [Rood, 1996].

This language-preserving transformation in effect decorates nodes in the parse tree with numbers up to j indicating the distance to the nearest ancestor node not in N_i. The second refinement we discuss has the effect of indicating the distance up to j to the furthest descendent not in N_i.

For this refinement, we again introduce j fresh nonterminals $A[1], \ldots, A[j]$ for each $A \in N_i$. Each $A \rightarrow X_1 \cdots X_m$ in P is replaced by a collection of other rules, which are created as follows. We determine the (possibly empty) list $k_1, \ldots, k_p$ of ascending indices of members that are in N_i: $\{k_1, \ldots, k_p\} = \{k \mid 1 \leq k \leq m \wedge X_k \in N_i\}$

[4] This is equivalent to stating that transformation of such a context-free grammar into a finite automaton accepting the same language is an unsolvable problem, which we mentioned before.

[5] This problem of cyclic grammars can be easily overcome, but this is beyond the scope of the present paper.

and $k_1 < \ldots < k_p$. For each list of p numbers $n_1, \ldots, n_p \in \{1, \ldots, j, j+1\}$ we create the rule $A' \rightarrow X'_1 \cdots X'_m$, where

$$\begin{aligned} X'_k &= X_k[n_k], \text{ if } X_k \in N_i \wedge n_k \leq j \\ &= X_k, \text{ otherwise} \\ A' &= A[h+1], \text{ if } A \in N_i \wedge h < j, \text{ where } h = \max_{1 \leq k \leq p} n_k \\ &= A, \text{ otherwise} \end{aligned}$$

We assume that h evaluates to 0 if $p = 0$. Note that $j+1$ is an auxiliary number which does not show up in the transformed grammar; it represents that the distance to the nearest ancestor node not in N_i is more than j.

For the running example, with $j = 3$, we obtain:

$$\begin{aligned} S^\dagger &\rightarrow S \mid S[3] \mid S[2] \mid S[1] \\ S &\rightarrow a\ S\ a \mid b\ S\ b \mid a\ S[3]\ a \mid b\ S[3]\ b \\ S[3] &\rightarrow a\ S[2]\ a \mid b\ S[2]\ b \\ S[2] &\rightarrow a\ S[1]\ a \mid b\ S[1]\ b \\ S[1] &\rightarrow \epsilon \end{aligned}$$

Approximation now results in palindromes up to length 6, but further in strings vww^Rv', for some $w \in \{a, b\}^3$ and $v, v' \in \{a, b\}^*$, where it is the innermost, not the outermost, parts that still have the characteristics of palindromes.

7 Comparison

The main purpose of this paper is to clarify what happens during the process of finding regular approximations of context-free languages. Our grammar-based method represented by Figure 5 can be seen as the simplest approach to remove self-embedding, and as we have shown, removing self-embedding is sufficient for obtaining a regular language.

Given its simplicity, it is not surprising that more sophisticated methods, such as the LR-based method from [Pereira and Wright, 1991], produce strictly more precise approximations, i.e. regular languages that are smaller, in terms of language inclusion $\subseteq$. However, our empirical experiments have shown that such sophistication sometimes deteriorates rather than improves practical usefulness of the method.

For example, the appendix of [Pereira and Wright, 1991] contains a small grammar which is already strongly regular. The authors of that paper report they obtain a finite automaton with 2615 states and 4096 transitions before determinization and minimization. For our method however, no approximation at all is needed, and construction of the finite automaton by means of the algorithm in Figure 3 produces a finite automaton with only 957 states and 2751 transitions. After determinization and minimization of course the same automaton results. This example suggests that much of the "sophistication" of the LR-based method entails wasted computational overhead.

We have also considered the grammar in Section 9 of [Church and Patil, 1982] and the 4-line grammar of noun phrases from [Pereira and Wright, 1991]. Neither of the grammars are strongly regular, yet they generate regular languages. For the first grammar, the LR-based and the grammar-based methods give the same results. For the second grammar, the two methods give the same result only provided the second refinement from Section 6 is incorporated into our grammar-based method, taking $j = 1$.

That the grammar-based method in general produces less precise approximations may seem a weakness: *ad hoc* refinements such as those discussed in Section 6 may be needed to increase precision. Our viewpoint is that the LR-based method also incorporates ad hoc mechanisms of obtaining approximations that are more precise than what is minimally needed to obtain a regular language, but that these are however outside of the control of the user.

A case in point is the grammar of palindromes. In Section 6 we demonstrated how precision for our method can be improved in a controlled manner. However, the LR-based method forces a result upon us which is given by $(a\{a,b\}^*a \Leftrightarrow a(ba)^*) \cup (b\{a,b\}^*b \Leftrightarrow b(ab)^*)$; in other words, just the left-most and right-most symbols are matched, but alternating series of a's and b's are excluded. This strange approximation is reached due to some intricate aspect of the structure of the LR automaton, and there is no reason to consider it as more natural

or desirable than any other approximation, and although the LR-based method allows additional refinement as well, as shown by [Rood, 1996], the nature of such refinement may be that even more of the same kind of idiosyncrasies is introduced.

In addition, we point out that construction of an LR automaton, of which the size is exponential in the size of the grammar, may be a prohibitively expensive task in practice [Nederhof and Satta, 1996]. This is however only a fraction of the effort needed for the unfolding step of the LR-based method, which is in turn exponential in the size of the LR automaton. This became apparent when we tried to apply two independently developed implementations of the LR-based method on a subgrammar for the Java programming language. Both implementations crashed due to excessive use of memory, which is in keeping with our estimates that the number of LR stacks that need to be considered in this case may be astronomical. By contrast, the complexity of our approximation algorithm (Figure 5) is polynomial. The only exponential behaviour may come from the subsequent construction of the finite automaton (Figure 3), when the grammar is descended in all ways, a source of exponential behaviour which is also part of the LR-based method. Our implementation produced a nondeterministic finite automaton from the Java subgrammar within 12 minutes.

Recently, [Grimley Evans, 1997] has proposed a third approach, which I rephrase as follows. We consider a context-free grammar as a recursive transition network [Woods, 1970]: for each rule $A \rightarrow X_1 \cdots X_m$ we make a finite automaton with states $q_0, \ldots, q_m$ and transitions (q_{i-1}, X_i, q_i), $1 \leq i \leq m$. These automata are then joined: Each transition (q_{i-1}, B, q_i), where B is a nonterminal, is replaced by a set of ϵ-transitions from q_{i-1} to the "left-most" states for rules with left-hand side B, and conversely, a set of ϵ-transitions from the "right-most" states for those rules to q_i.

This essentially replaces recursion by ϵ-transitions, which leads to a crude approximation. An additional mechanism is now introduced that ensures that the list of visits to the states $q_0, \ldots, q_m$ belonging to a certain rule satisfies some reasonable criteria: a visit to q_i, $0 \leq i < m$, should be followed by one to q_{i+1} or q_0. The latter option amounts to a nested incarnation of the rule. Similarly there is a condition for what should precede a visit to q_i, $0 < i \leq m$. Since only pairs of consecutive visits to states from the set $\{q_0, \ldots, q_m\}$ are considered, finite-state techniques suffice to implement such conditions.

If we compare that approach to our grammar-based method combined with e.g. the second refinement from Section 6, we can roughly say that the difference is that while our approach provides exact approximations for trees up to a certain height, the approach by [Grimley Evans, 1997] provides an exact approximation when no nested incarnations of individual rules occur. This emphasis on treating rules individually has the consequence that the order of terminals in a string can become mixed-up even when the approximation is still exact with respect to the *number* of occurrences of terminals; in effect, distinct rules interact in a way that is not consistent with the recursive structure of the original grammar.

It seems that the approach by [Grimley Evans, 1997] always results in approximations that are more precise than our approach without the refinements from Section 6.

An important difference of our grammar-based method with both the LR-based method and the one from [Grimley Evans, 1997] is that the structure of the context-free grammar is retained as long as possible as much as possible. This has two advantages. First, the remnants of the original structure present in the transformed grammar, including the names of the nonterminals, can be incorporated into the construction of the finite automaton, in such a way that the automaton produces output which can be used to build parse trees according to the original grammar. The algorithm from Figure 3 can be easily extended to construct such a finite *transducer*, using ideas from [Langendoen, 1975; Krauwer and des Tombe, 1981; Langendoen and Langsam, 1990].

Secondly, the approximation process itself can be monitored easily: The author of a grammar can still see the structure of the old grammar in the new strongly regular grammar, and can in this way observe what kind of consequences the approximation has on the generated language.

Regarding other related work, I hesitate to mention [Langendoen and Langsam, 1987], which seems to be concerned more with psycho-linguistic arguments than with any level of mathematical accuracy. As the LR-based method, this approach seems to be based on pushdown automata, which here implement a particularly incorrect variant of left-corner parsing, allowing ungrammatical sentences to be accepted, which seems to be unintentional. The manner in which finiteness of the automaton is achieved is by requiring a collapse of a certain group of elements already in the parsing stack (if such a group of elements is at all present) into a smaller combination of stack elements, upon finding that some stack element has more than 2 occurrences. This collapse represents a forced attachment of previously unconnected subtrees in the parse tree. The effect is

obviously a reduction of the language. However, due to the incorrectness of the variant of left-corner parsing, the resulting language may also contain sentences *not* in the original language. This makes comparison with other methods difficult, if not pointless.

The intended purpose of the above work may however be to reduce, as opposed to extend, a context-free language to become an (infinite) regular language.

Related to this is [Krauwer and des Tombe, 1981]: two kinds of pushdown automaton are presented that implement top-down and left-corner parsing, respectively. The regular approximation results simply by putting an upper-limit on the size of the stack, thus restricting the language. Unpublished work by Mark Johnson[6] solves the task in the same vein. The idea is extended to feature grammars in [Black, 1989].

Some theoretical limitations of these ideas have been investigated by [Ullian, 1967]: Given a context-free language, it is undecidable whether an infinite regular subset exists; yet, given that it exists, it can be computed. Note that for practical purposes one is interested in determining a "large" regular subset, not just any infinite subset of a context-free language as in the theorem from [Ullian, 1967].

Another way to obtain a finite-state approximation of a grammar is to retain only the information about allowable pairs (or triples, etc.) of adjacent parts of speech (cf. bigrams, trigrams, etc.). This simple approach is proposed by [Herz and Rimon, 1991], and is reported to be effective for the purpose of word tagging in Hebrew. (For extension to probabilistic formalisms, see [Stolcke and Segal, 1994].)

Acknowledgements

Special thanks go to René Deist for collaboration during my stay at the University of Bochum. I thankfully acknowledge valuable discussions with Gertjan van Noord, Mark Johnson, Eberhard Bertsch, Anton Nijholt and Rieks op den Akker.

This research was carried out within the framework of the Priority Programme Language and Speech Technology (TST). The TST-Programme is sponsored by NWO (Dutch Organization for Scientific Research). Further support was obtained from the German Research Foundation (DFG), under grant Be1953/1-1.

References

[Black, 1989] A.W. Black. Finite state machines from feature grammars. In *International Workshop on Parsing Technologies*, pages 277–285, Pittsburgh, 1989.

[Chomsky, 1959a] N. Chomsky. A note on phrase structure grammars. *Information and Control*, 2:393–395, 1959.

[Chomsky, 1959b] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[Church and Patil, 1982] K. Church and R. Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8:139–149, 1982.

[Grimley Evans, 1997] E. Grimley Evans. Approximating context-free grammars with a finite-state calculus. In *35th Annual Meeting of the ACL*, pages 452–459, Madrid, Spain, July 1997.

[Harrison, 1978] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[Heckert, 1994] E. Heckert. *Behandlung von Syntaxfehlern für LR-Sprachen ohne Korrekturversuche*. PhD thesis, Ruhr-Universität Bochum, 1994.

[Herz and Rimon, 1991] J. Herz and M. Rimon. Local syntactic constraints. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 200–209, Cancun, Mexico, February 1991.

[Hopcroft and Ullman, 1979] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[6] *Left Corner Transforms and Finite State Approximations*, draft, May 1996.

[Krauwer and des Tombe, 1981] S. Krauwer and L. des Tombe. Transducers and grammars as theories of language. *Theoretical Linguistics*, 8:173–202, 1981.

[Langendoen and Langsam, 1987] D.T. Langendoen and Y. Langsam. On the design of finite transducers for parsing phrase-structure languages. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 191–235. John Benjamins Publishing Company, Amsterdam, 1987.

[Langendoen and Langsam, 1990] D.T. Langendoen and Y. Langsam. A new method of representing constituent structures. *Annals New York Academy of Sciences*, 583:143–160, 1990.

[Langendoen, 1975] D.T. Langendoen. Finite-state parsing of phrase-structure languages and the status of readjustment rules in grammar. *Linguistic Inquiry*, 6(4):533–554, 1975.

[Meyer and Fischer, 1971] A.R. Meyer and M.J. Fischer. Economy of description by automata, grammars, and formal systems. In *IEEE Conference Record of the 12th Annual Symposium on Switching and Automata Theory*, pages 188–191, 1971.

[Nederhof and Satta, 1996] M.J. Nederhof and G. Satta. Efficient tabular LR parsing. In *34th Annual Meeting of the ACL*, pages 239–246, Santa Cruz, California, USA, June 1996.

[Nederhof, 1994a] M.J. Nederhof. *Linguistic Parsing and Program Transformations*. PhD thesis, University of Nijmegen, 1994.

[Nederhof, 1994b] M.J. Nederhof. An optimal tabular parsing algorithm. In *32nd Annual Meeting of the ACL*, pages 117–124, Las Cruces, New Mexico, USA, June 1994.

[Pereira and Wright, 1991] F.C.N. Pereira and R.N. Wright. Finite-state approximation of phrase structure grammars. In *29th Annual Meeting of the ACL*, pages 246–255, Berkeley, California, USA, June 1991.

[Rood, 1996] C.M. Rood. Efficient finite-state approximation of context free grammars. In A. Kornai, editor, *Extended Finite State Models of Language*, Proceedings of the ECAI'96 workshop, pages 58–64, Budapest University of Economic Sciences, Hungary, August 1996.

[Rosenkrantz and Lewis II, 1970] D.J. Rosenkrantz and P.M. Lewis II. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152, 1970.

[Stearns, 1967] R.E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11:323–340, 1967.

[Stolcke and Segal, 1994] A. Stolcke and J. Segal. Precise n-gram probabilities from stochastic context-free grammars. In *32nd Annual Meeting of the ACL*, pages 74–79, Las Cruces, New Mexico, USA, June 1994.

[Ullian, 1967] J.S. Ullian. Partial algorithm problems for context free languages. *Information and Control*, 11:80–101, 1967.

[Valiant, 1975] L.G. Valiant. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM*, 22(1):1–10, 1975.

[Woods, 1970] W.A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, October 1970.

A LEFT-TO-RIGHT TAGGER FOR WORD GRAPHS

Christer Samuelsson
Bell Laboratories, Lucent Technologies
600 Mountain Ave, Room 2D-339
Murray Hill, NJ 07974, USA
Email: christer@research.bell-labs.com

Abstract

An algorithm is presented for tagging input word graphs and producing output tag graphs that are to be subjected to further syntactic processing. It is based on an extension of the basic HMM equations for tagging an input word-string that allows it to handle word-graph input, where each arc has been assigned a probability. The scenario is that of some word-graph source, e.g., an acoustic speech recognizer, producing the arcs of a word graph, and the tagger will in turn produce output arcs, labelled with tags and assigned probabilities. The processing as done entirely left-to-right, and the output tag graph is constructed using a minimum of lookahead, facilitating real-time processing.

1 Background

Ever since [Church 1988], HMM-based part-of-speech (PoS) tagging has been very popular. The task of the particular PoS tagger described in the current article is to act as a module for lexical lookup that is capable of performing some local pruning, and the output is passed on to subsequent syntactic processing. For this reason, it is desirable to retain some ambiguity in the output, namely that which cannot be accurately removed at this stage of processing. This is very similar in spirit to the lexical-lookup and lexical-pruning stages of the SRI Core Language Engine, see [Rayner & Carter 1996], and the lexical-lookup phase and subsequent morphological disambiguation of the English Constraint-Grammar Parser of Helsinki, see [Karlsson *et al* (eds.) 1995].

The tagger is a component of a system employing a pipeline architecture with the design philosophy that at some processing level, a pruned search space, e.g., a word graph, is the input of one module, which produces an output pruned search space, e.g., a tag graph or preterminal chart, at the next processing level, which in turn serves as the input of a subsequent module, thus closing the circle.

It is also highly desirable, both for the goal of mimicking human behavior and for the more practical reason of achieving real-time processing that the tagger process the input left-to-right. This means that the tagger will continuously construct the arcs of the output tag graph as soon as enough input has been seen to do so. This in turn requires that the amount of lookahead employed by the tagger be kept to a minimum.

The work described in the current paper is based on previous theoretical work by the author on extending the basic HMM equations for word-string tagging to handle word-graph input, as described in detail in [Samuelsson 1997]. Section 2 below provides the final equations arrived at, together with some explanations; the reader is referred to the above publication for the underlying theory and their formal derivations. Section 3 constitutes the new body of work, describing the new left-to-right tagging algorithm and motivating the simplification made to achieve it. The algorithm has been implemented and works satisfactorily. The implementation itself is a straight-forward realization of the described algorithm, and of little interest. No serious empirical evaluation has yet been attempted, partly due to the lack of appropriate test material.

2 The Equations

The following is an extension of the basic HMM equations given in for example [Rabiner 1989], pp. 272–274, or [Krenn & Samuelsson 1997], pp. 42–46, to the case where the input is a word graph, rather than a word string.

By first producing a tag graph from the word graph, we reduce the problem of tagging an input word graph to that of applying an N-gram tag model to an input tag graph. This is done by defining

$$b_{rtj} \quad = \quad P(L_{rt} = l_j \mid G_r) \quad = \quad \sum_k P(L_{rt} = l_j \mid W_{rt} = k) \cdot P(W_{rt} = k \mid G_r) \tag{1}$$

where $P(W_{rt} = k \mid G_r)$ is the probability of word candidate k from node r to node t in the input word graph. The conditioning on G_r indicates that the probability distribution is the one of node r, i.e., that over the outgoing arcs of node r. $P(L_{rt} = l_j \mid W_{rt} = k)$ is the lexical tag probability, a model parameter estimated from training data.

We can thus recast the problem as that of applying an N-gram model to reestimate the arc probabilities of a labelled directed acyclic graph, where a probability distribution over the outgoing arcs has been associated with each node. We require that there be unique globally minimal and maximal elements, the start and end nodes of the graph. This is often referred to as a lattice, although strictly speaking it is not, as we only require a unique global minimal element, not one for any given node pair.

We introduce a(ny) complete ordering $<$ of the nodes of the graph which does not violate the topological ordering defined by the arcs. We refer to each node through its rank in this ordering, the node number, and we let 0 be the start node and T be the end node.

We recursively calculate the α, β, γ, δ and ϵ variables for each node (pair) $\{(r,t) : 0 \leq r < t = 1, \ldots, T\}$ and each possible state $i, j = 1, \ldots, n$. The states encode the N-gram model, and each state corresponds to a label sequence. In the bigram case, each state is a label, in the trigram case, a label pair, etc.

$$\alpha(t,j) \quad = \quad P(S_t = j \mid \mathbf{G}_{<t}) \quad = \quad \sum_r \epsilon(r,t,j) \tag{2}$$

This is the probability of being in state j at node t, given that you start from the start node.

$$\beta(r,i) \quad = \quad P(\langle end \rangle \mid S_r = i; \mathbf{G}_{\geq r}) \quad \approx \quad \sum_{t,j} q(r,t,i,j) \cdot \beta(t,j) \tag{3}$$

This is the probability of reaching the end node, given that you start in node r and that you are in state i.

$$\gamma(r,t,j) \quad = \quad P(S_t = j; \langle end \rangle \mid S_r = \bullet; \mathbf{G}) \quad \approx \quad \frac{\epsilon(r,t,j) \cdot \beta(t,j)}{\sum_{t,j} \epsilon(r,t,j) \cdot \beta(t,j)} \tag{4}$$

This is the joint probability of transiting from node r to node t, being in state j at node t, and reaching the end node, given that you start in node r.

$$\delta(t,j) \quad = \quad \max_{\mathbf{S}_{<t}} P(\mathbf{S}_{<t}; S_t = j \mid \mathbf{G}_{<t}) \quad \approx \quad \max_{r,i} \left[\delta(r,i) \cdot q(r,t,i,j)\right] \tag{5}$$

This is the joint probability of being in state j at node t and reaching it through the most likely node-state sequence, given that you start from the start node.

$$\epsilon(r,t,j) \quad = \quad P(S_r = \bullet; S_t = j \mid \mathbf{G}_{<t}) \quad \approx \quad \sum_i \alpha(r,i) \cdot q(r,t,i,j) \tag{6}$$

This is the joint probability of transiting from node r to node t and being in state j at node t, given that you start in the start node.

$$q(r,t,i,j) \quad = \quad P(S_t = j \mid S_r = i; G_r) \quad = \quad P(L_{rt} = l_j \mid S_r = i; G_r) \quad \approx \quad \frac{p_{ij} \cdot b_{rtj}}{p_j}$$

This is the joint probability of transiting from node r to node t and being in state j at node t, given that you start in node r and that you are in state i. Note that for N-grams with $N > 2$, l_i may equal l_j for distinct i and j.

$$b_{rtj} \quad = \quad P(L_{rt} = l_j \mid G_r)$$

This is the input arc probability of Eq. (1) as explained above.

$$p_{ij} \quad = \quad P(L_{x\bullet} = l_j \mid S_x = i)$$

This is the N-gram transition probability, a model parameter estimated from training data.

$$p_j \quad = \quad P(L_{\bullet\bullet} = l_j)$$

This is the label unigram probability, a model parameter estimated from training data.

We also need the initializations

$$\alpha(\bullet, j) \;=\; \delta(0, j) \;=\; \begin{cases} 1 & \text{if } j \text{ is the initial state} \\ 0 & \text{otherwise} \end{cases}$$

$$\beta(T, i) \;=\; \begin{cases} 1 & \text{if } i \text{ is a final state} \\ 0 & \text{otherwise} \end{cases}$$

Here $\mathbf{G}$ is the input graph, $\mathbf{G}_{<r}$ is the portion of the input graph from nodes 0 to r–1, $\mathbf{G}_{\geq t}$ is the portion of the input graph from nodes t to T, and G_r is the portion of the graph associated with node r, referring to the input probability distribution over the outgoing arcs of node r.

$S_t = j$ is the event of being in state j at node t, which will uniquely determine the label of the previous N–1 arcs leading to node t, and $S_r = \bullet$ is the event of being in some state at node r, i.e., of visiting node r. If in some expression $P(\ldots S_r = \ldots S_t = \ldots)$ or $P(\ldots S_t = \ldots \mid \ldots S_r = \ldots)$, where $r < t$, there is no node $s : r < s < t$, we will interpret this as implying an immediate transition from node r to node t.[1]

The δ variables are used to find the most probable node and state sequence $\mathbf{i}$, the latter which uniquely determines the most probable node and label sequence:

$$\mathbf{i} \;=\; \langle \tau_1, \iota_1 \rangle, \ldots, \langle \tau_m, \iota_m \rangle \;=\; \operatorname*{argmax}_{\mathbf{S}} P(\mathbf{S} \mid \mathbf{G})$$

This can be obtained by tracing the nodes and states for which the maximum was obtained in each application of the recurrence equation for the δ variables:

$$\langle \tau_m, \iota_m \rangle \;=\; \langle T, \iota_m \rangle \;=\; \operatorname*{argmax}_{j} \delta(T, j)$$

$$\langle \tau_{p-1}, \iota_{p-1} \rangle \;=\; \operatorname*{argmax}_{r,i} \left[\delta(r, i) \cdot q(r, \tau_p, i, \iota_p) \right]$$

In practice, the argmax will be constructed during the maximization procedure, and retained as a back pointer for each δ variable. In the interest of clarity and brevity, we have omitted these back pointers from Eq. (5).

The γ variables are used to estimate the probability of a particular label being assigned to a particular node pair:

$$P(L_{rt} = l_j \mid \mathbf{G}) \quad = \quad \sum_{i:\, l_i = l_j} P(S_r = \bullet : S_t = i \mid \mathbf{G}) \quad = \quad \sum_{i:\, l_i = l_j} \gamma(r, t, i) \tag{7}$$

This is the sum over all states whose last label is l_j. The γ variables allow us to retain ambiguity in the output, while pruning low-probability labels. A similar strategy is adopted for word-string input in [de Marcken 1990].

It is easy to verify, that using the above equations, the optimal label sequence and the reestimated arc probabilities can be constructed in $O(L^N T^3)$ time, where L is the cardinality of the label set, N is the "N" in "N-gram", and T is the number of input graph nodes. This is because in the worst case we have to scan to the beginning or end of the graph in the recurrence step, while constructing accumulators for each node pair. However, if there is a maximum arc length, which is a common situation, we recover the $O(L^N T)$ time complexity of the input-string case.

[1] If there is no arc from r to t in $\mathbf{G}$, the probability in question is simply zero.

3 The Algorithm

The problem with the traditional approach, i.e., using the δ variables to find the most likely tag sequence, is firstly that it is geared towards a single best tag sequence, and secondly requires scanning to the end of the current input before producing its output. A third problem is that of assigning probabilities to the arcs of the output tag graph. The first two problems can certainly be amended for example by maintaining multiple best candidates at each node and state, and start tracing back from the best candidates already before the end of the input is reached. The third problem can perhaps be solved by determining the output probabilities from the value of the relevant δ accumulators.

We will however choose another approach based on the γ variables, from which the output arc probabilities can be directly estimated. These variables too, as the equations stand, require that the input be processed in its entirety before any output arcs can be produced. The key observation here is that the β variables can be approximated with a reasonable degree of accuracy without scanning the entire input. We realize that without taking the N-gram model into account, the β variables will all be 1, if the input graph is consistent. A first, crude approximation would simply set them to 1, and only use the ϵ variables to calculate the γ variables. An improvement on this, that costs little, is to set them to 1 in the β-variable recurrence equation:

$$\beta(r,i) \;=\; P(\langle end\rangle \mid S_r = i; \mathbf{G}_{\geq r}) \;\approx\; \sum_{t,j} q(r,t,i,j)\cdot\beta(t,j) \;\approx\; \sum_{t,j} q(r,t,i,j)\cdot 1 \tag{8}$$

This is exactly what the tagging algorithm does.

When devising the tagging algorithm, we capitalize on the requirement that lexicographical ordering respect the topological one, and on the complete ordering introduced in Section 2.

The status of any node is one of EMPTY, PARTIAL, FULL, PENDING or FINISHED:

- EMPTY means that no inbound or outbound arcs have yet been seen.
- PARTIAL means that at least one inbound or outbound arc has been seen, but not all outbound arcs.
- FULL means that all outbound arcs have been seen, but that the β and ϵ accumulators have not yet been constructed.
- PENDING means that the β and ϵ accumulators have been constructed, using Eq. (8) and Eq. (6) respectively, but not the γ accumulators. Also, the contributions from the outgoing arcs to the α accumulators of their end nodes have been recorded according to Eq. (2).
- FINISHED means that the γ accumulators have been constructed using Eq. (4), and the reestimated arc probabilities of Eq. (7) have been passed on to the next processing step.

By the Last Node of some node r, we mean the greatest node t for which there exists an arc from node r to node t.

We define two distinguished nodes Front and Back with the following properties:

- All nodes prior to Front are either FINISHED or PENDING (or EMPTY but discarded).
- All nodes prior to Back are FINISHED (or EMPTY but discarded).
- Back is prior to Front.

If all nodes prior to some FULL node t are PENDING or FINISHED, then the α accumulators for node t are completed, and we can construct the ϵ accumulators for node t. This is also a convenient time to construct the β accumulators for node t, and to update the α accumulators for the end nodes of the arcs from node t.

1. Since all nodes prior to Front are PENDING or FINISHED, we can advance Front until a PARTIAL node is encountered, in the process converting the FULL nodes into PENDING nodes by constructing the ϵ and β accumulators, and updating subsequent α accumulators. Any EMPTY nodes encountered in the process can be discarded, since they are unreachable from the start node, due to the relationship between the node numbering and the topological ordering.

2. Since all nodes prior to Front have completed β accumulators and all PENDING nodes have completed ϵ accumulators, any PENDING node whose Last Node is prior to Front can be converted into a FINISHED node by constructing its γ accumulators and outputting the reestimated arc probabilities. We can thus scan to Front for such nodes, taking this action. Since all nodes prior to Back are FINISHED, we can start the scan from Back.

3. We now advance Back over all FINISHED nodes, again discarding EMPTY nodes, and await the arrival of more incoming arcs, converting EMPTY nodes to PARTIAL nodes and PARTIAL nodes to FULL nodes. Whenever Front becomes FULL, we goto 1.

We also need to know the start node to initialize Front, Back and the α variables. And we need to know the end node to allow breaking the recursion and proceeding to the next input graph, when the current one has been completely processed.

The algorithm allows the input arcs to arrive in any order they wish, and passes them on, appropriately processed, as quickly as the mathematical model allows. The mathematical model in question is the basic set of HMM equations found everywhere, extended to word-graph input, as described in [Samuelsson 1997], with the extra approximation in the β-variable recursion Eq. (8). The key issue is to introduce a complete node ordering that respects the topological ordering of the arcs of the input word graph. By doing this, the nodes can be processed in number order as soon as enough of the remaining input has been seen. The Back pointer indicates how far complete processing has progressed. The Front and Last Node pointers indicate how far sufficient information has been seen to complete processing.

4 Summary

The algorithm presented in the current article is based on an extension of the basic HMM equations for tagging an input word-string that allows it to handle word-graph input, where each arc has been assigned a probability. The produced output is a probabilistic tag graph, or chart, that is further processed in subsequent syntactic parsing steps.

The processing is done entirely left-to-right, and the output tag graph is constructed using a minimum of lookahead, facilitating real-time transduction of the input arcs of a word graph, generated by some word-graph source, e.g., an acoustic speech recognizer, into output arcs labelled with tags and assigned probabilities.

There are points of similarity with the presented approach and that of using weighted transductions to transduce a probabilistic word graph into a probabilistic (lexical) tag graph, and in a subsequent transduction step apply a probabilistic N-gram model, see [Pereira *et al* 1994]. There, the resulting tag graph is effectively the (probabilistic) intersection of the lexical tag graph with the tag N-gram model, where the nodes of the output graph encode the set of possible predecessor tags, thus changing the topology of the lexical tag graph, and generally drastically increasing its size. Their implementation allows realtime processing mimicking the behavior of the method presented in the current article. However, the method of the current article retains the topology of the lexical tag graph, and merely readjusts the arc probabilities, thus limiting the size of the output graph.

Future and ongoing work includes extending the scheme in both directions, i.e., to handle preceding phoneme-to-word transduction and subsequent phrasal parsing respectively.

References

[Church 1988] Kenneth W. Church. 1988. "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text". In *Procs. 2nd Conference on Applied Natural Language Processing*, pp. 136–143, ACL, 1988.

[Karlsson *et al* (eds.) 1995] F. Karlsson, A. Voutilainen, J. Heikkilä and A. Anttila (eds.). 1995. *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text.* Berlin and New York: Mouton de Gruyter, 1995.

[Krenn & Samuelsson 1997] Brigitte Krenn and Christer Samuelsson. 1994–1997. *The Linguist's Guide to Statistics.* Version of May 21, 1997. `http://coli.uni-sb.de/~christer`.

[de Marcken 1990] Carl G. de Marcken. 1990. "Parsing the LOB Corpus". In *Procs. 28th Annual Meeting of the Association for Computational Linguistics*, pp. 243–251, ACL, 1990.

[Pereira *et al* 1994] Fernando Pereira, Michael Riley and Richard Sproat. 1994. "Weighted Rational Transductions and their Application to Human Language Processing". In *Procs. of the Human Language Technology Workshop*, pp. 249–254, Morgan Kaufmann, 1994.

[Rabiner 1989] Lawrence R. Rabiner. 1989. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In *Readings in Speech Recognition*, pp. 267–296. Alex Waibel and Kai-Fu Lee (eds). Morgan Kaufmann, 1990.

[Rayner & Carter 1996] Manny Rayner and David Carter. 1996. "Fast Parsing Using Pruning and Grammar Specialization". In *Procs. 34th Annual Meeting of the Association for Computational Linguistics*, pp. 223–230, ACL, 1996.

[Samuelsson 1997] Christer Samuelsson. 1997. "Extending N-gram Tagging to Word Graphs". In *Procs. 2nd International Conference on Recent Advances in Natural Language Processing*, Tzigov Chark, Bulgaria.

PARSING BY SUCCESSIVE APPROXIMATION

Helmut Schmid
IMS-CL, University of Stuttgart
Azenbergstr. 12, D-70174 Stuttgart, Germany
Email: schmid@ims.uni-stuttgart.de

Abstract

It is proposed to parse feature structure-based grammars in several steps. Each step is aimed to eliminate as many invalid analyses as possible as efficiently as possible. To this end the set of feature constraints is divided into three subsets, a set of context-free constraints, a set of filtering constraints and a set of structure-building constraints, which are solved in that order. The best processing strategy differs: Context-free constraints are solved efficiently with one of the well-known algorithms for context-free parsing. Filtering constraints can be solved using unification algorithms for non-disjunctive feature structures whereas structure-building constraints require special techniques to represent feature structures with embedded disjunctions efficiently. A compilation method and an efficient processing strategy for filtering constraints are presented.

1 Introduction

The efficiency of context-free parsing is well known [Younger, 1967]. Since many featured structure-based grammars either have a context-free backbone or can be transformed into a grammar with a context-free backbone, it is possible to take advantage of the efficiency of context free parsing if the parser proceeds in two steps: First a context-free parser builds the context-free part of the syntactic analysis which is then extended by the calculation of feature structures. Maxwell and Kaplan [Maxwell III and Kaplan, 1994] experimented with variants of this strategy in their LFG parser. One result of their experiments was that their grammar is processed more efficiently by their parser if the rather broad context-free categories chosen by the grammar writers are replaced by more specific categories. To this end some relevant features have been compiled manually into the context-free grammar. Of course one would prefer to have a compiler perform this task automatically. This would enable the grammar writer to use the categories which he considers appropriate, without loosing efficiency. How this can be done is shown in section 3.

Often it is useful to split the second part of processing, the evaluation of feature constraints, into two steps as well. Feature constraints which are likely to eliminate analyses (filtering constraints) are evaluated first whereas the evaluation of other constraints (structure-building constraints) which mainly serve to build some (e.g. semantic) representation is delayed. The ALEP system (Advanced Language Engineering Platform [Simpkins, 1994]) allows the user to explicitly specify features whose constraints are to be delayed. Kasper and Krieger [Kasper and Krieger, 1996] present a similar idea for HPSG parsing.

Separating filtering constraints and structure-building constraints has two advantages: Since many analyses are eliminated by the filtering constraints, the parser does not waste time on the – usually costly – evaluation of structure-building constraints for analyses which will fail anyway. As another advantage, it is possible to choose the most efficient processing strategy for each of the two constraint types independently.

Structure-building features tend to reflect the syntactic structure of a constituent. The semantic feature of a VP node e.g. would encode the attachment side of an embedded PP if there is ambiguity. In order to avoid such local ambiguities multiplying out in the semantic representation, it is necessary to have means to represent ambiguity locally; i.e. the feature structure representation must allow for embedded disjunctions. A representation in disjunctive normal form (i.e. as a set of alternative non-disjunctive feature structures) would be inefficient because the number of feature structures would grow too fast as local ambiguities multiply out. More efficient algorithms for processing disjunctive feature constraints have been presented e.g. in [Kasper, 1987], [Dörre and Eisele, 1990], [Maxwell III and Kaplan, 1996], and [Emele, 1991].

Filtering constraints, on the other hand, can be processed with standard unification algorithms and a disjunctive normal form representation for feature structures if the feature values restricted by these constraints have limited depth and therefore limited compexity. The SUBCAT and SLASH features in HPSG e.g. have this property whereas e.g. the SUBJ and OBJ features in LFG do not. Even if the depth of a feature value is unbounded it is still possible to limit the complexity of the feature structures artificially by pruning feature structures below some level of embedding. Little of the restrictive power of the constraints is lost thereby, since constraints seldom refer to deeply embedded information. However, in order to ensure the correctness of the final result, the filtering constraints have to be evaluated again in the next step together with the structure-building constraints.

In this article a parser is presented which implements the first two steps of the parsing strategy outlined above. Currently it is assumed that the grammars do not contain structure-building constraints which would require the third processing step (see also section 4.3). Section 2 provides an overview of the grammar formalism used by this parser. Section 3 describes the compilation of the grammar. Details of the parsing strategy and its current limitations are given in section 4. Section 5 presents results from experiments with the parser and section 6 closes with a summary.

2 The Grammar Formalism

The parser employs a rule-based grammar formalism. Each grammar rule has a context-free backbone. One of the daughter nodes in a rule is marked as the head with a preceding backquote. Trace nodes are marked with an asterisk after the category name. At least one daughter node has to be non-empty because rules which generate empty strings are not allowed. Associated to each node in a rule is a set of feature constraint equations which restrict the values of its features. Variables are used to express feature unification: Two features are unified by assigning the value of the same variable to both of them (e.g. f1 = v; f2 = v;). Feature structures are totally well-typed, i.e. they are typed and each feature which is appropriate for some type is present and has a value of an appropriate type. Equality is interpreted extensionally, i.e. two feature structures are considered equal if they have the same type and all of their feature values are equal. Feature structures have to be acyclic. Type hierarchies are not supported currently.

Two predefined feature types and three classes of user-defined types are available to the grammar writer. Features of the predefined type STRING accept any character string as value. Features of the predefined type FS_LIST take a list of feature structures of the class *category* (see below) as value. The user defines his own feature types of the class *enumeration type* by listing the corresponding set of possible values which have to be atomic. Another class of user-defined feature types are the *structured types* which are defined by listing the set of attributes appropriate for this type with the types of their values. *Categories* are the last class of user-defined feature types. Their definition is analogous to that of a structured type. Each node of category X has an associated feature structure of type X.

To simplify the grammar writer's task, the grammar formalism supports templates, default inheritance between the mother node and the head daughter of a rule (the value of a feature of the head daughter of a rule is inherited from the mother node if it is undefined otherwise and if the feature structure of the mother node contains a feature with the same name and type – and vice versa), automatic handling of the two features *Phon* and *HeadLex* (lexical head), and special variable types called "restrictor" types which define a subset of features which are to be unified when two *category* feature structures are (partially) unified by assigning the same variable to both of them. The last feature is needed e.g. to exclude the *Phon* feature from unification when the feature structure of a trace node is unified with the feature structure of a filler node which has been threaded through the tree.

The grammar formalism allows disjunctive value specifications in the case of features of an *enumeration type*[1]. A simple toy grammar written in this formalism is shown in the appendix.

3 Compilation

A compiler transforms the plain text representation of the grammar into a form which is appropriate for the parser and provides error reports. Compilation aims to minimize the computations required during parsing.

[1] Using a bit-vector representation, such disjunctions are easy to store and process efficiently.

The compiler expands templates, adds constraint equations for automatic features and for inherited features, flattens feature structures by replacing structured features with a set of new features corresponding to the subfeatures of the structured feature, and infers in some cases additional constraints. E.g. while compiling the rule[2]

```
VP {Subcat=[*];Subcat=r;}  ->  'V {Subcat=[ NP{}=np | r ];}  NP {}=np;
```

the following constraint equations are obtained[3] (among others):

```
r = 0.VP.Subcat              x = 0.VP.Subcat.cdr(1)
r = 1.VP.Subcat.cdr(1)       x = []        ...
```

From these constraints the compiler infers the additional constraint:

```
x = 1.VP.Subcat.cdr(2)
```

These inferences are necessary for the constraint evaluation algorithm presented in section 4.1.

Finally, the compiler replaces unified variables with a single variable, merges equations of the form *x=constant1; x=constant2* into a new equation *x=constant3*, eliminates redundant equations and generates *fixed assignments* for equations with a feature path expression on the right hand side if the variable on the left hand side is unified with an unambiguous constant in some other equation. This is e.g. the case for the third equation and the inferred equation above. The fixed assignments derived from these equations are:

```
0.VP.Subcat.cdr(1) := []        1.VP.Subcat.cdr(2) := []
```

The three equations involved are removed at this point. For each of the remaining equations with a path expression on the right hand side, the compiler generates a *variable assignment*:

```
0.VP.Subcat := r        1.VP.Subcat.cdr(1) := r
```

Equations with the same variable on the left hand side are then grouped together:

```
r:  r = 0.VP.Subcat    r = 1.VP.Subcat.cdr(1)
```

The variables representing these groups are sorted so that variables which depend on the values of other variables will follow these other variables[4]. This ordering is required by the constraint evaluation algorithm presented in section 4.1.

3.1 Generation of Context-Free Rules

The compiler supports compilation of feature constraints into the context-free backbone of the grammar in the case of features of the class *enumeration type*. Features of other types cannot be compiled because the number of possible values is infinite. The user has to specify which features are to be *incorporated* – i.e. compiled – for each category, and the compiler automatically generates all valid context-free rules with the refined categories.

The following algorithm is used for the generation of the context-free rules: First the compiler orders the incorporated features of all nodes of a given grammar rule. A sequence $f_1, f_2, \ldots, f_n$ is obtained. Then the set of permitted values for the first feature f_1 is determined. To this end, the compiler checks whether there is a fixed assignment for this feature. If one exists, the corresponding value is the only permitted value. Otherwise, the compiler checks whether there are two constraint equations of the form $v = f_1$ and $v = (c_1; c_2; \ldots; c_m)$ where $(c_1; c_2; \ldots; c_m)$ is a disjunction of constant values. In this case the set of permitted values is $\{c_1, c_2, \ldots, c_m\}$. Otherwise all values appropriate for feature f_1 are permitted features. The compiler chooses one of the permitted values and switches to the next feature.

While assigning a value to feature f_i, the compiler first checks whether f_i is unified with some feature f_k where $k < i$. This is the case if there are two equations $y = f_i$ and $y = f_k$. If there is such a feature f_k, which already got a value since it has a smaller index, then its value is assigned to feature f_i. Otherwise the set of permitted values is computed as described above and one value is selected. After the value of the last

[2]The notation `NP{}=np` means "*unify the feature structure of the node* NP *with the feature structure denoted by the variable* np *according to the definition of the restrictor type of the variable* np," i.e. unify the subset of features listed in the restrictor definition. The list notation is similar to that in Prolog, but an asterisk rather than an underscore is used to mark dummy arguments.

[3]The number in front of a path expression refers to the position of the node in the rule. The expression *cdr(1)* refers to the rest list at position 1 of a list, i.e. the list minus its first element.

[4]Dependencies arise when a feature value of type STRING is defined as the concatenation of the values of two other STRING features. The value of the *Phon* feature e.g. is defined in this way.

feature has been fixed, the corresponding context-free rule is output. The other context-free rules are obtained by backtracking.

Assuming that the feature *Number* is to be incorporated into the categories *NP*, *DT*, and *N*, the parser will generate in case of the rule

```
NP {Number=n;} -> DT {Number=n;}  'N {Number=n;};
```

the following two context-free rules:

```
NP_sg -> DT_sg N_sg
NP_pl -> DT_pl N_pl
```

3.2 Compression of the Lexicon

For a parser to be able to process arbitrary text it is essential to have a large lexicon with broad coverage. In order to reduce the space requirements of such a large lexicon, the compiler checks for redundancies. Most information is stored in the form of linked lists and if two lists are identical from some position up to the end, the common tail of the lists is stored only once. Also if two list elements (not necessarily of the same list) are identical, only one copy is stored. With this technique it was possible to compress a lexicon with 300,000 entries to about 18 MBytes, which is about 63 bytes per entry.

4 Parsing

The parser proper consists of two components. The first component is a context-free parser which generates a parse forest, i.e. a compact representation of a set of parse trees which stores common parts of the parse trees only once. The BCKY parser developed by Andreas Eisele[5] is used for this purpose. It is a fast bit-vector implementation of the Cocke-Kasami-Younger algorithm. The second component of the parser reads the context-free parse forest and computes the feature structures in several steps. In each step the parse forest is traversed and a new parse forest with more informative feature structures is generated. Parsing is finished when the feature structures do not change anymore. The first step is the most expensive one computationally since most analyses are typically eliminated in this step. The goal is therefore to make the first step as efficient as possible, rather than minimizing the number of steps.

The recomputation of the parse forest proceeds bottom-up and top-down in turn. During bottom-up processing, the parser first computes the feature structures of terminal nodes by evaluating the constraints associated with the lexical rules. Since the number of lexical rules for a terminal node can be larger than one, there may be more than one resulting feature structure. The new nodes with their feature structures are inserted into a new chart. If a node with the same category and feature structure already exists in the new chart, the parser just adds the new analysis (i.e. the rule number and pointers to the daughter nodes) to the list of analyses at this node. Otherwise, a new node is generated. In both cases, the parser stores a link from the old node to the new one. When the feature structure of a nonterminal node is computed, the parser checks all alternative analyses of this node one after the other. For each analysis it has to try out all combinations of the new nodes which are linked to its daughter nodes (cp. figure 1). For each consistent combination, the parser builds an updated feature structure for the mother node and inserts it into the new chart as in the case of terminal nodes. This method is analogous to the chart parsing techniques used in context-free parsing.

During top-down processing, the parser first copies all top-level nodes which cover the whole input string to the new chart and inserts them into a queue. Then the first node is retrieved from the queue and its daughter nodes are recomputed. The recomputed daughter nodes are inserted into the new chart and, if new, also inserted into the queue for recursive processing. After a traversal of the parse forest is completed, it is checked whether any node has changed. If not, parsing is finished. Otherwise the old chart is cleared, the charts are switched and the next processing step begins.

Why is it necessary to traverse the parse forest more than once? In contrast to formalisms like LFG and HPSG it is not assumed that the feature structure of the root node of a parse tree contains all relevant information[6]. Hence it is necessary to compute the feature structures of all nodes in a parse tree. If there were only one unambiguous parse tree, it would be sufficient to traverse the parse forest once. By means of value sharing it

[5] Andreas Eisele, IMS-CL, University of Stuttgart, andreas@ims.uni-stuttgart.de

[6] It is even assumed that this is not the case (cp. section 4.3).

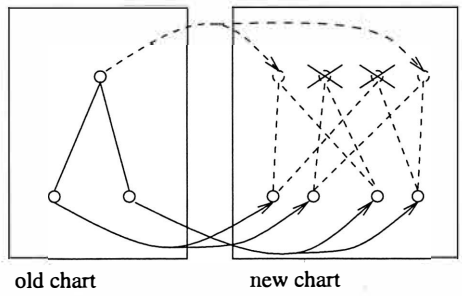


Figure 1: Recomputation of the parse forest

would be possible to update the values of unified features of different nodes in the parse tree synchronously. This is not possible in the case of parse forests, however, because cross-talk would result whenever two analyses have a common node, unless such a shared node is always copied before it is modified which is expensive and to no avail if the analysis later fails. Instead the parse forest is traversed again to update the feature structures of the non-root nodes.

The presented parser has to traverse the parse forest even more often because value sharing between different features of the same feature structure is not used, in order to keep data structures and algorithms as simple as possible. By repeated recomputation, information is properly propagated within the parse forest so that the correct result is obtained. This strategy might seem inefficient, but it turns out that the first two passes which are necessary in any case account for about three quarters of the total processing time and the number of passes seldom exceeds five. As mentioned earlier, it seems more important to speed up the first pass than to reduce the number of passes.

4.1 Constraint Evaluation

The recomputation of feature structures is carried out in four steps. First the input feature structures are specified. During top-down processing, the mother node is a node in the new chart and the daughter nodes are from the old chart. During bottom-up processing it is the mother node which is contained in the old chart and the daughter nodes are from the new chart. The parser then checks whether the fixed assignments are compatible with the input feature structures. If this is the case, the parser computes the values of the variables used in this rule by non-destructively unifying the values specified in the constraint equations for this variable (cp. section 3). The values to be unified are either values of feature paths, or constants[7], or results of string concatenation operations.

Once the values of the variables have been computed, the new feature structures are built by modifying the old feature structures according to the set of fixed assignments and variable assignments. The assignments have been sorted by the compiler so that assignments to less deeply embedded features are carried out first. A lazy copying strategy is used: Before the value of a feature is changed, all levels of the feature structure above this feature are copied unless they have been copied before. After all assignments have been made, the resulting feature structure is inserted into a hash table. If an identical feature structure is already contained in the hash table, a pointer to this feature structure is returned. Otherwise, the new feature structure is inserted. The hashing is done recursively: All embedded feature structures (i.e. elements of feature structure lists), are hashed before an embedding feature structure is hashed. Hashing simplifies the comparison of feature structures to a mere comparison of pointers.

4.2 Optimization

The parsing scheme presented so far has been modified in several ways in order to improve the speed of the parser.

1. The compatibility check for fixed assignments can be done for each node independently of all the other nodes. It is not necessary to repeat it for all combinations of daughter nodes. If a feature structure turns out to be

[7] Such constants are necessarily disjunctive values because otherwise a set of fixed assignments would have been generated.

incompatible, the number of combinations is reduced.

2. The probability that a constraint fails is not identical for all constraints in a rule. Therefore the constraints are sorted so that those constraints which are more likely to fail will be checked first. Inconsistent analyses are therefore eliminated earlier on average. The statistics are collected during parsing.

3. Sometimes it is known in advance that a recomputation of a feature structure will not change its content. This is the case if all input feature structures remained unchanged when they were recomputed the last time. In this case it is sufficient to copy the feature structures to the new chart without recomputing them.

4. Expensive computations are sometimes done repeatedly during parsing, e.g. feature structure unifications. In order to avoid this redundancy, the parser stores each unification operation with pointers to the argument feature structures and the resulting feature structure in a hash table. Before a unification of two feature structures is carried out it is checked whether the result is already in the hash table. Other expensive operations like string concatenations are stored as well.

5. The parser generates a large number of data structures dynamically. In order to avoid the overhead associated with memory allocation calls to the operating system, the parser uses its own simple memory management system which allocates memory from the operating system in large chunks and supplies it to other functions in smaller chunks as needed. Once a sentence has been parsed the allocated memory is freed in one step.

The parser and the compiler have been implemented in the C programming language.

4.3 Limitations

Only the first two processing steps discussed in section 1 – context-free parsing and processing of filtering constraints – have been implemented in the parser so far. In order to be able to build a semantic representation, it would be necessary to add another step which processes structure-building constraints efficiently. The algorithm presented in [Maxwell III and Kaplan, 1996] could be used for this purpose. An even better alternative is Dörre's algorithm [Dörre, 1997] which has polynomial complexity but only works if the constraints never fail. If alternative parse trees are scored after parsing, e.g. with a probabilistic model, semantic construction could also be confined to the best analyses.

The presented parsing method cannot immediately be used to process other grammar formalisms like LFG or HPSG. LFG has no feature typing which is essential for the compilation of the context-free grammar. A separation of filtering constraints and structure-building constraints is difficult in LFG because the SUBJ and OBJ features are used to check subcategorization and to build a simple semantic representation at the same time. The pruning strategy outlined in section 1 might help, but an additional module for the processing of structure-building constraints would still be needed. Of course, other modifications would also be necessary.

The main problem when parsing HPSG with the presented method is to obtain a rule-based grammar from the principle-based representation. Apart from this it would be necessary to emulate the type hierarchy with features. The head features could be partially compiled into the context-free grammar. It is not necessary to compute the DAUGHTERS feature because the tree structure is already represented in the chart. The computation of the features which store the semantic information would have to be done by an additional module.

5 Experimental Results

An English grammar with 290 phrase structure rules[8] has been written for the parser. A lexicon of about 300,000 entries with subcategorization information was extracted from the COMLEX lexical database [Grishman et al., 1994]. The parser has been used to parse 30,000 sentences from the Penn Treebank corpus [Marcus et al., 1993]. Missing lexical entries were automatically generated from the part-of-speech tags in the tagged version of the corpus. However, the part-of-speech tags were not used for parsing itself. Quotation marks were ignored during parsing. More than 7 words per second were parsed on average with a Sun Ultra-2

[8] About 90 rules only deal with coordination, quotation and punctuation.

workstation. Three times the parser stopped prematurely due to memory exhaustion. The calculation of the feature structures was the most time-consuming part of parsing.

For 80 percent of the sentences the parser produced at least one analysis. For 54 percent of the sentences there was at least one analysis which was compatible with the Penn Treebank analysis. An analysis was considered compatible if there were no crossing brackets. However, analyses without crossing brackets are not necessarily acceptable analyses. 100 sentences have been parsed and inspected manually to estimate how often there was an acceptable analysis. For 57 of these sentences the parser had produced a Treebank-compatible analysis, but for only 48 an acceptable one. Interpolating these results, the portion of sentences with an acceptable analysis is probably around 45 percent in the larger corpus.

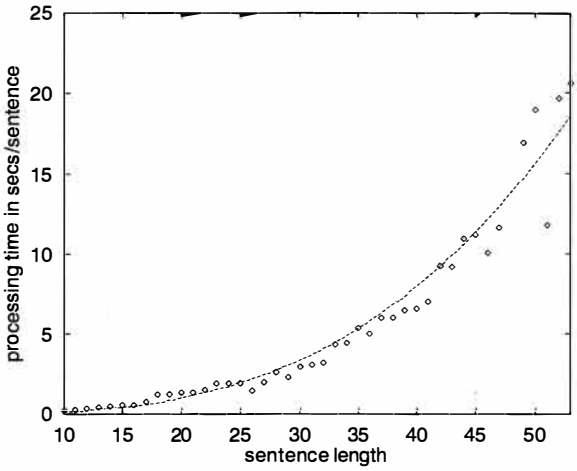


Figure 2: Empirical parsing complexity

Figure 2 shows the empirical parsing complexity which is close to n^3 (the dashed line in the diagram) where n is the sentence length[9].

strategy	25 sentences	1 complex sent.
all optimizations	65.9	180
no hashing of unifications	67.4	193
no hashing of string concatenations	79.3	244
recomputing always	67.3	236

Table 1: Parsing times for 25 randomly selected sentences and a single complex sentence

Another experiment was carried out to check the influence of some of the optimization strategies described in section 4.2 on parsing time. A randomly selected set of 25 sentences was parsed with different variants of the parser in the first part of the experiment. In the second part a single complex sentence was parsed. In each variant of the parser one optimization was switched off. Table 1 shows the results. Hashing of unifications only showed minor effects on parsing speed. Hashing of string concatenation operations was more effective. Presumably string concatenation operations are more likely to be repeated than feature structure unifications. Avoiding unnecessary recomputation of feature structures had a bigger influence on the parsing of the complex sentence than on the parsing of the simpler sentences.

The impact of the incorporation of features into the context-free grammar has also been examined. We observed in contrast to Maxwell and Kaplan [Maxwell III and Kaplan, 1994], only a marginal speedup of about 3 percent from feature incorporation. The incorporation of some features let to disastrous results because the parse forest generated by the context-free parser became very big, slowing down both context-free parsing and the calculation of the feature structures. A close relationship between the number of nodes in the context-free parse forest and parsing time has been observed.

[9]There is an outlier at (48, 36.7) which is not shown in the diagram.

The parser was also compared to a state-of-the-art parser, the XLE system developed at Rank Xerox which was available for the experiments. A corpus of 700 words which both parsers have been able to parse completely was used in this experiment. The XLE system parsed this corpus in 110 seconds whereas our parser needed 123 seconds. Of course it is very difficult to compare these figures since the parsers are too different wrt. the grammar formalisms used, the information contained in the analyses, the degree of ambiguity and other criteria.

6 Summary

A parsing strategy has been outlined which splits parsing into three steps: context-free parsing, evaluation of filtering constraints and evaluation of structure-building constraints. A parser has been presented which implements the first two of these steps. A compiler is used to transform grammar descriptions into a form which the parser is able to process efficiently. The compiler automatically refines the context-free backbone of the grammar by compiling a user-defined set of feature constraints into the context-free backbone. An iterative procedure is used to compute feature structures in disjunctive normal form after a context-free parse forest has been built. As long as the feature structures are not used to build representations which encode the structure of constituents, this parsing strategy works very well: Wall Street Journal data has been parsed at a speed of 7 words per second.

References

[Dörre, 1997] Dörre, J. (1997). Efficient construction of underspecified semantics under massive ambiguity. submitted to ACL'97.

[Dörre and Eisele, 1990] Dörre, J. and Eisele, A. (1990). Feature logic with disjunctive unification. In *Proceedings of the 13th International Conference on Computational Linguistics*, pages 100–105, Helsinki, Finland.

[Emele, 1991] Emele, M. (1991). Unification with lazy non-redundant copying. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 323–330, Berkeley.

[Grishman et al., 1994] Grishman, R., Macleod, C., and Meyers, A. (1994). Comlex syntax: Building a computational lexicon. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan.

[Kasper, 1987] Kasper, R. T. (1987). A unification method for disjunctive feature descriptions. In *Proceedings of the 25th Annual Meeting of the ACL*, pages 235–242, Stanford, CA.

[Kasper and Krieger, 1996] Kasper, W. and Krieger, H.-U. (1996). Modularizing codescriptive grammars for efficient parsing. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 628–633, Copenhagen, Denmark.

[Marcus et al., 1993] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[Maxwell III and Kaplan, 1994] Maxwell III, J. T. and Kaplan, R. M. (1994). The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

[Maxwell III and Kaplan, 1996] Maxwell III, J. T. and Kaplan, R. M. (1996). Unification-based parsers that automatically take advantage of context freeness. Draft.

[Schiehlen, 1996] Schiehlen, M. (1996). Semantic construction from parse forests. In *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, Denmark.

[Simpkins, 1994] Simpkins, N. K. (1994). *ALEP-2 User Guide.* CEU, Luxembourg. This document is online available at http://www.anite-systems.lu/alep/doc/index.html.

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

A A Toy Grammar

```
% comments start with a percent sign

%%%%%% declarations %%%%%%%%%%%%%%%

% definition of the automatic
% feature 'Phon'
auto Phon;

% enumeration type features
enum PERSON  {1st,2nd,3rd};
enum NUMBER  {sg,pl};
enum CASE    {nom,acc};
enum VFORM   {fin,inf,bse,prp,pap,pas};
enum BOOLEAN {yes,no};

% definition of a structured feature
struct AGR {
  NUMBER  Number;
  PERSON  Person;
  CASE    Case;
};

% category definitions
category TOP {
};

category COMP {
};

category SBAR {
  BOOLEAN Wh;
};

category S {
  FS_LIST Slash;
};

category VP {
  VFORM   VForm;
  BOOLEAN Aux;
  FS_LIST Subcat;
  FS_LIST Slash;
};

% Definition of the features which
% are to be compiled into the context
%  free grammar.

VP incorporates {VForm, Aux};

category VBAR {
  VFORM   VForm;
  FS_LIST Subcat;
  FS_LIST Slash;
};

VP incorporates {VForm};

category V {
  VFORM   VForm;
  BOOLEAN Aux;
  FS_LIST Subcat;
};

VP incorporates {VForm,Aux};

category NP {
  BOOLEAN Wh;
  AGR     Agr;
};

NP incorporates {Wh};

category N {
  AGR     Agr;
};

category DT {
  BOOLEAN Wh;
  AGR     Agr;
};

DT incorporates {Wh};

category PP {
};

category P {
};

% definition of restrictor types
restrictor+ NP_R(NP) {Phon, Wh, Agr};

% In the next definition, the Phon
% feature is exempted from unification.
restrictor+ NP2_R(NP) {Wh, Agr};
restrictor+ SBAR_R(SBAR) {Phon, Wh};

% variable declarations
BOOLEAN wh;
AGR agr;
NP_R np;
NP2_R np2;
SBAR_R sbar;
FS_LIST r, r2;

%%%%%% grammar rules %%%%%%%%%%%%%%%

TOP {} ->
    'S {Slash=[];};

S {} ->
    NP {Agr.Case=nom;}=np
   'VP {Subcat=[NP{}=np];};
% The subject-NP is unified with the
% single element of the Subcat list.

VP {} ->     % All features of the two
   'VP {}    % VP nodes are unified due
    PP {};   % to feature inheritance.

VP {} ->
   'VBAR {};

VBAR {} ->
   'VBAR {}
    PP   {};

VBAR {Subcat=r;} ->
   'VBAR {Subcat=[NP{}=np|r];}
    NP   {Agr.Case=acc;}=np;
% All features of the VBAR nodes are
% unified by default feature inheritance
```

```
% excepted the Subcat features.

VBAR {Subcat=r;} ->
   'VBAR {Subcat=[SBAR{}=sbar|r];}
    SBAR {}=sbar;

VBAR {Slash=[];} ->
   'V {};

PP {} ->
   'P  {}
    NP {Agr.Case=acc;};

NP {Wh=wh;} ->
    DT {Wh=wh;Agr=agr;}
   'N  {Agr=agr;};

SBAR {Wh=no;} ->
    COMP {}
      'S {Slash=[];};

SBAR {Wh=yes;} ->
    NP {Wh=yes;}=np2
   'S  {Slash=[NP{}=np2];};
% All features of the NP node and the
% element on the Slash list are unified
% excepted the Phon feature.
% See the definition of NP2_R.

VBAR {Subcat=r;Slash=[np|r2];} ->
   'VBAR {Subcat=[NP{}=np|r];Slash=r2;}
    NP*  {Agr.Case=acc;}=np;
% An NP trace is generated. Information
% from the filler node is threaded via
% the Slash feature.


%%%%%% template definitions %%%%%%%%%%

N_sg    : N {Agr.Number=sg;};
PRO     : NP {Agr.Number=sg;
              Agr.Person=3rd;};
NPRO    : PRO {Wh=no;};
WHPRO   : PRO {Wh=yes;};

%%%%%% lexical entries %%%%%%%%%%%%%%%

"the"   : DT {Wh=no;Agr.Person=3rd;};
"a"     : DT {Wh=no;Agr.Number=sg;
                    Agr.Person=3rd;};
"which": DT {Wh=yes;Agr.Person=3rd;};
"man"   : N_sg {};
"pizza": N_sg {};
"restaurant": N_sg {};
"he"    : NPRO {Agr.Case=nom;};
"him"   : NPRO {Agr.Case=acc;};
"it"    : NPRO {};
"what"  : WHPRO {};
"eats"  : V {Subcat=[
              NP{},
              NP{Agr.Number=sg;
                 Agr.Person=3rd;}];};
"at"    : P {};
"that"  : COMP {};
```

Performance Evaluation of Supertagging for Partial Parsing

B. Srinivas
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
srini@linc.cis.upenn.edu

Abstract

In previous work we introduced the idea of supertagging as a means of improving the efficiency of a lexicalized grammar parser. In this paper, we present supertagging in conjunction with a lightweight dependency analyzer as a robust and efficient partial parser. The present work is significant for two reasons. First, we have vastly improved our results; 92% accurate for supertag disambiguation using lexical information, larger training corpus and smoothing techniques. Second, we show how supertagging can be used for partial parsing and provide detailed evaluation results for detecting noun chunks, verb chunks, preposition phrase attachment and a variety of other linguistic constructions. Using supertag representation, we achieve a recall rate of 93.0% and a precision rate of 91.8% for noun chunking, improving on the best known result for noun chunking.

1 Introduction

A number of grammar formalisms such as HPSG [Pollard and Sag, 1987], CCG [Steedman, 1987], Lexicon-Grammars [Gross, 1984], LTAG [Schabes et al., 1988], Link Grammars [Sleator and Temperley, 1991] fall into the class of lexicalized grammar formalisms. Lexicalized grammar formalisms associate increasingly rich and complex descriptions with each lexical item. Typically, a lexical item in these frameworks is associated with more than one description. The task of a parser for such formalisms can be viewed as first selecting the appropriate description for individual words given the context of the input and then combining them to arrive at a description for the entire input.

In [Joshi and Srinivas, 1994], we introduced the idea of supertagging as a means of selecting the appropriate descriptions for each word given the context of a sentence, even before parsing begins, so as to improve the efficiency of a lexicalized grammar parser. In this paper, we present supertagging in conjunction with a lightweight dependency analyzer as a robust and efficient partial parser. We provide detailed evaluation results of using supertag representation for detecting noun chunks, verb chunks, preposition phrase attachment, appositives and parenthetical constructions. Using supertags, we achieve a recall rate of 93.0% and a precision rate of 91.8% for noun chunking, improving on the best known result for noun chunking [Ramshaw and Marcus, 1995]. We also present vastly improved supertag disambiguation results from previously published 68% accurate to 92% accurate, a significant result keeping in mind that the supertags contain richer information than part-of-speech tags.

The outline of this paper is as follows. In Section 2, we present a brief introduction to Lexicalized Tree-Adjoining Grammars. In Section 3, we review the notion of supertags and in Section 4, we discuss the details of the trigram model for disambiguating supertags and the results of evaluation on the Wall Street Journal corpus. In Section 5, we introduce the lightweight dependency analyzer. A detailed evaluation of the supertag and lightweight dependency analyzer system is presented in Section 6. In Section 7, we briefly discuss two new models for supertagging.

2 Lexicalized Tree-Adjoining Grammars

Each elementary tree of Lexicalized Tree-Adjoining Grammar (LTAG)[Joshi, 1985, Schabes et al., 1988] is associated with at least one lexical item called the *anchor* of that tree. All the arguments of the anchor are realized

as substitution or adjunction slots within an elementary tree. Thus an elementary tree serves as a complex description of the anchor and provides a domain of locality over which the anchor specifies syntactic and semantic (predicate-argument) constraints. Elementary trees are of two types: *initial trees* (α trees in Figure 2) that represent non-recursive linguistic structures such as NPs, PPs and *auxiliary trees* (β trees in Figure 2) that represent recursive structures which are adjuncts to basic structure (e.g. relative clauses, sentential adjuncts, adverbials).

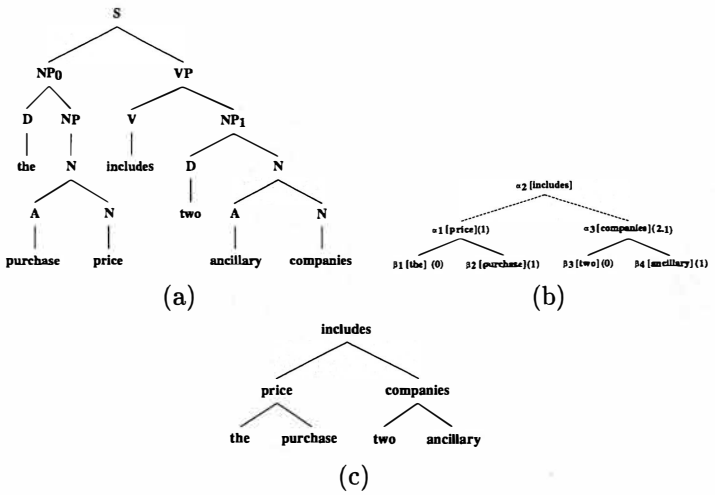


Figure 1: (a): Derived tree (b): Derivation structure (c): Dependency tree for the sentence *the purchase price includes two ancillary companies*

Elementary trees are combined by two operations, *substitution* and *adjunction*. The result of combining the last row of elementary trees of Figure 2 is the *derived tree* (Figure 1(a)). But the more important structure in an LTAG parse is the *derivation tree* (Figure 1(b)) which represents the process of combining the elementary trees to yield a parse. The derivation tree can also be interpreted as a *dependency tree* (Figure 1(c)) with unlabeled arcs between words of the sentence. A wide-coverage English grammar called XTAG has been implemented in the LTAG framework. This grammar has been used to parse sentences from the Wall Street Journal, IBM manual and ATIS domains. A detailed description of this system and its performance results are presented in [Doran et al., 1994].

3 Supertags

The elementary trees of LTAG localize dependencies, including long distance dependencies, by requiring that all and only the dependent elements be present within the same tree. As a result of this localization, a lexical item may be (and almost always is) associated with more than one elementary tree. The example in Figure 2 illustrates the set of elementary trees assigned to each word of the sentence *the purchase price includes two ancillary companies*. We call these elementary trees *supertags*, since they contain more information (such as subcategorization and agreement information) than standard part-of-speech tags. Supertags for recursive and non-recursive constructs are labeled with βs and αs respectively.

The task of a lexicalized grammar parser can be viewed as a two step process. The first step is to select the appropriate supertags for each word of the input and the second step is to combine the selected supertags with substitution and adjunction operations. We call the first step as *Supertagging*. Note that, as in standard part-of-speech disambiguation, supertagging could have been done by a parser. However, just as carrying out part-of-speech disambiguation prior to parsing makes the job of the parser much easier and therefore run faster, supertagging reduces the work of the parser even further.

More interesting is the fact that the result of supertagging is almost a parse in the sense that the parser need 'only' link the individual structures to arrive at a complete parse. We present such a simple linking procedure (*Lightweight Dependency Analyzer*) in Section 5. This method can also be used to parse sentence fragments where it is not possible to combine the disambiguated supertag sequence into a single structure.

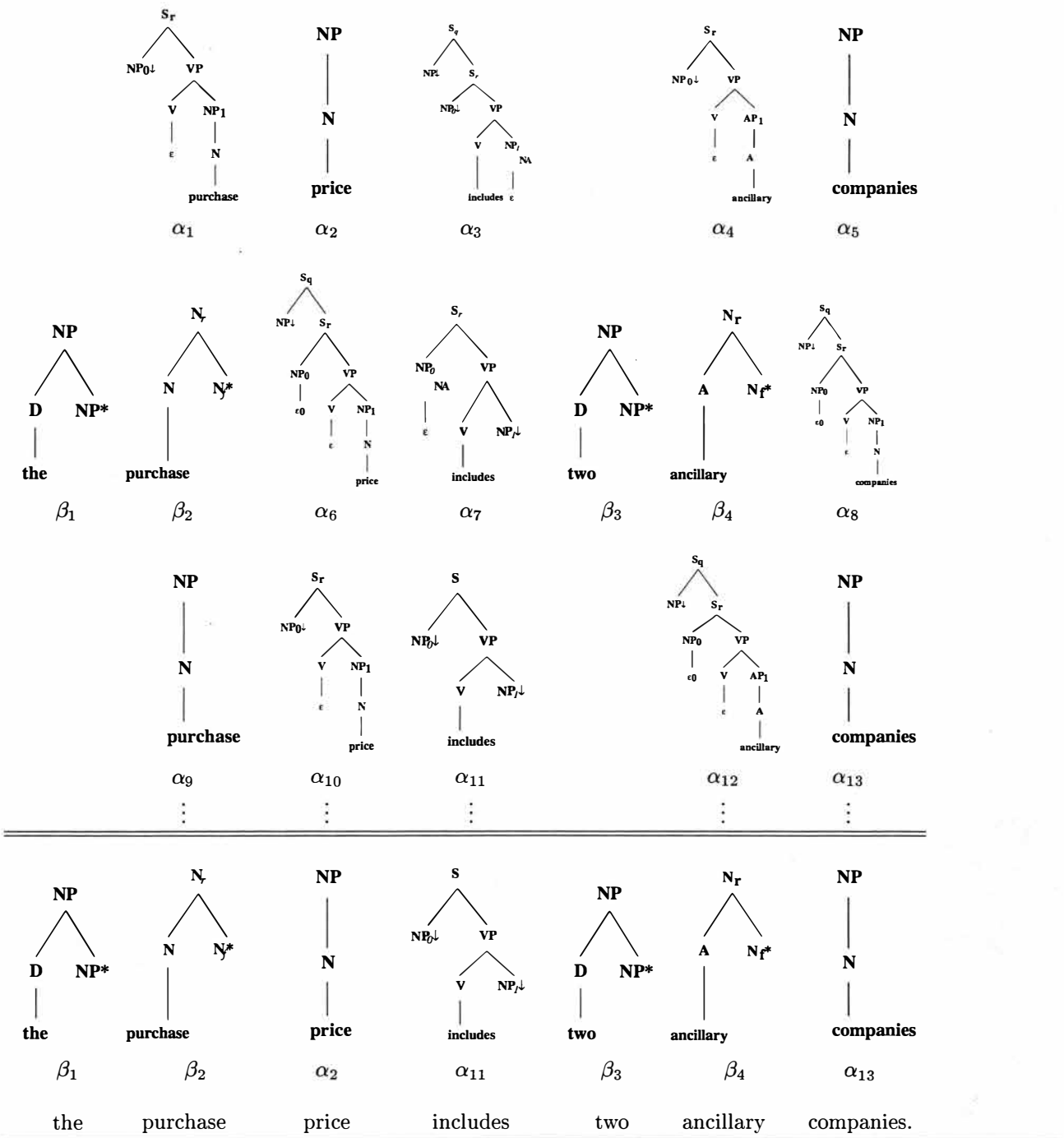


Figure 2: A selection of the supertags associated with each word of the sentence *the purchase price includes two ancillary companies*

4 Trigram Model for Supertagging

The task of supertagging is similar to part-of-speech tagging in that, given a set of tags for each word, the objective is to assign the appropriate tag to each word based on the context of the sentence. Owing to this similarity of supertagging to part-of-speech tagging, we use a trigram model [Church, 1988, Weischedel et al., 1993] to disambiguate supertags. The objective in a trigram model is to assign the most probable supertag sequence for a sentence given the approximation that the supertag for the current word is only influenced by the lexical preference of the current word and the contextual preference based on the supertags of the preceding two words.

Although it is quite evident, owing to the rich information present in supertags, that the dependencies between supertags can easily span beyond the trigram context, one of the goals of this work is to explore the limits of the trigram tagging approach. It appears that a CKY style dynamic programming model that takes advantage of the dependency requirements of each supertag may perform better for supertag disambiguation. However, such an approach is too much like parsing and the objective here is to see how much disambiguation can be done without really parsing.

The lexical and contextual preferences for the trigram model are estimated from a corpus of sentences where the words are tagged with the correct supertag. The estimates for unseen events are arrived at using a smoothing technique. We use Good-Turing discounting technique [Good, 1953] combined with Katz's back-off model for smoothing. We use word features similar to the ones used in [Weischedel et al., 1993], such as capitalization, hyphenation and endings of words, for estimating the unknown word probability. In conjunction with the word features, we exploit the organization of the supertags. The supertags are organized so that transformationally related supertags (indicative, passives, relative clauses, extraction supertags) are grouped into a single "family". Using this notion, if a word w_i in the training material appears with a supertag t_i which belongs to a tree family T, then w_i is associated with all the other members of the tree family T.

Experiments and Results

As reported in the COLING'94 paper [Joshi and Srinivas, 1994], we experimented with a trigram model for supertagging that was trained on (part-of-speech, supertag) pairs collected from the LTAG derivations of 5000 WSJ sentences and tested on 100 WSJ sentences produced a correct supertag for 68% of the words in the test set. We have since significantly improved the performance of the trigram model by making the model lexically sensitive, using a larger training set and incorporating smoothing techniques.

Table 1 shows the performance of the trigram model that was trained on two sets of Wall Street Journal data, 200K words[1] and 1000K words[2] and tested on 50K words[3]. The Treebank parses for the training and test sentences were converted into supertag representation using heuristics specified over parse tree contexts (parent, grandparent, children and sibling information)[4]. A total of 300 different supertags were used in these experiments. Supertag performance is measured as the percentage of words that are correctly supertagged by the model when compared against the supertags for the words in the test corpus.

Size of training set	Training	Size of test set	% Correct
200K	Unigram (Baseline)	50K	75.3%
	Trigram	50K	90.9%
1000K	Unigram (Baseline)	50K	77.2%
	Trigram	50K	92.2%

Table 1: Performance of the supertagger on the WSJ corpus

As mentioned earlier, the supertagger can be used as a front-end to a lexicalized grammar parser so as to prune the search space of the parser even before parsing begins. Alternatively, the dependency information

[1] Sentences in wsj_15 through wsj_18 of Penn Treebank.
[2] Sentences in wsj_00 through wsj_24, except wsj_20 of Penn Treebank.
[3] Sentences in wsj_20 of Penn Treebank.
[4] An example of the heuristics is given in [Srinivas, 1997]

encoded in the supertagger can be used in conjunction with a simple linking procedure as a robust, fast and efficient partial parser. Such an approach can also be used to parse sentence fragments where it is not possible to combine the disambiguated supertag sequence into a single structure.

5 Lightweight Dependency Analyzer

Supertagging associates each word with a unique supertag. To establish the dependency links among the words of the sentence, we exploit the dependency requirements encoded in the supertags. Substitution nodes and foot nodes in supertags serve as slots that must be filled by the arguments of the anchor of the supertag. A substitution slot of a supertag is filled by the complements of the anchor while the foot node of a supertag is filled by a word that is being modified by the supertag. These argument slots have a polarity value reflecting their orientation with respect to the anchor of the supertag. Also associated with a supertag is a list of internal nodes (including the root node) that appear within the supertag. Using the structural information coupled with the argument requirements of a supertag, a simple algorithm such as the one below provides a method for annotating the sentence with dependency links.

Step 1: `For each modifier supertag` s `in the sentence`
`Compute the dependencies for` s
`Mark the words serving as complements as unavailable for step 2.`

Step 2: `For the non-recursive supertags` s `in the sentence`
`Compute the dependencies for` s

Compute Dependencies for s_i of w_i:
`For each slot` d_{ij} `in` s_i `do`
`Connect word` w_i `to the nearest word` w_k `to the left or right of` w_i `depending on the direction of` d_{ij}`, skipping over marked supertags if any, such that` $d_{ij} \in$ `internal_nodes(`s_k`)`

An example illustrating the output from this algorithm is shown in Table 2. The first column lists the word positions in the input, the second column lists the words, the third lists the names of the supertags assigned to each word by a supertagger. The slot requirement of each supertag is shown in column four and the dependency links among the words, computed by the above algorithm, is shown in the fifth column. The $*$ and the . beside a number indicate the type of the dependency relation, $*$ for modifier relation and . for complement relation.

Position	Word	Supertag	Slot req.	Dependency links
0	The	β_1	+NP$*$	2$*$
1	purchase	β_2	+N$*$	2$*$
2	price	α_2	–	
3	includes	α_{11}	–NP. +NP.	2. 6.
4	two	β_3	+NP$*$	6$*$
5	ancillary	β_4	+N$*$	6$*$
6	companies	α_{13}	–	

Table 2: An example sentence with the supertags assigned to each word and dependency links among words

6 Evaluation of Supertag and LDA system

Due to the fact that our system produces a dependency annotated sentence as the output of the parsing process, parsing evaluation metrics that measure the performance of constituent bracketing such as Parseval [Harrison et al., 1991] are unsuitable. Although the supertags contain constituent information, and it is possible to convert a dependency linkage into a constituency based parse, the parseval metric does not have provision for evaluating unrooted parse trees since the output of our system can result in disconnected dependency linkages.

In contrast, we evaluate the performance of our system in terms of its ability to identify certain linguistic structures such as noun groups, verb groups, preposition attachments, appositive and parenthetical constructions. We compare the performance of our system to the performance of other systems specifically designed to identify these structures, when available. The results presented in this section are based on a trigram supertagger trained on 200K words of wsj_15 through wsj_18 and tested on 50K words from wsj_20.

6.1 Text Chunking using Supertags

We have applied the supertag and Lightweight Dependency Analyzer (LDA) system for text chunking. Text chunking, proposed by Abney [Abney, 1991] involves partitioning a sentence into a set of non-overlapping segments. Text chunking serves as a useful and relatively tractable precursor to full parsing. Text chunks primarily consist of non-recursive noun and verb groups. The chunks are assumed to be minimal and hence non-recursive in structure.

Noun Chunking: Supertag based text chunking is performed by the local application of functor-argument information encoded in supertags. Once the head noun of the noun chunk is identified, the prenominal modifiers that are functors of the head noun or functors of the functors of the noun are included in the noun chunk. The head of a noun phrase is identified as the noun with a particular initial (non recursive) supertag (α) in the grammar. Based on that simple algorithm, we can identify noun chunks, for example, shown in (1) and (2).

(1) its increasingly rebellious citizens

(2) two $ 400 million real estate mortgage investment conduits

Ramshaw and Marcus [Ramshaw and Marcus, 1995] present a transformation-based noun chunker that uses a learning scheme presented in [Brill, 1993]. The performance of this noun chunker using rules defined with and without rules incorporating lexical information is shown in Table 3.

System	Training Size	Recall	Precision
R&M	Baseline	81.9%	78.2%
R&M (without lexical information)	200K	90.7%	90.5%
R&M (with lexical information)	200K	92.3%	91.8%
Supertags	Baseline	74.0%	58.4%
Supertags	200K	93.0%	91.8%

Table 3: Performance comparison of the transformation based noun chunker and the supertag based noun chunker

Table 3 also shows the performance of the supertag based noun chunking, trained and tested on the same texts as the transformation-based noun chunker. The supertag-based noun chunker performed better than transformation-based noun chunker with lexical templates Moreover, the supertag-based noun chunking not only identifies the extents of the noun chunks but by the virtue of the functor-argument information, provides internal structure to the noun chunks. This internal structure of the noun chunks could be utilized during subsequent linguistic processing.

Verb Chunking: We also performed an experiment similar to noun chunking to identify verb chunks. We treat a sequence of verbs and verbal modifiers, including auxiliaries, adverbs, modals as constituting a verb group (shown in (3) and (4)). Similar to noun chunking, verb chunking can be performed using local functor-argument information encoded in supertags. However, for verb chunks, the scan is made from left to right starting with a verbal modifier supertag, either an auxiliary verb or an adverbial supertag and including all functors of a verb or a verb modifier. The verb chunker performed at 86.5% recall and 91.4% precision.

(3) would not have been stymied

(4) just beginning to collect

Preposition Phrase Attachment

Supertags distinguish a noun-attached preposition from a verb-attached preposition in a sentence, as illustrated by the two different supertags in (5) and (6). Due to this distinction, the supertagging algorithm can be evaluated based on its ability to select the correct supertag for the prepositions in a text.

(5) sell 500 railcar platforms to/B_vxPnx Trailer Train Co. of/B_nxPnx Chicago

(6) begin delivery of/B_nxPnx goods in/B_vxPnx the first quarter

The task of preposition attachment has been worked on by a number of researchers in the past. Humans perform only at an accuracy of 88% accuracy [Ratnaparkhi et al., 1994] which gives an indication of the complexity of the task. Table 4 presents a comparative evaluation of various approaches in the literature against the supertag based approach, for the preposition attachment task.

System	Accuracy
Ratnaparkhi, Reynar & Roukos	81.6%
Hindle & Rooth	78-80%
Brill & Resnik	81.9%
Collins & Brooks	84.5%
Supertags	81.1%

Table 4: Performance comparison of the supertag based preposition phrase attachment against other approaches to preposition phrase attachment

It must be pointed out that the supertagger makes the preposition attachment decision (choosing between B_vxPnx and B_nxPnx) based on a trigram context. Most often than not the preposition is not within the same trigram window as the noun and the verb due to modifiers of the noun and the verb. However, despite this limitation, the trigram approach performs as well as Ratnaparkhi, Reynar & Roukos [Ratnaparkhi et al., 1994] and Hindle and Rooth [Hindle and Rooth, 1991], and is outperformed only by methods (Brill and Resnik [Brill and Resnik, 1994], Collins and Brooks[Collins and Brook, 1995]) that use four words: the verb, the object noun, the preposition and the complement of the preposition in making the attachment decision.

Other Constructions

In this section, we summarize the performance of the supertagger in identifying constructions such as appositives, parentheticals, relative clauses and coordinating conjunctions.

Appositives: In the XTAG grammar, the appositive construction has been analyzed using a Noun Phrase modifying supertag that is anchored by a comma which takes the appositive Noun Phrase as an argument; for example, the comma in (7) and the second comma in (8) anchor appositive supertags. Thus, a comma in a sentence could either anchor an appositive supertag or a set of coordination supertags, one supertag for each type of coordination. Further, a comma also anchors supertags that mark parentheticals as in (9). The task of the supertagger is to disambiguate among the various supertags and assign the appropriate supertag to the comma, in the appositive context. In Table 5, we present the performance results of the supertagger on such a task. The baseline of assigning the most likely supertag to comma results in zero percent recall.

Parentheticals: Propositional attitude verbs such as *say* and *believe* can not only appear in the canonical word order of subject-verb-complement, but can also appear at other positions in a sentence, as in (8) and (9). Also, the relative order of the subject and the verb can also be reversed as in (8). The XTAG grammar distinguishes such adverbial constructions from the sentence complement construction by assigning different supertags to the verb. A detailed analysis of this construction is presented in [Doran, 1996]. The task of the supertagger is to disambiguate among the sentence complement supertag and the various adverbial supertags for the verb given the context of the sentence. Table 5 presents the performance results of the supertagger on this task. Once again the baseline of selecting the most likely supertag of the verb results in zero percent recall.

(7) Van Pell , 44

(8) " The U.S. underestimated Noriega all along , " *says* Ambler Moss , a former Ambassador to Panama

(9) Mr. Noriega 's relationship to American intelligence agencies became contractual in either 1966 or 1967 , intelligence officials *say* .

Construction	# of occurrences	# identified	# correct	Recall	Precision
Appositive	362	491	306	84.5%	62.3%
Parentheticals	225	200	170	75.5%	85%
Coordination	1886	1750	1329	70.5%	75.9%
Relative Clauses	297	269	116	39.0%	43.2%
Relative Clauses (ignoring valency)	297	269	156	52.5%	58%

Table 5: Performance of the trigram supertagger on Appositive, Parentheticals, Coordination and Relative Clause constructions.

Coordination Conjunctions: In Table 5, we also present the performance of the supertagger in identifying coordination constructions. Coordination conjunctions anchor a supertag that requires two conjuncts, one on either side of the anchor. There is one supertag for every possible pair of conjunct types. We not only have supertags that coordinate like types but we also include supertags that coordinate unlike types amounting to about 30 different supertags for coordination.

Relative Clauses: Due to extended domain of locality of LTAGs, verbs are associated with one relative clause supertag for each of the arguments of the verb. The task of the supertagger in LTAG is not only to identify that the verb is in a relative clause structure but also to identify the valency of the verb and the argument that is being relativized. The performance of the supertagger with and without the valency information being taken into account is present in Table 5.

We are unaware of any other quantitative results on WSJ data for identifying these constructions, for comparative evaluation. It is interesting to note that the performance of the supertagger in identifying appositives and parenthetical construction is better than for coordination conjunctions and relative clause constructions. We believe that this might in part be due to the fact that Appositives and Parentheticals can mostly be disambiguated using relatively local contexts. In contrast, disambiguation of Coordinate constructions and Relative clauses typically require large contexts that are not available for a trigram supertagger.

6.2 Performance of Supertag and LDA system

In this section, we present results from two experiments using the supertagger in conjunction with the LDA system as a dependency parser. In order to evaluate the performance of this system, we need a dependency annotated corpus. However, the only annotated corpora we are aware of, the Penn Treebank [Marcus et al., 1993] and the SUSANNE corpus [Sampson, 1994], annotate sentences with constituency trees. In the interest of time and the need for a dependency annotated corpus, we decided to transform the constituency trees of the two corpora using some rules and heuristics. It must be noted that the resulting corpora is only but an approximation to a manually annotated dependency corpus. However, although the annotation resulting from the transformation does not conform to a standard dependency annotation, it is nevertheless an invaluable resource for performance evaluation of dependency parsers.

For each constituent of a parse a head word is associated using the head percolation table introduced in [Jelinek et al., 1994, Magerman, 1995]. The head percolation table associates with each possible constituent label an ordered list of possible children of the constituent whose head word is passed to the parent. The annotation of a parse tree with head word information proceeds bottom up. At any constituent, the percolation information is consulted to determine which of the constituent's children would pass the head word over to their parent. Once the head words are annotated, the dependency notation is generated by making the head words of non-head constituents to be dependent on the head of the head constituent. In a similar manner to the WSJ conversion process, we converted the subset of the LOB annotation of the SUSANNE corpus into a dependency notation.

In the first experiment, we use the dependency versions of the Penn Treebank annotation of the WSJ corpus and the LOB annotation of the SUSANNE corpus as gold standards. However, in the second experiment, we

use derivation structures of WSJ sentences that were parsed using the XTAG system as the gold standard. The derivation structures serve as dependency structures that are closest in conventions to those assumed by the LDA system.

Experiment 1:

The trigram supertagger trained on 200,000 words of the WSJ corpus was used in conjunction with the LDA to provide a dependency analysis for 2000 sentences of Section 20 of the WSJ corpus. The Penn Treebank parses for these sentences were converted into dependency notation which was used as the gold standard. A dependency link produced by the LDA was regarded to be correct if the words being related by the link were also present in the gold standard. However, to account for the differences in the annotation, the dependency relations among words in a noun group were treated as equivalent to each other. So also, dependency relations among words in a verb group were treated as equivalent to each other. There were 47,333 dependency links in the gold standard and the LDA produced 38,480 dependency links correctly, resulting in a recall score of 82.3%. Also, a total of 41,009 dependency links were produced by the LDA, resulting in a precision score of 93.8%.

We conducted a similar experiment using the SUSANNE corpus. Using the same trigram supertagger trained on the 200,000 words of the WSJ corpus in conjunction with the LDA, we provided a dependency analysis for the sentences in the SUSANNE corpus (125,000 words). The gold standard was created by converting the phrase structure annotations for these sentences into dependency notation using the procedure described above. As in the case of WSJ corpus, to account for the differences in the annotation, the dependency relations among words in a noun group were treated as equivalent to each other. So also, dependency relations among words in a verb group were treated as equivalent to each other. There were a total of 126,493 dependency links in the output of the LDA of which, 112,420 also present in the gold standard, resulting in a precision score of 88.8%. There were a total of 140,280 links in the gold standard, resulting a recall of 80.1%. It is interesting to note that although the trigram model was trained on the WSJ corpus, the performance of the LDA on SUSANNE is comparable to the performance of the LDA on the WSJ corpus.

On analyzing the errors, we discovered that several of the errors were due to the approximate nature of the dependency corpus created by the conversion process. A second source of errors was due to the differences in the notion of dependency produced by the LDA system and that resulting from the conversion of the Treebank. This is largely due to a lack of a standard dependency notation.

Corpus	System	# of dependency links	# produced by LDA	# correct	Recall	Precision
Brown	LDA	140,280	126,493	112,420	80.1%	88.8%
WSJ	LDA	47,333	41,009	38,480	82.3%	93.8%

Table 6: Comparative Evaluation of LDA on Wall Street Journal and Brown Corpus

Experiment 2

In this experiment, we used the correct derivation structure produced by XTAG for 1350 WSJ sentences[5] as the gold standard. These sentences were supertagged using the supertagger trained on 8000 sentences of WSJ and dependency annotated using the LDA system. Table 7 shows the performance of the system. Although, the derivation trees produced by the XTAG system are closest in terms of conventions used in the dependency output of the LDA system; there are a few points of divergences, a major one being the annotation for sentence complement verbs. While in the LDA system, the embedded verb depends on the matrix verb the reverse is the case in an LTAG derivation structure. Also, since the XTAG system, as it is implemented does not permit two auxiliary trees to be adjoined at the same node, certain valid derivation structures are not possible in XTAG. A precise formulation of possible derivation structures in LTAG framework is presented in [Schabes and Shieber, 1992].

Table 8 tabulates the percentage of sentences that have zero, one, two and three dependency link errors. In contrast to evaluation against a skeletally bracketed treebank, evaluation against LTAG derivation trees is much more strict. For example, an LTAG derivation contains detailed annotation in terms of the internal structure of the nominal modifiers in Noun Phrases and verbal modifiers in Verb groups. Also, a derivation structure has

[5] sentences of length less than 16

Training Size (words)	Test Size (words)	Recall	Precision
200,000	12,000	83.6%	83.5%

Table 7: Performance of LDA system compared against the XTAG derivation structures

% sentences with 0 errors	% sentences with with $\leq$1 error	% sentences with with $\leq$2 errors	%sentences with with $\leq$3 errors
35%	60.3%	78%	89.8%

Table 8: The percentage of sentences with zero, one, two and three dependency link errors.

a stricter imposition of the argument-adjunct distinction and distinguishes readings that have similar phrase structure trees such as predicative and equative readings and idiomatic and non-idiom readings of a sentence. Further, the derivation structure is much more closer to semantic interpretation of a sentence than the phrase structure. Hence, the performance figures that are shown in Table 7 and Table 8 are more strict and hence more significant than the crossing bracket, precision and recall figures measured against skeletally bracketed corpora.

7 New Models for Supertag Disambiguation

7.1 Head Trigram Model

As is well known, a trigram model is inadequate at modeling dependencies that appear beyond a three word window. One method of allowing supertags beyond the trigram window to influence the selection of a supertag is to allow it to be "visible" in the current context. This can be achieved in one of two ways. First, the size of the window could be increased to be more than three words. One major drawback of this method is that since an a priori bound is set on the size of the window, no matter how large it is, dependencies that appear beyond that window size cannot be adequately modeled.[6] Also, by increasing the size of the window, the number of parameters to be estimated increases exponentially and sparseness of data becomes a significant problem.

An alternate approach to making head supertags "visible" in the current context is to first identify the head positions and then percolate the supertags at the head positions through the preceding context. Once the positions of the head supertags are identified, the contextual probability in a trigram model is changed so as to be conditioned on the supertags of the two previous head words instead of the two immediately preceding previous supertags. Thus, the contextual probability for the head trigram model is given as:

(10) $\Pr(T_1,T_2,\ldots,T_N) \approx \prod_{i=1}^{N} \Pr(T_i \mid T_{H_{i-2}}, T_{H_{i-1}})$
where
$T_{H_{i-1}}$ and $T_{H_{i-2}}$ are the supertags associated with the two previous head words.

The head positions are percolated according to the following equations.

(11)
Initialize:	(H_{-2},H_{-1})	$= (-2,-1)$
Update :	(H_{i-1},H_i)	$= (H_{i-2},H_{i-1})$ if W_i is not a head word
(head positions at i+1)		$= (H_{i-1},i)$ if W_i is a head word

The head trigram supertagger has two passes. In the first pass, the words are annotated for heads and in the second pass the head information is used to compute the appropriate contextual probability. For the first pass, the head words can be identified using a stochastic trigram tagger that uses two labels and tags words either as head words or as non-head words. The training and test material for the head-word tagger was created from the Penn Treebank parses of WSJ sentences using head percolation rules presented in [Jelinek et al., 1994, Magerman, 1995]. The tagger was trained on 1,000,000 words of WSJ corpus and tested on 47,000 words from Section 20 of WSJ. The tagger assigned the correct head-tag for 91.2% of the words, improving on the baseline of 81.4%. Most of head-word tagging errors were mistakes of tagging modifier nouns as head words in nouns group sequences. Using the head information provided by head-word tagger, the head trigram supertagger, trained

[6] An interesting approach of incorporating variable length n-grams in a part-of-speech tagging model is presented in [T.R. Niesler and P.C. Woodland, 1996].

on the 1,000,000 words of WSJ corpus and tested on 47,000 words, assigned the correct supertag for 87% of words. We suspect that since local constraints are not modeled by the head trigram model, the performance of the trigram model is better than this model. We are working towards integrating the two models to produce a mixed model that takes into account both local and non-local constraints.

7.2 Head Trigram Model with Supertags as Feature Vectors

As previously mentioned, supertags contain subcategorization information and localize filler-gap dependency information within the same structure. As a result, for example, there are not only supertags that realize the complements of a transitive verb in their canonical position, but there are supertags that realize complements in passives, supertags for wh-extraction and relative clause for each of the argument positions. These various supertags are present for each subcategorization frame and are transformationally related to one another in a movement-based approach. The supertags related to one another are said to belong to a "family". The supertags for verbs can be organized in a hierarchy as shown in Figure 3. It must be noted that just as the hierarchy in Figure 3 is organized in terms of the subcategorization information, similar hierarchies can be organized based on transformations such as Relative Clause or Wh-extraction. The supertags for other categories such as nouns, adjectives and prepositions can also be organized in similar hierarchies based on features such as modifiers, complements and predicates.

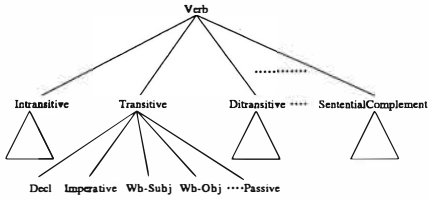


Figure 3: Supertag hierarchy

The representation of supertags as labels, as was done in the previous models, does not exploit the relationship among supertags. Instead, supertags can be viewed as feature vectors where each feature represents a dimension along which the supertags can be clustered. We are currently working on implementing a trigram model with appropriate back-off strategies. A more appropriate model, we believe, for such a distributed encoding of supertags is a model that has the flexibility to select the best back-off strategy as in maximum entropy model. We plan to implement a maximum entropy model for supertagging by the final version of this paper.

8 Conclusions

We have presented the trigram model for supertagging in conjunction with a lightweight dependency analyzer as a robust and efficient partial parser. One of the goals of this work is to explore the limits of the trigram tagging approach and to see how much supertag disambiguation can be done without really parsing. A significant result of the current work is that a trigram model can achieve 92% accuracy for supertagging. Second, we have shown how the disambiguated supertags can be used for partial parsing, detecting noun chunks, verb chunks, preposition phrase attachment, appositives and parenthetical constructions. Using supertags, we have achieved a recall rate of 93.0% and a precision rate of 91.8% for noun chunking, improving on the best known result for noun chunking.

References

[Abney, 1991] Abney, S. (1991). Parsing by chunks. In Berwick, R., Abney, S., and Tenny, C., editors, *Principle-based parsing*. Kluwer Academic Publishers.

[Brill, 1993] Brill, E. (1993). Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, Columbus, Ohio.

[Brill and Resnik, 1994] Brill, E. and Resnik, P. (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of the International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan.

[Church, 1988] Church, K. W. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *2nd Applied Natural Language Processing Conference*, Austin, Texas.

[Collins and Brook, 1995] Collins, M. and Brook, J. (1995). Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, MIT, Cambridge, Boston.

[Doran, 1996] Doran, C. (1996). Punctuation in Quoted Speech. In *Proceedings of the SIGPARSE96*, Santa Cruz, California.

[Doran et al., 1994] Doran, C., Egedi, D., Hockey, B. A., Srinivas, B., and Zaidel, M. (1994). XTAG System - A Wide Coverage Grammar for English. In *Proceedings of the* 17^{th} *International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan.

[Good, 1953] Good, I. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika 40 (3 and 4)*.

[Gross, 1984] Gross, M. (1984). Lexicon-Grammar and the Syntactic Analysis of French. In *Proceedings of the* 10^{th} *International Conference on Computational Linguistics (COLING'84)*, Stanford, California.

[Harrison et al., 1991] Harrison, P., Abney, S., Fleckenger, D., Gdaniec, C., Grishman, R., Hindle, D., Ingria, B., Marcus, M., Santorini, B., and Strzalkowski, T. (1991). Evaluating syntax performance of parser/grammars of English. In *Proceedings of the Workshop on Evaluating Natural Language Processing Systems, ACL*.

[Hindle and Rooth, 1991] Hindle, D. and Rooth, M. (1991). Structural ambiguity and lexical relations. In *29th Meeting of the Association for Computational Linguistics*, Berkeley, CA.

[Jelinek et al., 1994] Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., Ratnaparkhi, A., and Roukos, S. (1994). Decision Tree Parsing using a Hidden Derivation Model. In *Proceedings from the ARPA Workshop on Human Language Technology Workshop*.

[Joshi, 1985] Joshi, A. K. (1985). Tree Adjoining Grammars: How much context Sensitivity is required to provide a reasonable structural description. In Dowty, D., Karttunen, I., and Zwicky, A., editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge, U.K.

[Joshi and Srinivas, 1994] Joshi, A. K. and Srinivas, B. (1994). Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the* 17^{th} *International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan.

[Magerman, 1995] Magerman, D. M. (1995). Statistical Decision-Tree Models for Parsing. In *Proceedings of the* 33^{rd} *Annual Meeting of the Association for Computational Linguistics*.

[Marcus et al., 1993] Marcus, M. M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19.2:313–330.

[Pollard and Sag, 1987] Pollard, C. and Sag, I. A. (1987). *Information-Based Syntax and Semantics. Vol 1: Fundamentals*. CSLI.

[Ramshaw and Marcus, 1995] Ramshaw, L. and Marcus, M. P. (1995). Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, MIT, Cambridge, Boston.

[Ratnaparkhi et al., 1994] Ratnaparkhi, A., Reynar, J., and Roukos, S. (1994). A maximum entropy model for prepositional phrase attachment. In *Proceedings of ARPA Workshop on Human Language Technology*, Plainsboro, NJ.

[Sampson, 1994] Sampson, G. (1994). SUSANNE: a Doomsday book of English Grammar. In *Corpus-based Research into Language*. Rodopi, Amsterdam.

[Schabes et al., 1988] Schabes, Y., Abeillé, A., and Joshi, A. K. (1988). Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary.

[Schabes and Shieber, 1992] Schabes, Y. and Shieber, S. (1992). An Alternative Conception of Tree-Adjoining Derivation. In *Proceedings of the* 20^{th} *Meeting of the Association for Computational Linguistics*.

[Sleator and Temperley, 1991] Sleator, D. and Temperley, D. (1991). Parsing English with a Link Grammar. *Technical report CMU-CS-91-196, Department of Computer Science, Carnegie Mellon University*.

[Srinivas, 1997] Srinivas, B. (1997). Complexity of Lexical Descriptions and its Relevance to Partial Parsing. *PhD Dissertation, University of Pennsylvania*.

[Steedman, 1987] Steedman, M. (1987). Combinatory Grammars and Parasitic Gaps. *Natural Language and Linguistic Theory*, 5:403–439.

[T.R. Niesler and P.C. Woodland, 1996] T.R. Niesler and P.C. Woodland (1996). A variable-length category-based n-gram language model. In *Proceedings, IEEE ICASSP*.

[Weischedel et al., 1993] Weischedel, R., Schwartz, R., Palmucci, J., Meteer, M., and Ramshaw, L. (1993). Comping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19.2:359–382.

AN EARLEY ALGORITHM FOR GENERIC ATTRIBUTE AUGMENTED GRAMMARS AND APPLICATIONS

Frédéric Tendeau*
INRIA-Rocquencourt†, BP 105, 78153 Le Chesnay CEDEX, France
E-mail: Frederic.Tendeau@inria.fr

Abstract

We describe an extension of Earley's algorithm which computes the decoration of a shared forest in a generic domain. Attribute computations are defined by a morphism from leftmost derivations to the generic domain, which leaves the computations independent from (even if guided by) the parsing strategy. The approach is illustrated by the example of a definite clause grammar, seen as CF-grammars decorated by attributes.

1 Introduction

Computational linguistics offers a variety of grammatical formalisms, such as DCGs (Pereira and Warren [13]), LFGs (Kaplan and Bresnan [9]), FUGs (Kay [11]), PATR (Shieber [14]), ... They often rely upon a context-free backbone, on which contextual information is grafted. Such information amounts to a collection of attributes of different types: first-order terms, numerical data, probabilities, feature structures, etc. Hence a parser has to perform context-free and attribute operations, but essentially we think that the algorithmics is not radically different accross linguistic formalisms. Of course the computation details differ but we show that it is possible to abstract sufficiently to see many attribute domains in a unified framework.

In a non-deterministic parser, Lang [12] showed the independence between the strategy (expressed by a non-deterministic pushdown automaton) and the dynamic programming interpretation, which handles non-determinism. The present paper can be seen as a proposal to consider the attribute decoration independently from the two previous aspects.

1. Provided the attribute domain respects some algebraic structure (namely a semiring),

2. given both a strategy for attribute computation, and a dynamic programming interpretation,

the decoration of the parse forest can be computed abstractly, i.e. relying entirely on the algebraic structure.

Our goal is modularity: Earley's algorithm is used to show that a parser can be designed independently from the attribute domain and can be applicable to several linguistic formalisms.

Our extension of Earley's algorithm computes the abstract decoration of a parse forest, and we show that abstract computations are applicable to a variety of interpretations: from probabilities to unification grammar formalisms.

Dynamic programming techniques were introduced by Bellman [3] to reduce the complexity of operational research problems from exponential to polynomial (e.g. the knapsack problem), Cocke, Kasami [10], Younger [17] as well as Earley [8] applied them to non-deterministic context-free parsing. They are applicable on recursive problems for which each sub-problem can be identified by an index and they consist in tabulating the computations, ensuring the solution of each sub-problem is computed only once.

In [15], we presented computations of stochastic information in parallel with the parsing process, taking advantage of dynamic programming techniques. We attempt to apply them to attribute augmented context-free grammars in such a way that the following algorithm appears as a generalization of the stochastic Earley's algorithm in [15]. In this paper, probabilities are defined with respect to leftmost derivations and the so-called *recognition probability* corresponds to the

*presently working at Rank Xerox Research Centre, Grenoble, France.

†this work was partially supported by the grant 95-B030 from the Centre National d'Études des Télécommunications.

classical *inside probability* (Baker [2]). We propose to abstract away from probabilities and to compute a *recognition decoration*, defined with respect to the (abstract) decoration of leftmost derivations, in the same way the recognition probability is in [15].

We described in [16] a formalism for generic decoration, generalizing the probabilities. The sum and the product between positive real numbers are replaced by an abstract sum and product. The sum models non-determinism, or ambiguity, i.e. the fact that a set of leftmost derivations can be associated with a single sentence. The order in which the set of possible leftmost derivations is built is not relevant, neither is the order of evaluation of the associated attributes; therefore, the abstract sum must be commutative.

The product models the construction of a single leftmost derivation from two others (by plugging the second one into the yield of the first one, which is equivalent to parse-tree construction). When considering a sequence of three derivations, the associated semantics must be the same with a bottom-up and a top-down construction, meaning that the abstract product must be associative. The same holds for the sum.

A last condition expected for the abstract operations is distributivity of the product w.r.t. the sum, in order to factorize and expand whenever needed.

As a consequence, the abstract domain must have a semiring structure, which justifies the use of the algebraic power series formalism in [16], where we specify in which conditions and how dynamic programming techniques can be used to compute the abstract decoration of a shared forest, from the definition of an abstract decoration on the grammar.

The theory of algebraic power series, introduced by Schützenberger and Chomsky [6], allows not only to generalize many attribute domains with the semiring structure, but also the context-free structure itself; indeed, the computed values can be probabilities, or first order terms (i.e. classical attributes), but also leftmost derivations or shared forests. Therefore, the decoration of a syntactic construction by attributes can be seen as a morphism from the context-free backbone to the attribute domain: a semiring morphism. So, any parsing algorithm can be extended by a morphism in order to compute attribute decorations.

In many cases however, the semiring structure is too strong: the product might be a partial function rather than an application. For instance, we describe leftmost derivations in an algebraic manner and naturally, composing (multiplying) two derivations is possible only if the second one *can actually* be applied after the first one: consider for example $A \Longrightarrow \alpha \textcircled{\ell} B \Longrightarrow \beta$ (where $\textcircled{\ell}$ denotes the composition of leftmost derivations) then such a product is defined only if $\alpha = wB\gamma$, in which case the composition yields $A \Longrightarrow \alpha \Longrightarrow w\beta\gamma$.

Hence the *partial* semiring of leftmost derivations is defined and attribute computations are defined as a partial semiring morphism from derivations to the attribute domain. As parse trees are equivalent to leftmost derivations, any syntactic algorithm only has to apply the decoration morphism to compute the decoration.

The sequel is organized as follows: section 2 defines the algebra of leftmost derivations, the attribute algebra, and the decoration (which is a semiring morphism) of the leftmost derivations in the attribute domain. Then, after the presentation of shared forests, section 2 introduces the adaptation of Earley items in order to carry attributes. Section 3 describes the algorithm with attribute computations. Section 4 applies our approach to DCGs, which are presented as CFGs decorated in a semiring. Section 5 runs the algorithm on an example of DCG.

2 Analysis material

2.1 Grammar and decoration

- Consider: a context-free grammar $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, the vocabulary $\mathcal{V} = \Sigma \cup \mathcal{N}$, the relation *leftmost derive* is $\underset{\ell}{\Longrightarrow} = \{(xA\omega,\ x\eta\omega) \mid xA\omega \Longrightarrow x\eta\omega\}$. From this relation, a *leftmost derivation* is defined as a sequence s.t. each element derives the following one, more formally: $(\alpha_i)_{1 \leq i \leq l} \in \mathcal{V}^{\mathbb{N}}$ s.t. $l \geq 0$ and if $l > 1$ then $\forall i \geq 1$: $\alpha_i \underset{\ell}{\Longrightarrow} \alpha_{i+1}$ —for $l = 0$, the empty sequence corresponds to the empty derivation denoted by (ϵ). For example, the derivation denoted by $A \Longrightarrow \alpha \Longrightarrow w\beta\gamma$ in the introduction corresponds to the sequence $(A, \alpha, w\beta\gamma)$.

  The set of all leftmost derivations of α into β is denoted by $\alpha \overset{\ell}{\rightsquigarrow} \beta$, and for $n \in \mathbb{N}$, the subset of the latter, restricted to derivations of length n, is $\alpha \overset{n}{\underset{\ell}{\rightsquigarrow}} \beta = \{d \in \alpha \overset{\ell}{\rightsquigarrow} \beta \mid d \text{ is a sequence of length } n+1\}$. Note that for $n = 0$: $\alpha = \beta$ and the derivation is (α).

- Attribute decoration of G in an abstract domain $\mathcal{A}$
  - First, we define a *partial* monoid (this unusual algebraic structure has already been introduced by Arbib and Manes [1]): $(\mathcal{A}, \times_{\mathcal{A}}, 1_{\mathcal{A}})$ is a partial monoid iff $\times_{\mathcal{A}}$ is a partial function over $\mathcal{A} \times \mathcal{A}$, $1_{\mathcal{A}}$ is a neutral element for $\times_{\mathcal{A}}$ ($\forall a \in \mathcal{A}$:

$a \times_{\mathcal{A}} 1_{\mathcal{A}} = 1_{\mathcal{A}} \times_{\mathcal{A}} a = a$), and $\times_{\mathcal{A}}$ is partially associative ($\forall a, b, c \in \mathcal{A}$: $a \times_{\mathcal{A}} b$ and $b \times_{\mathcal{A}} c$ are defined implies $(a \times_{\mathcal{A}} b) \times_{\mathcal{A}} c$ and $a \times_{\mathcal{A}} (b \times_{\mathcal{A}} c)$ are defined and equal). Naturally, if $\times_{\mathcal{A}}$ is a mapping then $(\mathcal{A}, \times_{\mathcal{A}}, 1_{\mathcal{A}})$ is a monoid, and in addition, if $\times_{\mathcal{A}}$ is commutative then so is the monoid.

- Then $(\mathcal{A}, +_{\mathcal{A}}, \times_{\mathcal{A}}, 0_{\mathcal{A}}, 1_{\mathcal{A}})$ is a *partial* semiring iff $(\mathcal{A}, \times_{\mathcal{A}}, 1_{\mathcal{A}})$ is a partial monoid, $(\mathcal{A}, +_{\mathcal{A}}, 0_{\mathcal{A}})$ is a commutative monoid, $0_{\mathcal{A}}$ is absorbent for $\times_{\mathcal{A}}$ (i.e. $\forall c \in \mathcal{A}$: $c \times_{\mathcal{A}} 0_{\mathcal{A}} = 0_{\mathcal{A}} = 0_{\mathcal{A}} \times_{\mathcal{A}} c$), and $\times_{\mathcal{A}}$ is distributive with respect to $+_{\mathcal{A}}$.
- The decoration of G in the partial semiring $(\mathcal{A}, +_{\mathcal{A}}, \times_{\mathcal{A}}, 0_{\mathcal{A}}, 1_{\mathcal{A}})$ is represented by the mapping $\phi_{\mathcal{A}}$: $\mathcal{R} \to \mathcal{A}$.

We want to see the syntactic constructions from an algebraic point of view, in order to present the decoration as a (partial) semiring morphism. Before the operations, the ground set must be defined.

Definition 1. The *set of leftmost derivations* is

$$\Delta_\ell = (\epsilon) \cup \bigcup_{n \in \mathbb{N}} \alpha \overset{n}{\underset{\ell}{\rightsquigarrow}} \beta$$

For short, the composition between leftmost derivations $d' \textcircled{\ell} d''$ is defined if the first element of d'' occurs in the last element of the d', and it consists in continuing the derivation d' by using d''.

Definition 2. The *composition of leftmost derivations* is the function $\textcircled{\ell}$: $\Delta_\ell \times \Delta_\ell \longrightarrow \Delta_\ell$ such that $\forall d', d'' \in \Delta_\ell$:

$$d' \textcircled{\ell} d'' = \begin{cases} d' & \text{if } d'' = (\epsilon) \\ d'' & \text{if } d' = (\epsilon) \\ d \in \alpha \overset{l'+l''}{\underset{\ell}{\rightsquigarrow}} x\gamma\delta & \text{if } d' \in \alpha \overset{l'}{\underset{\ell}{\rightsquigarrow}} x\beta\delta, \text{ and } d'' \in \beta \overset{l''}{\underset{\ell}{\rightsquigarrow}} \gamma \text{ such that } \begin{cases} \forall i \leq l' \colon d_i = d'_i \\ \forall i > l' \colon d_i = x d''_{i-l'} \delta \end{cases} \\ \text{undefined} & \text{otherwise} \end{cases}$$

where d_i denotes the i-th element of the sequence d.

For instance, the example of composition in the introduction can be rewritten: $(A, \alpha) \textcircled{\ell} (B, \beta) = (A, \alpha, w\beta\gamma)$.

In order to deal with non-determinism, sets of leftmost derivations are considered and the function $\textcircled{\ell}$ is naturally extended on $\wp(\Delta_\ell)$ (the power set of Δ_ℓ) in such a way that $\forall e \in \wp(\Delta_\ell)$: $\emptyset \textcircled{\ell} e = \emptyset = e \textcircled{\ell} \emptyset$. Then one easily checks that $(\wp(\Delta_\ell), \cup, \textcircled{\ell}, \emptyset, (\epsilon))$ is a partial semiring.

We consider derivations starting from the axiom, or more generally, from a non-terminal. At parse time, dealing with a $\mathcal{V}^*$ string always comes from the derivation of a non-terminal. Therefore, consider the set $\Delta_\ell^{\mathcal{N}}$ of leftmost derivations of which first element is a non-terminal. Clearly, $(\wp(\Delta_\ell^{\mathcal{N}}), \cup, \textcircled{\ell}, \emptyset, (\epsilon))$ is a partial semiring. This partial semiring is going to be the starting point to compute the abstract decoration.

Definition 3. The function $\phi_{\mathcal{A}}$ is extended as a partial semiring morphism of $\wp(\Delta_\ell^{\mathcal{N}})$ to $\mathcal{A}$, i.e., $\phi_{\mathcal{A}}(\emptyset) = 0_{\mathcal{A}}$, $\phi_{\mathcal{A}}((\epsilon)) = 1_{\mathcal{A}}$, $\phi_{\mathcal{A}}(e_1 \cup e_2) = \phi_{\mathcal{A}}(e_1) +_{\mathcal{A}} \phi_{\mathcal{A}}(e_2)$, and $\phi_{\mathcal{A}}(e_1 \textcircled{\ell} e_2) = \phi_{\mathcal{A}}(e_1) \times_{\mathcal{A}} \phi_{\mathcal{A}}(e_2)$, for all $e_1, e_2 \in \wp(\Delta_\ell^{\mathcal{N}})$.

Grammar rules can be considered as elementary leftmost derivations. If a set s of leftmost derivations is represented with the help of an algebraic expression involving only $\cup$ and $\textcircled{\ell}$as only operations, and with $\mathcal{R}$ as only constants, then it is possible to associate any decoration in $\mathcal{A}$ with s: only by substituting $\cup$ by $+_{\mathcal{A}}$, $\textcircled{\ell}$ by $\times_{\mathcal{A}}$ and elements of $\mathcal{R}$ respectiveley by elements of $\phi_{\mathcal{A}}(\mathcal{R})$, i.e. by a semiring morphism.

In fact, our algorithm computes such a symbolic expression.

2.2 Shared forest

The following notation is used to describe substrings of any $w \in \Sigma^*$: for $0 \leq i \leq j \leq |w|$: $w_{]i,j]}$ stands for the substring of w comprised within the index interval $]i, j]$ (if $i = j$, the empty string is intended).

Billot and Lang [4] represent the set of all parse trees for x with respect to G by mean of a context-free grammar $\boldsymbol{g}_x$. This grammar is called the shared forest of x with respect to G and has a polynomial size (cubic if G is in Chomsky normal form). Classically, the nodes of a parse-tree are instances of G, which justifies the grammatical form of $\boldsymbol{g}_x$. In addition, a suitable choice of the names of these instances leads to the polynomial complexity. Indeed, new non-terminals are introduced: $A^{i,j}$ is such that $A \in \mathcal{N}$ and $A \overset{*}{\Longrightarrow} x_{]i,j]}$. The (finite) set of such non-terminals is $\overline{\mathcal{N}} = \mathcal{N} \times \{0, \cdots, |x|\} \times \{0, \cdots, |x|\}$. A labelling mapping is defined from instances to original symbols of G: consider *lab*: $\overline{\mathcal{N}} \to \mathcal{N}$ such that $\forall A^{i,j} \in \overline{\mathcal{N}}$: $lab(A^{i,j}) = A$; consider $\overline{\mathcal{V}} = \overline{\mathcal{N}} \cup \Sigma$, then *lab* is extended first as the identity mapping on Σ and naturally as a morphism of $\overline{\mathcal{V}}^*$ to $\mathcal{V}^*$. A new (finite) set of rules is defined: $\overline{\mathcal{R}}$ is the biggest subset of $\overline{\mathcal{N}} \times \overline{\mathcal{V}}^*$ such that *lab* can be extended as a morphism of $\overline{\mathcal{R}}$ to $\mathcal{R}$.

Definition 4. The *shared forest* g_x for x with respect to $(\Sigma, \mathcal{N}, \mathcal{R}, S)$ is the context-free grammar obtained after reduction of $(\Sigma, \overline{\mathcal{N}}, \overline{\mathcal{R}}, S^{0,n})$.

Note that $\phi_{\mathcal{A}}(A \overset{}{\underset{\ell}{\rightsquigarrow}} x_{]i,\,j]}) = \phi_{\mathcal{A}}(A^{i,j} \overset{}{\underset{\ell}{\rightsquigarrow}} x_{]i,\,j]})$ (the first derivation uses G, the latter uses g_x).

Exemple 1 Let G be defined by the rules $S \rightarrow SS$ and $S \rightarrow a$. The sentence aaa has two parse-trees which can be represented in a grammatical form in the following way:

$$\begin{array}{lcll} S_1 & \rightarrow & S_2 & S_3 \\ S_2 & \rightarrow & a & \\ S_3 & \rightarrow & S_4 & S_5 \\ S_4 & \rightarrow & a & \\ S_5 & \rightarrow & a & \end{array} \qquad \text{and} \qquad \begin{array}{lcll} S'_1 & \rightarrow & S'_2 & S'_3 \\ S'_2 & \rightarrow & S'_4 & S'_5 \\ S'_3 & \rightarrow & a & \\ S'_4 & \rightarrow & a & \\ S'_5 & \rightarrow & a & \end{array}$$

but with a more suitable choice of the instance names, these can be represented in a single grammar: the shared forest

$$\begin{array}{lcll} S^{0,3} & \rightarrow & S^{0,1} & S^{1,3} \\ S^{0,3} & \rightarrow & S^{0,2} & S^{2,3} \\ S^{0,2} & \rightarrow & S^{0,1} & S^{1,2} \\ S^{1,3} & \rightarrow & S^{1,2} & S^{2,3} \\ S^{0,1} & \rightarrow & a & \\ S^{1,2} & \rightarrow & a & \\ S^{2,3} & \rightarrow & a & \end{array}$$

□

2.3 Earley items and decoration

At parse-time, we rename the non-terminals of the items incrementally in the following manner: when a non-terminal $N \in \mathcal{N}$ is shifted in a completion step of Earley's algorithm, it is renamed by some $N^{i,j} \in \overline{\mathcal{N}}$.

Definition 5. An *Earley item* is an element of $\mathcal{I}(G,x) = \mathcal{N} \times \overline{\mathcal{V}}^* \times \mathcal{V}^* \times \{0,\cdots,|x|\} \times \{0,\cdots,|x|\}$, denoted by $[A \rightarrow \overline{\omega} \bullet \delta,\ i,\ j]$ and satisfying: $A \rightarrow \omega\delta \in \mathcal{R}$, $\overline{\omega} \overset{*}{\underset{\ell}{\Longrightarrow}} x_{]i,\,j]}$, and $\overline{\omega} = \overline{\alpha} B^{p,q} \overline{\beta}$ implies $B^{p,q} \overset{*}{\underset{\ell}{\Longrightarrow}} x_{]p,\,q]}$.

Definition 6. The 4-tuple $< A \rightarrow \overline{\alpha} \bullet \beta, i, j, \mathbf{R} >$ is a *decorated item* iff $I = [A \rightarrow \overline{\alpha} \bullet \beta,\ i,\ j]$ is an Earley item such that $\phi^{\mathbf{r}}_{\mathcal{A}}(I) = \mathbf{R}$, where $\phi^{\mathbf{r}}_{\mathcal{A}}$ is defined below.

The recognition decoration corresponds, for an item $[A \rightarrow \overline{\alpha} \bullet \beta,\ i,\ j]$, to the decoration of the recognized part: $\overline{\alpha}$. In general, it is a vector because the respective decoration of $\overline{\alpha}$ elements might not be composable, i.e., their product in $\mathcal{A}$ may be undefined ($\mathcal{A}$ is a partial semiring). This explains that the yield of $\phi^{\mathbf{r}}_{\mathcal{A}}$ is $\mathcal{A}^*$. Note that if $\mathcal{A}$ is a semiring then the recognition decoration is a scalar: the product ($\times_{\mathcal{A}}$) of all components of the vector.

Definition 7. The *recognition decoration* is the function $\phi^{\mathbf{r}}_{\mathcal{A}}$: $\mathcal{I}(G,x) \rightarrow \mathcal{A}^*$ such that $\forall \overline{\alpha} = w_0 A^{p_1,q_1} w_1 \cdots w_{m-1} A^{p_m,q_m} w_m$:

$$\begin{array}{rcl} \phi^{\mathbf{r}}_{\mathcal{A}}([A \rightarrow \overline{\alpha} \bullet \beta,\ i,\ j]) & = & v \in \mathcal{A}^m \\ \text{such that } \forall k \in \{1,\cdots,m\} : v_k & = & \phi_{\mathcal{A}}(A^{p_k,q_k} \overset{}{\underset{\ell}{\rightsquigarrow}} x_{]p_k,\,q_k]}) \end{array}$$

Note that the recognition decoration is a bottom-up information.

When a right-hand side is entirely recognized, e.g. $[A \rightarrow \overline{\alpha} \bullet,\ i,\ j]$, the whole rule is recognized and the decoration of $A \rightarrow \alpha$ must be composed to the decoration of $\overline{\alpha}$. Hence, the product $\times_{\mathcal{A}}$ is extended on $\mathcal{A} \times \mathcal{A}^*$.

Definition 8. The function $\times_{\mathcal{A}}: \mathcal{A} \times \mathcal{A}^* \rightarrow \mathcal{A}$ is such that $\forall a \in \mathcal{A}, v \in \mathcal{A}^m$:

$$a \times_{\mathcal{A}} v = \begin{cases} a & \text{if } m = 0 \\ (\cdots(a \times_{\mathcal{A}} v_1) \times_{\mathcal{A}} \cdots) \times_{\mathcal{A}} v_m & \text{otherwise} \end{cases}$$

i.e. the product from left to right of a and elements of v.

3 Earley-like algorithm with attribute valuation

Our dynamic programming interpretation consists in destroying the stack of the pushdown automaton: all stack elements (i.e. decorated items) are stored in a set $\mathcal{E}$. The name of these items ensures the correctness of (context-free) computations.

Because of non-determinism, attribute computation cannot be performed immediately: symbols are introduced at each level and they are evaluated after each level is complete.

3.1 Symbolic decoration: recognition of A between i and j

$\phi_{\mathcal{A}}(A^{i,j})$ is introduced because a subtree rooted in $A^{i,j}$ and spanning $x_{]i,j]}$ may be produced more than once. This symbol represents the decoration of all derivations starting from A and ending with $x_{]i,j]}$. The index j is assumed fixed, there is one such symbol for each pair (A, i):

$$\phi_{\mathcal{A}}(A^{i,j}) = \phi_{\mathcal{A}}(A \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{]i,j]}) \tag{1}$$

$$= \sum_{\substack{r=A\longrightarrow w_0A_1w_1\cdots w_{p-1}A_pw_p \\ s.t.\ x_{]i,j]}=w_0y_1w_1\cdots w_{p-1}y_pw_p}} \phi_{\mathcal{A}}(r) \times_{\mathcal{A}} \phi_{\mathcal{A}}(A_1 \overset{*}{\underset{\ell}{\rightsquigarrow}} y_1)\times_{\mathcal{A}}\cdots\times_{\mathcal{A}}\phi_{\mathcal{A}}(A_p \overset{*}{\underset{\ell}{\rightsquigarrow}} y_p) \tag{2}$$

hence $\phi_{\mathcal{A}}(A^{i,j}) =$

$$\sum_{\substack{r=A\longrightarrow w_0A_1w_1\cdots w_{p-1}A_pw_p \\ s.t.\ x_{]i,j]}=w_0y_1w_1\cdots w_{p-1}y_pw_p}} \phi_{\mathcal{A}}(r) \times_{\mathcal{A}} \phi_{\mathcal{A}}({A_1}^{i+|w_0|, i+|w_0y_1|})\times_{\mathcal{A}}\cdots\times_{\mathcal{A}}\phi_{\mathcal{A}}({A_p}^{i+|w_0\cdots w_{p-1}|, j-|w_p|}) \tag{3}$$

The symbols $\phi_{\mathcal{A}}(A^{j,j})$ form an independant equation system. Let us assume that for all A, $\phi_{\mathcal{A}}(A^{j,j}) = \phi_{\mathcal{A}}(A \overset{*}{\underset{\ell}{\rightsquigarrow}} \epsilon)$ is known. Note that it can be statically computed.

Now, consider $i \neq j$ and assume that for all $k_1 \leq k_2 < j$: $\phi_{\mathcal{A}}(A^{k_1,k_2})$ is known. The system is then reduced to the variables $\phi_{\mathcal{A}}(A^{k,j})$ s.t. $k < j$. Moreover, in the equation (3), and in each term of the right-hand side, there is at most one variable $\phi_{\mathcal{A}}(A^{k,j})$ ($i \leq k < j$), hence the system is linear.

Let us solve this system by substitution.

Applying the substitution method to solve the system leads to cyclic (linear) equations: the same variable occurs in both the right-hand side and the left-hand side. Indeed, some cycles appear in equations of the form

$$\phi_{\mathcal{A}}(A^{k,j}) = a\times_{\mathcal{A}}\phi_{\mathcal{A}}(A^{k,j})+_{\mathcal{A}}b \tag{4}$$

with a and b independent from $\phi_{\mathcal{A}}(A^{k,j})$.

Let the Kleene star *: $\mathcal{A} \to \mathcal{A}$ be defined by $\forall a \in \mathcal{A}$:

$$a^* = \lim_{k\to\infty} \sum_{i\in\{0,\cdots,k\}}^{\mathcal{A}} a^i$$

then the solution of (4) is a^*b, if this limit exists in $\mathcal{A}$. More generally, cycles look like

$$\phi_{\mathcal{A}}(A^{k,j}) = a_1\times_{\mathcal{A}}\phi_{\mathcal{A}}(A^{k,j})\times_{\mathcal{A}}a_2 +_{\mathcal{A}} b \tag{5}$$

with a_1, a_2 and b independent from $\phi_{\mathcal{A}}(A^{k,j})$. This amounts to the equation (4) only if $\mathcal{A}$ is commutative (then $a = a_1\times_{\mathcal{A}}a_2$). Otherwise let the limit function be defined by $lim_{\mathcal{A}}$: $\mathcal{A}\times\mathcal{A}\times\mathcal{A}\to\mathcal{A}$, s.t. $\forall a, b, c \in \mathcal{A}$:

$$lim_{\mathcal{A}}(a,\ b,\ c) = \lim_{k\to\infty} \sum_{i\in\{0,\cdots,k\}}^{\mathcal{A}} (a^i\times_{\mathcal{A}}b\times_{\mathcal{A}}c^i)$$

then the solution of 5 is $lim_{\mathcal{A}}(a_1, b, a_2)$, provided that this limit exists.

The definition domains of these functions are such that the limits above exist in $\mathcal{A}$ with respect to an appropriately chosen notion of convergence in $\mathcal{A}$.

Note that in non-commutative semirings, $lim_{\mathcal{A}}$ cannot be defined from a^*. But $a^* = lim_{\mathcal{A}}(a, 1_{\mathcal{A}}, 1_{\mathcal{A}})$ and if $\mathcal{A}$ is commutative, then the limit function amounts to the Kleene star: $lim_{\mathcal{A}}(a_1, b, a_2) = (a_1\times_{\mathcal{A}}a_2)^*\times_{\mathcal{A}}b$.

Therefore, for a given j, the system $\phi_{\mathcal{A}}(A^{i,j})$ has a computable solution assuming

1. $\phi_{\mathcal{A}}(A^{j,j})$ is known;

2. $lim_{\mathcal{A}}$ is well defined and always used in its definition domain; the Kleene star is enough if $\mathcal{A}$ is commutative.

3.2 Algorithm

Vectors of $\mathcal{A}^*$ are denoted between $\langle$ and $\rangle$. A concatenation, denoted by $\diamond$, is defined on $\mathcal{A}^*$. The empty vector is denoted by $\langle\rangle$. The mapping $\overline{\phi_{\mathcal{A}}}$: $\overline{\mathcal{R}} \rightarrow \mathcal{A}$ is a decoration mapping introduced during parsing because it offers a decoration of the shared forest, hence more precise than $\phi_{\mathcal{A}}$: $\mathcal{R} \rightarrow \mathcal{A}$.

$\mathcal{E}$:= $\{<S' \rightarrow \bullet S\$,\ 0,\ 0,\ 1_{\mathcal{A}}>\}$;
`for` j := 0 `to` $|x|$ `loop`
  `loop` **(intra-level)**
    (Prediction)
    `if` $<A \rightarrow \overline{\alpha} \bullet B\delta,\ i,\ j,\ \mathbf{R}> \in \mathcal{E}$ `then add` $<B \rightarrow \bullet\eta,\ j,\ j,\ \langle\rangle>$ `to` $\mathcal{E}$; `end if`;
    (Completion)
    `if` $<A \rightarrow \overline{\alpha}\bullet,\ i,\ j,\ \mathbf{R}'> \in \mathcal{E}$ `and` $<B \rightarrow \overline{\eta} \bullet A\beta,\ k,\ i,\ \mathbf{R}> \in \mathcal{E}$ `then`
      `add` $<B \rightarrow \overline{\eta} A^{i,j} \bullet \beta,\ k,\ j,\ \mathbf{R} \diamond \langle \phi_{\mathcal{A}}(A^{i,j}) \rangle>$ `to` $\mathcal{E}$;
      `output` $(A^{i,j} \rightarrow \overline{\alpha},\ \overline{\phi_{\mathcal{A}}}(A^{i,j} \rightarrow \overline{\alpha}) \times_{\mathcal{A}} \mathbf{R}')$;
    `end if`;
  `until` no more item is added to $\mathcal{E}$;
  `loop` **(Scan)**
    `if` $<A \rightarrow \overline{\alpha} \bullet x_j \beta,\ i,\ j,\ \mathbf{R}> \in \mathcal{E}$ `then add` $<A \rightarrow \overline{\alpha} x_j \bullet \beta,\ i,\ j+1,\ \mathbf{R}>$ `to` $\mathcal{E}$; `end if`;
  `until` no more item is added to $\mathcal{E}$;
`end loop`;

The complexity depends on the semiring under consideration. If $\mathcal{A}$ is so that the cost of each operation is 1 then the time and space complexities are the same as the original syntactic parser, i.e. $\mathcal{O}(n^{L+1})$, where L is the size of the longest right-hand side.

For example, the semiring $(\mathbb{R}^+, +, \times, 0, 1)$ respects such a constraint. If $\phi_{\mathcal{A}}$ is defined so that to describe a probabilistic grammar, then we get a probabilistic parser which computes at each node N of the shared forest, the probability of the whole shared forest dominated by N (see [15]). Another interesting computation is the selection of the best tree, which is achieved by taking the semiring $(\mathbb{R}^+, \max, \times, 0, 1)$ and memorizing only the nodes corresponding to the best probability.

4 Application to DCGs

Naturally, DCGs have a context-free backbone. We just have to find a semiring structure which describes computations with first order terms. This is done in two steps: first, section 4.1 presents Herbrand's domain in a (sup-semi-) lattice framework (which is very close to a semiring) and then, section 4.2 puts first order terms together with the CF information, to form the (partial) semiring of DCGs.

4.1 Herbrand's domain

Consider: a set of variables *Var*, a set of function symbols *Fonc*, a set of (first order) terms $\mathcal{T}$, constructed over *Var* and *Fonc* (by definition $0_{\mathcal{T}}$ is a term called inconsistent term), a set *Subst* of substitutions. The application of a substitution σ to a term t is conventionally denoted by $t\sigma$. By definition 0_{Subst} is a substitution called inconsistent substitution and verifies $\forall t \in \mathcal{T}$: $t0_{Subst} = 0_{\mathcal{T}}$.

Substitutions are used to define a partial pre-order over terms, for which *small* means *precise*.

Definition 9. $\forall t, r \in \mathcal{T}$: $t \sqsubseteq r$ iff there exists a substitution σ verifying $t = r\sigma$.

Note that $0_{\mathcal{T}}$ is the least element of $\mathcal{T}$ w.r.t. $\sqsubseteq$.

To get a partial order, terms which are alphabetical variants are collapsed in a single term. From the (partial) pre-order, an equivalence relation between terms is defined: $\forall s, t \in \mathcal{T}$: $s \equiv t$ iff $s \sqsubseteq t$ and $t \sqsubseteq s$.

Consequently, two quotient terms in $\mathcal{T}/_{\equiv}$ always have a greatest lower bound $\sqcap$ in $\mathcal{T}/_{\equiv}$: $(\mathcal{T}/_{\equiv}, \sqcap, 0_{\mathcal{T}}, \sqsubseteq/_{\equiv})$ is a complete sup-semi lattice.

The unification between two terms is then defined as their greatest lower bound.

Definition 10. For two (quotient) terms s and t, the unification of s and t is $s \sqcap t$.

The unification is extended elementwise to vectors of terms —the only vector with which the empty vector (the vector with no components: $\vec{\epsilon}$) is unifiable is $\vec{\epsilon}$, the result is the substitution 1_{Subst}.

4.2 The partial semiring

Definite clause grammars

A definite clause is seen as a pair composed by a context-free rule $r = A_0 \rightarrow x_0 A_1 x_1 ... A_k x_k$ (in which terminal and non terminal symbols respectively correspond to extensional and intentional predicates) and a mapping $\{0, \cdots, k\} \rightarrow \mathcal{T}^*$ (where $\mathcal{T}^*$ is the set of finite vectors over $\mathcal{T}$) that associates each non-terminal of r with a vector of first order terms. The collection of definite clauses over $\mathcal{R}$ is $\mathcal{C}(\mathcal{R}) = \mathcal{R} \times (\mathcal{T}^*)^{\mathbb{N}}$.

A definite clause grammar is a set of definite clauses.

Domain

The domain $\mathcal{D}$ is the subset of $(\overline{\mathcal{N}} \times (\overline{\mathcal{N}} \cup \Sigma)^*) \times (\mathcal{T}^*)^{\mathbb{N}} \cup \{0_\mathcal{D}, 1_\mathcal{D}\}$ such that for any $d = ((\overline{A}, \overline{\alpha}), \Theta) \in \mathcal{D}$: $(\overline{A}, \overline{\alpha})$ verifies $\overline{A} \underset{\ell}{\overset{*}{\Longrightarrow}} \overline{\alpha}$, and $\mathrm{Dom}(\Theta) = \{0, \cdots, k\}$ iff $\overline{\alpha}$ contains k non-terminals.

The domain of the semiring is the power set of $\mathcal{D}$: $\wp(\mathcal{D})$, denoted by $\mathcal{H}$. The sum $+_\mathcal{H}$ is the union in $\mathcal{H}$. Now we need a product.

Product First, we define $\times_\mathcal{D}$: $\mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$

- $\forall d \in \mathcal{D}$: $1_\mathcal{D} \times_\mathcal{D} d = d = d \times_\mathcal{D} 1_\mathcal{D}$
- $\forall d \in \mathcal{D}$: $0_\mathcal{D} \times_\mathcal{D} d = 0_\mathcal{D} = d \times_\mathcal{D} 0_\mathcal{D}$
- $\forall d_1 = ((\overline{A}, w\overline{A'}\overline{\alpha}), \Theta, \sigma) \in \mathcal{D}$, $d_2 = ((\overline{B}, \overline{\beta}), \Omega, \rho) \in \mathcal{D}$: let $k_\Theta = \max(\mathrm{Dom}(\Theta))$, $k_\Omega = \max(\mathrm{Dom}(\Omega))$. Then

$$d_1 \times_\mathcal{D} d_2 = \begin{cases} \text{undefined} & \text{if } \overline{A'} \neq \overline{B} \text{ (context-free mismatch)} \\ 0_\mathcal{D} & \text{if } \Theta(1) \sqcap \Omega(0) = 0_\mathcal{T} \text{ (unification failure)} \\ ((\overline{A}, w\overline{\beta}\overline{\alpha}), \Theta \cup \Omega) & \text{otherwise (the product succeeds)} \end{cases}$$

where $\Theta \cup \Omega$: $\mathbb{N} \rightarrow \mathcal{T}^*$ s.t. $\mathrm{Dom}(\Theta \cup \Omega) = \{0, \cdots, k_\Theta - 1 + k_\Omega\}$ and

$$\Theta \cup \Omega(v) = \begin{cases} \Theta(1) \sqcap \Omega(0) & \text{if } v = 0 \text{ (unification at grafting point)} \\ \Omega(v) & \text{if } 0 < v \leq k_\Omega \\ \Theta(v - k_\Omega + 1) & \text{if } v > k_\Omega \end{cases}$$

Note that this operation is not commutative, and *partially* associative: take three elements d_1, d_2, d_3 of $\mathcal{H}$ s.t. their respective first component is $A \rightarrow \alpha B \beta C \gamma$, $B \rightarrow \delta$, $C \rightarrow \omega$. Then $(d_1 \times_\mathcal{H} d_2) \times_\mathcal{H} d_3$ is not necessarily undefined whereas $d_2 \times_\mathcal{D} d_3$ is.

The product $\times_\mathcal{D}$ is extended as a function $\mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$ being the natural extension of $\times_\mathcal{D}$ in the usual (distributive) way. Then $1_\mathcal{H} = \{1_\mathcal{D}\}$ and $0_\mathcal{H} = \emptyset$. Finally, $(\mathcal{H}, +_\mathcal{H}, \times_\mathcal{H}, 0_\mathcal{H}, 1_\mathcal{H})$ is a partial semiring, because of $\times_\mathcal{H}$ partial associativity.

The decoration mapping

Consider the CF-grammar G and a definite clause grammar DG.

$$\overline{\phi_\mathcal{H}}(\overline{A} \rightarrow \overline{\alpha}) = \{((\overline{A}, \overline{\alpha}), \Theta) \mid (A \rightarrow \alpha, \Theta) \in DG\}$$

5 Example

Notations are simplified to improve readability: clauses are represented in the usual way, without splitting the context-free backbone from the terms. The vectors of terms are noted between square brackets: e.g., $[c, f(X), g(a, f(Y))]$.

This example shows both how our algorithm works and how operations are performed in the definite clause semiring. The following grammar DG recognizes trivial sentences, verifying that the number of the subject matches the number of the verb. We have $\mathcal{F}onc = \{sg, pl\}$, both functions are constants so $\mathcal{T}$ is reduced to $\mathcal{F}onc$ and $\mathcal{V}ar$.

$$\begin{array}{rcl}
S & \rightarrow & NP[Nb]VP[Nb] \\
NP[Nb_1] & \rightarrow & DetN[Nb_1] \\
NP[Nb_2] & \rightarrow & Pron[Nb_2] \\
VP[Nb_3] & \rightarrow & V[Nb_3] \\
VP[Nb_4] & \rightarrow & V[Nb_4]NP[X] \\
Det & \rightarrow & the \\
N[sg] & \rightarrow & man \mid apple \\
N[pl] & \rightarrow & men \mid apples \\
V[sg] & \rightarrow & eats \mid sings \\
V[pl] & \rightarrow & eat \mid sing \\
Pron[pl] & \rightarrow & you
\end{array}$$

It corresponds to the context-free grammar G:

$$\begin{array}{lll}
S \rightarrow NPVP & NP \rightarrow DetN \mid Pron & VP \rightarrow VNP \mid V \\
Det \rightarrow the & N \rightarrow man \mid men \mid apple \mid apples & V \rightarrow eats \mid sings \mid eat \mid sing
\end{array}$$

with the decoration mapping ϕ:

$$\begin{array}{rcl}
\phi(S \rightarrow NPVP) & = & \{ (S, NP[Z]VP[Z]) \} \\
\phi(NP \rightarrow DetN) & = & \{ (NP[X_2], DetN[X_2]) \} \\
\phi(NP \rightarrow Pron) & = & \{ (NP[Y_2], Pron[Y_2]) \} \\
\phi(VP \rightarrow VNP) & = & \{ (VP[X_3], V[X_3]NP[Y]) \} \\
\phi(VP \rightarrow V) & = & \{ (VP[Z_3], V[Z_3]) \} \\
\phi(Det \rightarrow the) & = & \{ (Det, the) \} \\
\phi(N \rightarrow man) & = & \{ (N[sg], man) \} \\
\phi(N \rightarrow men) & = & \{ (N[pl], men) \} \\
\phi(N \rightarrow apple) & = & \{ (N[sg], apple) \} \\
\phi(N \rightarrow apples) & = & \{ (N[pl], apples) \} \\
\phi(V \rightarrow eats) & = & \{ (V[sg], eats) \} \\
\phi(V \rightarrow eat) & = & \{ (V[pl], eat) \} \\
\phi(V \rightarrow sing) & = & \{ (V[pl], sing) \} \\
\phi(V \rightarrow sings) & = & \{ (V[sg], sings) \}
\end{array}$$

We parse the sentence "the man eats the apples".

The decorated items $<r, i, j, \mathbf{R}>$ computed by the algorithm are given below, ranked with respect to the index (level) j.

Level 0

$$\begin{array}{rll}
<S' \rightarrow \bullet S\$, & 0, 0, & \langle\rangle> \\
<S \rightarrow \bullet NPVP, & 0, 0, & \langle\rangle> \\
<NP \rightarrow \bullet DetN, & 0, 0, & \langle\rangle> \\
<Det \rightarrow \bullet the, & 0, 0, & \langle\rangle>
\end{array}$$

Level 1

$$\begin{array}{rll} <Det{\rightarrow}the\bullet, & 0,1, & \{\,(S,\ Det\ N[Z]\ VP[Z])\,\}> \\ <NP{\rightarrow}Det^{0,1}\bullet N, & 0,1, & \langle\rangle\diamond\langle\phi(Det^{0,1})\rangle> \\ <N{\rightarrow}\bullet man, & 1,1, & \langle\rangle> \\ <N{\rightarrow}\bullet apple, & 1,1, & \langle\rangle> \\ <N{\rightarrow}\bullet men, & 1,1, & \langle\rangle> \\ <N{\rightarrow}\bullet apples, & 1,1, & \langle\rangle> \end{array}$$

Recognition symbol:

$$\begin{array}{rcl} \phi(Det^{0,1}) & = & \overline{\phi}(Det^{0,1}{\rightarrow}the) \\ & = & \{\,(Det^{0,1}, the)\,\} \end{array}$$

Level 2

$$\begin{array}{rll} <N{\rightarrow}man\bullet, & 1,2, & \{\,(S,\ the\ man\ VP[sg])\,\}> \\ <NP{\rightarrow}Det^{0,1}N^{1,2}\bullet, & 0,2, & \langle\phi(Det^{0,1})\rangle\diamond\langle\phi(N^{1,2})\rangle> \\ <S{\rightarrow}NP^{0,2}\bullet VP, & 0,2, & \langle\rangle\diamond\langle\phi(NP^{0,2})\rangle> \\ <VP{\rightarrow}\bullet VNP, & 2,2, & \langle\rangle> \\ <V{\rightarrow}\bullet eats, & 2,2, & \langle\rangle> \\ <V{\rightarrow}\bullet sings, & 2,2, & \langle\rangle> \\ <V{\rightarrow}\bullet eat, & 2,2, & \langle\rangle> \\ <V{\rightarrow}\bullet sing, & 2,2, & \langle\rangle> \end{array}$$

Recognition symbols:

$$\begin{array}{rcl} \phi(N^{1,2}) & = & \overline{\phi}(N^{1,2}{\rightarrow}man) \\ & = & \{\,(N^{1,2}[sg], man)\,\} \\ \phi(NP^{0,2}) & = & \overline{\phi}(NP^{0,2}{\rightarrow}Det^{0,1}N^{1,2}){\times_{\mathcal{H}}}\overline{\phi}(Det^{0,1}){\times_{\mathcal{H}}}\overline{\phi}(N^{1,2}) \\ & = & \{\,(NP^{0,2}[X_1], Det^{0,1}N^{1,2}[X_1])\,\}{\times_{\mathcal{H}}}\{\,(Det^{0,1}, the)\,\}{\times_{\mathcal{H}}}\overline{\phi}(N^{1,2}) \\ & = & \{\,(NP^{0,2}[X_1],\ the\ N^{1,2}[X_1])\,\}{\times_{\mathcal{H}}}\{\,(N^{1,2}[sg], man)\,\} \\ & = & \{\,(NP^{0,2}[sg],\ the\ man)\,\} \end{array}$$

As a consequence, the 2 last items disappear while computing their prediction decoration: the unification of sg and pl fails.

Level 3

$$\begin{array}{rll} <V{\rightarrow}eats\bullet, & 2,3, & \{\,(S,\ the\ man\ eatNP[Y])\,\}> \\ <VP{\rightarrow}V^{2,3}\bullet NP, & 2,3, & \langle\rangle\diamond\langle\phi(V^{2,3})\rangle> \\ <NP{\rightarrow}\bullet Det\ N, & 3,3, & \langle\rangle> \\ <Det{\rightarrow}\bullet the, & 3,3, & \langle\rangle> \end{array}$$

Recognition symbol:

$$\begin{array}{rcl} \phi(V^{2,3}) & = & \overline{\phi}(V^{2,3}{\rightarrow}eats) \\ & = & \{\,(V^{2,3}[sg], eats)\,\} \end{array}$$

Level 4

$$\begin{array}{rll} <Det{\rightarrow}the\bullet, & 3,4, & \{\,(S,\ the\ man\ eats\ the\ N[Y])\,\}> \\ <NP{\rightarrow}Det^{3,4}\bullet N, & 3,4, & \langle\rangle\diamond\langle\phi(Det^{3,4})\rangle> \\ <N{\rightarrow}\bullet man, & 4,4, & \langle\rangle> \\ <N{\rightarrow}\bullet apple, & 4,4, & \langle\rangle> \\ <N{\rightarrow}\bullet men, & 4,4, & \langle\rangle> \\ <N{\rightarrow}\bullet apples, & 4,4, & \langle\rangle> \end{array}$$

Recognition symbol:

$$\begin{array}{rcl} \phi(Det^{3,4}) & = & \overline{\phi}(Det^{3,4}{\rightarrow}the) \\ & = & \{\,(Det^{3,4}, the)\,\} \end{array}$$

Level 5

$$\begin{array}{lll} <N \rightarrow apples\bullet, & 4,5, & \{(S, the\ man\ eats\ the\ apples)\}> \\ <NP \rightarrow Det^{3,4}N^{4,5}\bullet, & 3,5, & \langle\phi(Det^{3,4})\rangle\diamond\langle\phi(N^{4,5})\rangle> \\ <VP \rightarrow V^{2,3}NP^{3,5}\bullet, & 2,5, & \langle\phi(V^{2,3})\rangle\diamond\langle\phi(NP^{3,5})\rangle> \\ <S \rightarrow NP^{0,2}VP^{2,5}\bullet, & 0,5, & \langle\phi(NP^{0,2})\rangle\diamond\langle\phi(VP^{2,5})\rangle> \end{array}$$

Recognition symbols:

$$\begin{array}{rcl} \phi(N^{4,5}) & = & \overline{\phi}(N^{4,5} \rightarrow apples) \\ & = & \{(N^{4,5}[pl], apples)\} \\ \phi(NP^{3,5}) & = & \overline{\phi}(NP^{3,5} \rightarrow Det^{3,4}N^{4,5}) \times_{\mathcal{H}} \overline{\phi}(Det^{3,4}) \times_{\mathcal{H}} \overline{\phi}(N^{4,5}) \\ & = & \{(NP^{3,5}[X_4], Det^{3,4}N^{4,5}[X_4])\} \times_{\mathcal{H}} \{(Det^{3,4}, the)\} \times_{\mathcal{H}} \overline{\phi}(N^{4,5}) \\ & = & \{(NP^{3,5}[X_4],\ the\ N^{4,5}[X_4])\} \times_{\mathcal{H}} \{(N^{4,5}[pl], apples)\} \\ & = & \{(NP^{3,5}[pl],\ the\ apples)\} \\ \phi(VP^{2,5}) & = & \overline{\phi}(VP^{2,5} \rightarrow V^{2,3}NP^{3,5}) \times_{\mathcal{H}} \phi(V^{2,3}) \times_{\mathcal{H}} \phi(NP^{3,5}) \end{array}$$

$$\overline{\phi}(VP^{2,5} \rightarrow V^{2,3}NP^{3,5}) = \{(VP^{2,5}[X_3], V^{2,3}[X_3]NP^{3,5}[Y])\}$$

$$\{(VP^{2,5}[X_3], V^{2,3}[X_3]NP^{3,5}[Y])\} \times_{\mathcal{H}} \{(V^{2,3}[sg], eats)\} = \{(VP^{2,5}[sg], eats\ NP^{3,5}[Y])\}$$

then

$$\begin{array}{rcl} \phi(VP^{2,5}) & = & \{(VP^{2,5}[sg], eats\ NP^{3,5}[Y])\} \times_{\mathcal{H}} \{(NP^{3,5}[pl],\ the\ apples) \\ & = & \{(VP^{2,5}[sg], eats\ the\ apples)\} \\ \phi(S^{0,5})) & = & \overline{\phi}(S^{0,5} \rightarrow NP^{0,2}VP^{2,5}) \times_{\mathcal{H}} \phi(NP^{0,2}) \times_{\mathcal{H}} \phi(VP^{2,5}) \end{array}$$

$$\overline{\phi}(S^{0,5} \rightarrow NP^{0,2}VP^{2,5}) = \{(S^{0,5}, NP^{0,2}[Z]VP^{2,5}[Z])\}$$

$$\{(S^{0,5}, NP^{0,2}[Z]\ VP^{2,5}[Z])\} \times_{\mathcal{H}} \{(NP^{0,2}[sg],\ the\ man)\} = \{(S^{0,5}, the\ man\ VP^{2,5}[sg])\}$$

then

$$\begin{array}{rcl} \phi(S^{0,5}) & = & \{(S^{0,5}, the\ man\ VP^{2,5}[sg])\} \times_{\mathcal{H}} \{(VP^{2,5}[sg], eats\ the\ apples)\} \\ & = & \{(S^{0,5}, the\ man\ eats\ the\ apples)\} \end{array}$$

One remarks that the last substitution is the identity. It can be any substitution because there are no free variables. If the axiom was associated with the variable Z then the substitution would be $(Z \mapsto sg)$.

6 Conclusion

The presentation of first-order terms in a semi-lattice frame suggests the extension to feature structures, especially for hierarchies of types which are distributive lattices (see Carpenter [5]) because they are equivalent to a semiring structure.

In case of cyclic grammars and if limit computations cannot be provided to solve the encountered equation systems, one can think of approximating the solution with the help of abstract interpretation (Cousot and Cousot [7]): instead of $(\mathcal{A}, +_{\mathcal{A}}, \times_{\mathcal{A}}, 0_{\mathcal{A}}, 1_{\mathcal{A}})$, one can consider $(\mathcal{A}^\sharp, +_{\mathcal{A}^\sharp}, \times_{\mathcal{A}^\sharp}, 0_{\mathcal{A}^\sharp}, 1_{\mathcal{A}^\sharp})$ with an abstraction function Θ: $\mathcal{A} \rightarrow \mathcal{A}^\sharp$ which must be a semiring morphism. Naturally, $\mathcal{A}^\sharp$ must be chosen so that limit computations are possible in $\mathcal{A}^\sharp$. Recall that cyclic grammars are usually proscribed by formalisms from computational linguistics, however introducing cycles may be an elegant manner to cope with ill-formed sentences, and then, an approximation just injures the error recovery.

Earley's algorithm is essentially bottom-up (the final information is synthesized) but the search space is reduced by a top-down phase (prediction), and we claim that each syntactic operation can be performed (by morphism) in the decoration domain. In Left Corner, these predictions are statically performed. This suggests that prediction computations could also be performed in the decoration domain, allowing to prune some wrong parses earlier, and possibly statically.

References

[1] M.A. Arbib and E.G. Manes, Partially-additive categories and flow-diagram semantics, *Journal of Algebra* **62** (1980) 203–227.

[2] J.K. Baker, Trainable grammars for speach recognition, *Speech Comm. Papers for 97th Meet. of the Acoustical Society of America, J.J. Wolf and D.H. Klatt Eds.* (1979) 547–550.

[3] R.E. Bellman, *Dynamic Programming* (1957), Princeton Univ. Press.

[4] S. Billot, B. Lang, The structure of shared forest in ambiguous parsing, *Proc. 27th Ann. Meet. of the Assoc. for Comput. Ling.* (1989) .

[5] B. Carpenter, *The Logic of Typed Features Structures* (1992), Cambridge Univ. Press (Cambridge Tracts in Theoretical Computer Science), Cambridge, England.

[6] N. Chomsky, M.P. Schützenberger, The algebraic theory of context-free languages, in: P. Braffort and D. Hirschberg, *Computer Programming and Formal Systems* (1963) 118–161, North-Holland, Amsterdam.

[7] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proc. 4th ACM Symp. on Principles of Programming Languages* (1977) 238–252.

[8] J. Earley, An efficient context-free parsing algorithm, *Comm. Assoc. Comp. Mach.* **13** (1970) 94–102.

[9] R. Kaplan, J. Bresnan, Lexical-functional grammar: a formal system for grammatical representation, *J. Bresnan Ed., The mental Representation of Grammatical Relations, MIT Press, Cambridge, MA* (1982) 173–281.

[10] T. Kasami, *An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages* (1965), Scientific Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

[11] M. Kay, Functional Unification Grammars: a formalism for machine translation, *Proc. 10th Int. Conf. on Comput. Ling., Stanford* (1984) 75–78.

[12] B. Lang, Deterministic techniques for efficient non-deterministic parsers, in J. Loeckx (ed.), *Proc. 2nd Colloquium on Automata, Languages and Programming, Lect. Notes in Comp. Sci.* **14** (1974) 255–269, Springer, Berlin, etc.

[13] F.C.N. Pereira, D.H.D. Warren, Definite Clause Grammars for language analysis – survey of the formalism and comparision with augmented transition networks, *Artificial Intel.* **13:3** (1980) 231–278.

[14] S.M. Shieber, The design of a computer language for linguistic information, *Proc. 10th Int. Conf. on Comput. Ling., Stanford* (1984) 362–366.

[15] F. Tendeau, Stochastic parse-tree recognition by a pushdown automaton, *Proc. 4th International Workshop on Parsing Technology,* Prague / Karlovy Vary (1995) 234–249.

[16] F. Tendeau, Computing abstract decorations of parse forests using dynamic programming and algebraic power series, *To appear in Theor. Comp. Sci.* (1997) .

[17] D.H. Younger, Recognition and parsing of context-free languages in time n^3, *Inform. Contr.* **10** (1967) 189–208.

A CASE STUDY IN OPTIMIZING PARSING SCHEMATA BY DISAMBIGUATION FILTERS

Eelco Visser
Programming Research Group
University of Amsterdam
visser@acm.org

Abstract

Disambiguation methods for context-free grammars enable concise specification of programming languages by ambiguous grammars. A disambiguation filter is a function that selects a subset from a set of parse trees—the possible parse trees for an ambiguous sentence. The framework of filters provides a declarative description of disambiguation methods independent of parsing. Although filters can be implemented straightforwardly as functions that prune the parse forest produced by some generalized parser, this can be too inefficient for practical applications.

In this paper the optimization of parsing schemata, a framework for high-level description of parsing algorithms, by disambiguation filters is considered in order to find efficient parsing algorithms for declaratively specified disambiguation methods. As a case study the optimization of the parsing schema of Earley's parsing algorithm by two filters is investigated. The main result is a technique for generation of efficient LR-like parsers for ambiguous grammars disambiguated by means of priorities.

1 Introduction

The syntax of programming languages is conventionally described by context-free grammars. Although programming languages should be unambiguous, they are often described by ambiguous grammars because these allow a more natural formulation and yield better abstract syntax. For instance, consider the following grammars. The first, ambiguous grammar gives a clearer and more concise description of arithmetic expressions than the second unambiguous one.

```
"a"         -> E
E "+" E     -> E
E "*" E     -> E
"(" E ")"   -> E
```

```
"a"         -> V    T -> E
E "+" T     -> E    V -> T
T "*" V     -> T
"(" E ")"   -> V
```

To obtain an unambiguous specification of a language described by an ambiguous grammar it has to be disambiguated. For example, the first grammar above can be disambiguated by associativity and priority rules that express that `E "*" E -> E` has higher priority than `E "+" E -> E` and that both productions are left associative. In the second grammar these disambiguation rules have been encoded in the grammar itself by means of extra non-terminals.

In Klint and Visser (1994) we have set up a framework for specification and comparison of disambiguation methods. In this framework a disambiguation method is described as a *filter* on sets of parse trees. A disambiguation filter is interpreted by parsing sentences according to the ambiguous context-free grammar with some generalized parsing method, for instance Generalized LR parsing (Tomita, 1985, Rekers, 1992), and then prune the resulting parse forest with the filter. Because this method of specification of disambiguation is independent of parsing, a language definition can be understood without understanding a parsing algorithm and it can be implemented by any generalized parser.

Although filters provide a uniform model for the description of disambiguation, they are too inefficient for several applications because all possible parse trees for a sentence have to be built before the intended ones are selected. (The number of possible parse trees for the first grammar above grows exponentially with the length of strings.) The *optimization problem* for filters is to find an efficient parser for the combination of a context-free grammar and a disambiguation filter. The filter can be used to prevent parse steps that lead to

parse trees that would be removed by the filter after parsing. *Parsing schemata*, introduced by Sikkel (1993, 1994), are high-level descriptions of parsing algorithms that abstract from control- and data-structures and provide a suitable framework for the study of the interaction between filters and parsers.

Since it is not clear how to solve the optimization problem in general, if that is possible at all, an instance of the problem is studied in this paper, i.e., the optimization of the underlying parsing schema of Earley's (1970) parsing algorithm by a filter for disambiguation by priorities. This method, which is the disambiguation method of the formalism SDF (Heering *et al.*, 1989), interprets a priority relation on context-free productions as two consecutive filters. The first selects trees without a priority conflict. The second selects trees that are minimal with respect to a multi-set ordering on trees induced by the priority relation.

The main result of this paper is a parsing schema for parsing with priorities. The schema specifies a complete implementation of parsing modulo priority conflicts and a partial implementation for the multi-set order. The schema can be implemented as an adaptation of any parser generator in the family of LR parser generators. The resulting parsers yield parse trees without priority conflicts.

The method of specifying a disambiguation method by a filter and applying it to optimize the parsing schema of some parsing algorithm appears to be fertile soil for growing new parsing algorithms from old ones.

The rest of the paper is structured as follows. In §2 some preliminary notions are defined. In §3 disambiguation filters are defined. In §4 parsing schemata are informally introduced. In §5 priority rules and the notion of priority conflict are defined and a parsing schema optimized for the priority conflict filter is derived. In §6 the relation between Earley parsing and LR parsing is discussed and it is shown how optimization results can be translated from the former to the latter. Furthermore, the results are extended to SLR(1) parsing. In §7 the multi-set filter induced by a priority declaration is defined and a partial optimization of the Earley schema for this filter is derived. The two optimizations can be combined in a single schema, obtaining an efficient implementation of disambiguation with priorities.

2 Preliminaries

Definition 2.1 (Context-free Grammar) A *context-free grammar* $\mathcal{G}$ is a triple $\langle V_N, V_T, \mathcal{P} \rangle$, where V_N is a finite set of nonterminal symbols, V_T a finite set of terminal symbols, V the set of symbols of $\mathcal{G}$ is $V_N \cup V_T$, and $\mathrm{P}(\mathcal{G}) = \mathcal{P} \subseteq V^* \times V_N$ a finite set of productions. We write $\alpha \to A$ for a production $p = \langle \alpha, A \rangle \in \mathcal{P}$. □

The $\alpha \dashrightarrow A$ notation for productions (instead of the traditional $A \to \alpha$) is a convention of the syntax definition formalism SDF to emphasize the use of productions as mixfix function declarations. The string rewrite relation $\to_{\mathcal{G}}^*$ induced by a context-free grammar is therefore also reversed, from a generation relation to a recognition relation. Repeated application of productions rewrites a string to its syntactic category. The statement $w \to^* A$ means that the string w can be reduced to the symbol A.

Observe that we do not distinguish a start symbol from which sentences are derived. Each nonterminal in V_N generates a set of phrases as is defined in the following definition.

Definition 2.2 (Parse Trees) A context-free grammar $\mathcal{G}$ generates a family of sets of *parse trees* $\mathcal{T}(\mathcal{G}) = (\mathcal{T}(\mathcal{G})(X) \mid X \in V)$, which contains the minimal sets $\mathcal{T}(\mathcal{G})(X)$ such that

$$\frac{X \in V}{X \in \mathcal{T}(\mathcal{G})(X)}$$

$$\frac{A_1 \ldots A_n \to A \in \mathrm{P}(\mathcal{G}),\ t_1 \in \mathcal{T}(\mathcal{G})(A_1),\ \ldots,\ t_n \in \mathcal{T}(\mathcal{G})(A_n)}{[t_1 \ldots t_n \to A] \in \mathcal{T}(\mathcal{G})(A)}$$

We will write t_α for a list $t_1 \ldots t_n$ of trees where α is the list of symbols $X_1 \ldots X_n$ and $t_i \in \mathcal{T}(\mathcal{G})(X_i)$ for $1 \leq i \leq n$. Correspondingly we will denote the set of all lists of trees of type α as $\mathcal{T}(\mathcal{G})(\alpha)$. Using this notation $[t_1 \ldots t_n \to A]$ can be written as $[t_\alpha \dashrightarrow A]$ and the concatenation of two lists of trees t_α and t_β is written as $t_\alpha t_\beta$ and yields a list of trees of type $\alpha\beta$.

The *yield* of a tree is the concatenation of its leaves. The language $L(\mathcal{G})$ defined by a grammar $\mathcal{G}$ is the family of sets of strings $L(\mathcal{G})(A) = \mathrm{yield}(\mathcal{T}(\mathcal{G})(A))$. □

Definition 2.3 (Parsing) A *parser* is a function Π that maps each string $w \in V_T^*$ to a set of parse trees. A parser Π *accepts* a string w if $|\Pi(w)| > 0$. A parser Π is *deterministic* if $|\Pi(w)| \leq 1$ for all strings w. A parser for a context-free grammar $\mathcal{G}$ that accepts exactly the sentences in $L(\mathcal{G})$ is defined by

$$\Pi(\mathcal{G})(w) = \{t \in \mathcal{T}(\mathcal{G})(A) \mid A \in V_N, \text{yield}(t) = w\}$$ □

Example 2.4 As an example consider the ambiguous grammar

```
"a"         -> E
E "+" E     -> E
E "*" E     -> E
"(" E ")"   -> E
```

from the introduction. According to this grammar the string $a + a * a$ has two parses:

$$\begin{aligned}\Pi(\mathcal{G})(a + a * a) = \{&[[[a \to E] + [a \to E] \to E] * [a \to E] \to E] \\ &[[a \to E] + [[a \to E] * [a \to E] \to E] \to E]\}\end{aligned}$$ □

3 Disambiguation Filters

Definition 3.1 (Disambiguation Filter) A *filter* $\mathcal{F}$ for a context-free grammar $\mathcal{G}$ is a function $\mathcal{F} : \wp(\mathcal{T}) \to \wp(\mathcal{T})$ that maps sets of parse trees to sets of parse trees, where $\mathcal{F}(\Phi) \subseteq \Phi$ for any $\Phi \subseteq \mathcal{T}$. The *disambiguation* of a context-free grammar $\mathcal{G}$ by a filter $\mathcal{F}$ is denoted by $\mathcal{G}/\mathcal{F}$. The *language* $L(\mathcal{G}/\mathcal{F})$ generated by $\mathcal{G}/\mathcal{F}$ is the set

$$L(\mathcal{G}/\mathcal{F}) = \{w \in V_T^* \mid \exists \Phi \subseteq \mathcal{T}(\mathcal{G}) : \text{yield}(\Phi) = \{w\} \ \wedge \ \mathcal{F}(\Phi) = \Phi\}$$

The *interpretation* of a string w by $\mathcal{G}/\mathcal{F}$ is the set of trees $\mathcal{F}(\Pi(\mathcal{G})(w))$. A filter $\mathcal{F}_2$ is also applicable to a disambiguated grammar $\mathcal{G}/\mathcal{F}_1$, which is denoted by $(\mathcal{G}/\mathcal{F}_1)/\mathcal{F}_2$ and is equivalent to $\mathcal{G}/(\mathcal{F}_2 \circ \mathcal{F}_1)$. □

Several properties and examples of filters are discussed in Klint and Visser (1994). In §5 and §7 two examples of disambiguation filters will be presented. The optimization problem for disambiguation filters can be formulated as follows.

Definition 3.2 (Optimization by Filter) Given a context-free grammar $\mathcal{G}$ and a filter $\mathcal{F}$, a parser π is an optimization of $\Pi(\mathcal{G})$ if for any string w

$$\mathcal{F}(\Pi(\mathcal{G})(w)) \subseteq \pi(w) \subseteq \Pi(\mathcal{G})(w)$$

We say that π *approximates* $\mathcal{F} \circ \Pi(\mathcal{G})$. π is an optimal approximation if $\pi(w) = \mathcal{F}(\Pi(\mathcal{G})(w))$ for any w. □

4 Parsing Schemata

Parsing schemata (Sikkel, 1993, 1997) abstract from the details of control- and data-structures of full parsing algorithms by only considering the intermediate results of parsing. A parsing system is a deduction system that specifies how from a set of hypotheses (the tokens of a sentence) assertions (the intermediate parser states) can be derived according to a set of deduction rules for some context-free grammar. A parsing schema is a parsing system parameterized with a context-free grammar and a sentence. Below parsing schemata are introduced informally by means of an example. A formal treatment can be found in Sikkel (1993, 1997). A related approach is the deductive parsing method of Shieber *et al.* (1995), where inference rules describing parsing algorithms much like parsing schemata are interpreted as chart parsers in Prolog.

Definition 4.1 defines a parsing schema for Earley's (1970) parsing algorithm. Its specification consists of an implicit definition of the set of hypotheses H, the definition of a set of items $\mathcal{I}$ and the definition of a set of deduction rule schemata. For each string $a_1 \ldots a_n$ the set of hypotheses H is the set containing the items $[a_i, i-1, i]$ for $1 \leq i \leq n$. The set of items $\mathcal{I}$ is the domain of the deduction system, i.e., the items are the subject of deductions. According to this definition, Earley items are of the form $[\alpha \bullet \beta \to A, i, j]$, where $\alpha\beta \to A$

is a production of grammar $\mathcal{G}$. The indices refer to positions in the string $a_1 \dots a_n$. The intention of this definition is that an item $[\alpha \bullet \beta \to A, i, j]$ can be derived if $a_{i+1} \dots a_j \to^*_{\mathcal{G}} \alpha$ and $a_1 \dots a_i A\gamma \to^*_{\mathcal{G}} B$, for some non-terminal B and string of symbols γ. The deduction rules (I) through (C) describe how these items can be derived. Rule (I), the *initialization* rule, specifies that the item $[\bullet\alpha \to A, 0, 0]$ can always be derived. The *predict* rule (P), states that a production $\gamma \to B$ can be predicted at position j, if the item $[\alpha \bullet B\beta \to A, i, j]$ has already been derived. Finally, the rules (S) and (C) finalize the recognition of a predicted and recognized token or nonterminal—witnessed by the second premise—by shifting the $\bullet$ over the predicted symbol.

Definition 4.1 (Earley) Parsing schema for Earley's parsing algorithm (Earley, 1970).

$$\frac{\alpha\beta \to A \in \mathrm{P}(\mathcal{G}),\ 0 \le i \le j}{[\alpha \bullet \beta \to A, i, j] \in \mathcal{I}}$$

$$\frac{}{[\bullet\alpha \to A, 0, 0]} \quad \text{(I)}$$

$$\frac{[\alpha \bullet B\beta \to A, i, j]}{[\bullet\gamma \to B, j, j]} \quad \text{(P)}$$

$$\frac{[\alpha \bullet a\beta \to A, i, j],\ [a, j, j+1]}{[\alpha a \bullet \beta \to A, i, j+1]} \quad \text{(S)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i],\ [\gamma \bullet \to B, i, j]}{[\alpha B \bullet \beta \to A, h, j]} \quad \text{(C)}$$

□

A derivation according to a parsing schema is a sequence $I_0, \dots, I_m$ of items such that for each i $(0 \le i \le m)$ $I_i \in H$ or there is a $J \subseteq \{I_0, \dots, I_{i-1}\}$ such that $J \vdash I_i$ is (the instantiation of) a deduction rule. (Observe that if J is empty this corresponds to the case of using a rule without premises, such as the initialization rule.) A string $w = a_1 \dots a_n$ is in the language of context-free grammar $\mathcal{G}$ if an item $[\alpha\bullet \to A, 0, n]$ is derivable from the hypotheses corresponding to w in the instantiation of the parsing schema in Definition 4.1 with $\mathcal{G}$. An item of the form $[\alpha\bullet \to A, 0, n]$ is called a *final* item and signifies that the entire string is recognized as an A phrase. The predicate $w \vdash^P_{\mathcal{G}} I$ expresses that there is a derivation $I_0, \dots, I_m = I$ of the item I from the hypotheses generated from string w in the instantiation of parsing schema P with grammar $\mathcal{G}$.

The schema in Example 4.1 only defines how strings can be recognized. Since disambiguation filters are defined on sets of trees and not on items, a way to relate items to trees is needed. Definition Definition 4.3 gives an extension of the schema in Definition 4.1 that describes how trees can be built as a result of the deduction steps. First we need a definition of partial parse tree

Definition 4.2 (Partial Parse Tree) A partial parse tree is a tree expression of the form $[t_\alpha \to A]$ where $t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha)$ and such that the tree can be completed to a normal tree by adding a list of trees t_β, i.e., $[t_\alpha t_\beta \to A] \in \mathcal{T}(\mathcal{G})(A)$. (which requires $\alpha\beta \to A \in \mathrm{P}(\mathcal{G})$.) □

The items in the schema have the form $[\alpha \bullet \beta \to A, i, j] \Rightarrow [t_\alpha \to A]$ and express that from position i to position j a phrase of type α has been recognized and the partial parse tree $[t_\alpha \to A]$ has been built as a result. The set of hypotheses H is changed such that token items are annotated with trees, i.e., for each token a_i in the string $[a_i, i-1, i] \Rightarrow a_i \in H$. Note how the shift and complete rules extend partial parse trees.

Definition 4.3 (Earley with Trees) Parsing schema for Earley's algorithm with construction of parse trees.

$$\frac{\alpha\beta \to A \in \mathrm{P}(\mathcal{G}),\ 0 \le i \le j,\ t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha)}{[\alpha \bullet \beta \to A, i, j] \Rightarrow [t_\alpha \to A] \in \mathcal{I}}$$

$$\frac{}{[\bullet\alpha \to A, 0, 0] \Rightarrow [\to A]} \quad \text{(I)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A]}{[\bullet\gamma \to B, i, i] \Rightarrow [\to B]} \quad \text{(P)}$$

$$\frac{[\alpha \bullet a\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [a, i, i+1] \Rightarrow a}{[\alpha a \bullet \beta \to A, h, i+1] \Rightarrow [t_\alpha a \to A]} \quad \text{(S)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [\gamma \bullet \to B, i, j] \Rightarrow t_B}{[\alpha B \bullet \beta \to A, h, j] \Rightarrow [t_\alpha t_B \to A]} \quad \text{(C)}$$

$$
\begin{array}{ll}
[a,0,1] & \Rightarrow a \\
[+,1,2] & \Rightarrow + \\
[a,2,3] & \Rightarrow a \\
[\bullet E + E \rightarrow E,0,0] & \Rightarrow [\rightarrow E] \\
[\bullet a \rightarrow E,0,0] & \Rightarrow [\rightarrow E] \\
[a\bullet \rightarrow E,0,1] & \Rightarrow [a \rightarrow E] \\
[E \bullet + E \rightarrow E,0,1] & \Rightarrow [[a \rightarrow E] \rightarrow E] \\
[E + \bullet E \rightarrow E,0,2] & \Rightarrow [[a \rightarrow E] + \rightarrow E] \\
[\bullet a \rightarrow E,2,2] & \Rightarrow [\rightarrow E] \\
[a\bullet \rightarrow E,2,3] & \Rightarrow [a \rightarrow E] \\
[E + E\bullet \rightarrow E,0,3] & \Rightarrow [[a \rightarrow E] + [a \rightarrow E] \rightarrow E]
\end{array}
$$

Figure 1: Derivation with the parsing schema in Definition 4.3 and the grammar from Example 2.4.

□

Figure 1 shows the derivation of a parse tree for the string $a + a$ with the grammar from Example 2.4. The following theorem states that parsing as defined in Definition 2.3 and derivation with Earley's parsing schema in Definition 4.3 are equivalent.

Theorem 4.4 (Correctness) *Parsing schema Earley with trees derives exactly the trees produced by a parser, i.e.,* $\{t \mid A \in V_N, w \vdash_{\mathcal{G}}^{4.3} [\alpha\bullet \rightarrow A,0,n] \Rightarrow t\} = \Pi(\mathcal{G})(w)$

The following proposition states that the decoration of items with partial parse trees makes no difference to what can be derived. Items in a parsing schema can be annotated with trees as long as they do not affect the deduction.

Proposition 4.5 *Parsing schema Earley with trees preserves the derivations of parsing schema Earley, i.e.,* $w \vdash_{\mathcal{G}}^{4.1} [\alpha \bullet \beta \rightarrow A,i,j] \iff \exists t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha) : w \vdash_{\mathcal{G}}^{4.3} [\alpha \bullet \beta \rightarrow A,i,j] \Rightarrow [t_\alpha \rightarrow A]$

The optimization problem can now be rephrased as:

Definition 4.6 (Optimizing Parsing Schemata) The optimization of a parsing schema P by a disambiguation filter $\mathcal{F}$ constitutes in finding a derived parsing schema P' such that

$$\mathcal{F}(\Pi(\mathcal{G})(w)) \subseteq \{t \mid w \vdash_{\mathcal{G}}^{P'} I \Rightarrow t\} \subseteq \{t \mid w \vdash_{\mathcal{G}}^{P} I \Rightarrow t\}$$

where I is some final item. □

5 Priority Conflicts

We consider the optimization of parsing schema Earley by two disambiguation filters that are used to interpret the priority disambiguation rules of the formalism SDF of Heering *et al.* (1989). This disambiguation method is also used in the generalization of SDF to SDF2 presented in Visser (1997a). The subject of this section is a filter that removes trees with a priority conflict. This filter is similar to the conventional precedence and associativity filter. The declaration of priority rules will also be used in the definition of the multi-set filter in §7.

Definition 5.1 (Priority Declaration) A *priority declaration* $\mathrm{Pr}(\mathcal{G})$ for a context-free grammar $\mathcal{G}$ is a tuple $\langle \mathrm{L}, \mathrm{R}, \mathrm{N}, > \rangle$, where $\oplus \subseteq \mathcal{P} \times \mathcal{P}$ for $\oplus \in \{\mathrm{L}, \mathrm{R}, \mathrm{N}, >\}$, such that L, R and N are symmetric and $>$ is irreflexive and transitive. □

The relations L, R and N declare left-, right- and non-associativity, respectively, between productions. The relation $>$ declares priority between productions. A tree with signature p_1 can not be a child of a tree with signature p_2 if $p_2 > p_1$. The syntax of priority declarations used here is similar to that in Earley (1975). In SDF (Heering *et al.*, 1989) a formalism with the same underlying structure but with a less Spartan and more concise syntax is used. In SDF one writes `left` for L, `right` for R and `non-assoc` for N. We will use both notations.

Definition 5.2 (Priority Conflict) The set conflicts($\mathcal{G}$) generated by the priority declaration of a grammar $\mathcal{G}$ is the smallest set of partial trees of the form $[\alpha[\beta \to B]\gamma \to A]$ defined by the following rules.

$$\frac{\alpha B\gamma \to A > \beta \to B \in \Pr(\mathcal{G})}{[\alpha[\beta \to B]\gamma \to A] \in \text{conflicts}(\mathcal{G})}$$

$$\frac{\gamma \neq \epsilon,\ \beta \to B\ (\texttt{right} \cup \texttt{non-assoc})\ B\gamma \to A \in \Pr(\mathcal{G})}{[[\beta \to B]\gamma \to A] \in \text{conflicts}(\mathcal{G})}$$

$$\frac{\alpha \neq \epsilon,\ \beta \to B\ (\texttt{left} \cup \texttt{non-assoc})\ \alpha B \to A \in \Pr(\mathcal{G})}{[\alpha[\beta \to B] \to A] \in \text{conflicts}(\mathcal{G})}$$

This set defines the patterns of trees with a priority conflict. □

Using the definition of priority conflict we can define a filter on sets of parse trees.

Definition 5.3 (Priority Conflict Filter) A tree t has a *root priority conflict* if its root matches one of the tree patterns in conflicts($\mathcal{G}$). A tree t has a *priority conflict*, if t has a subtree s that has a root priority conflict. The filter $\mathcal{F}^{\Pr}$ is now defined by $\mathcal{F}^{\Pr}(\Phi) = \{t \in \Phi \mid t \text{ has no priority conflict}\}$. The pair $\langle \mathcal{G}, \Pr \rangle$ defines the disambiguated grammar $\mathcal{G}/\mathcal{F}^{\Pr}$. □

Example 5.4 Consider the following grammar with priority declaration

```
syntax
  "a"      -> E
  E "*" E -> E {left}
  E "+" E -> E {left}
priorities
  E "*" E -> E >
  E "+" E -> E
```

Here the attribute `left` of a production p abbreviates the declaration p L p. The tree

$$[[[a \to E] + [a \to E] \to E] * [a \to E] \to E]$$

has a priority conflict over this grammar—it violates the first priority condition since multiplication has higher priority than addition. The tree

$$[[a \to E] + [[a \to E] * [a \to E] \to E] \to E]$$

does not have a conflict. These trees correspond to the (disambiguated) strings $(a + a) * a$ and $a + (a * a)$, respectively. The implication operator in logic is an example of a right associative operator: $a \to a \to a$ should be read as $a \to (a \to a)$. Non-associativity can be used to exclude unbracketed nested use of the equality operator in expressions using the production `E "=" E -> E`. □

The priority conflict filter induced by a priority declaration can be used to optimize the Earley parsing schema. By the following observation a more general optimization problem can be solved.

Definition 5.5 (Subtree Exclusion) A *subtree exclusion filter* based on a set Q of excluded subtrees is defined by

$$\mathcal{F}^{Q}(\Phi) = \{t \in \Phi \mid \neg t \lhd Q\}$$

where $t \lhd Q$ (t is excluded by Q) if t has a subtree that matches one of the patterns in Q. □

The optimized parsing schema should not derive trees that contain a subtree contained in Q. As is shown in definition 4.3 such patterns are constructed in the complete rule and predicted in the predict rule. The construction of trees with priority conflicts can be prevented by adding an extra condition to these rules. This leads to the following adaptation of the Earley parsing schema.

Definition 5.6 (Earley modulo Q) Parsing schema Earley modulo a set Q of parse trees of the form $[\alpha[\gamma \to B]\beta \to A]$, which are excluded as subtrees. The set of items $\mathcal{I}$ and the deduction rules (H), (I) and (S) are copied unchanged from Definition 4.3.

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [\alpha[\gamma \to B]\beta \to A] \notin Q}{[\bullet\gamma \to B, i, i] \Rightarrow [\to B]} \quad \text{(P)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [\gamma\bullet \to B, i, j] \Rightarrow t_B,\ [\alpha[\gamma \to B]\beta \to A] \notin Q}{[\alpha B \bullet \beta \to A, h, j] \Rightarrow [t_\alpha t_B \to A]} \quad \text{(C)}$$

□

The following theorem states that parsing schema in Definition 5.6 is an optimal approximation of the composition of a subtree exclusion filter (with trees of the form $[\alpha[\gamma \to B]\beta \to A]$) and a generalized parser.

Theorem 5.7 (Correctness) *Parsing schema Earley modulo Q derives exactly the trees produced by the composition of a parser and a subtree exclusion filter for Q, i.e.,* $\{t \in \mathcal{T}(\mathcal{G})(A) \mid A \in V_N,\ w \vdash^{5.6}_{\mathcal{G},Q} [A \to \alpha\bullet, 0, n] \Rightarrow t\} = \mathcal{F}^Q(\Pi(\mathcal{G})(w))$

This is proved using two lemmas. The soundness lemma asserts that no intermediate parse tree derived with the deduction rules has an excluded subtree (i.e., a priority conflict). The completeness lemma states that every parse tree without a priority conflict can be derived. The completeness lemma is obtained by reverting the implication of the soundness lemma. In these lemmata we use the notion of a context $t_B[\bullet]$ that represents a tree context of type B with one subtree that is a hole $\bullet$. The instantiation $t_B[t_A]$ of a context $t_B[\bullet]$ is the tree obtained by replacing the $\bullet$ subtree by the tree t_A.

Lemma 5.8 (Soundness) *For all context-free grammars $\mathcal{G}$, strings $w = a_1 \dots a_n \in V_T^*$, symbols $A \in V_N$ and $\alpha, \beta \in V^*$, natural numbers $i \leq j \in \mathbf{N}$, and trees $t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha)$ such that $\alpha\beta \to A \in \mathrm{P}(\mathcal{G})$, and Q a set of parse tree patterns of the form $[\alpha[\gamma \to B]\beta \to A]$ we have that*

$$\frac{w \vdash^{5.6}_{\mathcal{G}} [\alpha \bullet \beta \to A, i, j] \Rightarrow [t_\alpha \to A]}{\begin{array}{l}\mathrm{yield}(t_A = [t_\alpha\beta \to A]) = a_{i+1} \dots a_j\beta, \\ \exists t_B[\bullet] \in \mathcal{T}(\mathcal{G})(B) : \neg t_B[t_A] \lhd Q \wedge \mathrm{yield}(t_B[A]) = a_1 \dots a_i A\delta\end{array}}$$

6 From Earley to LR

There is a close correspondence between Earley's algorithm and LR parsing (Knuth, 1965). In fact, parsing schema Earley in Definition 4.1 can also be considered the underlying parsing schema of an LR(0) parser. The main difference between the algorithms is that in LR parsing the instantiation of the parsing schema with a grammar is compiled into a transition table. Definition 6.1 defines a parsing schema for 'compiled' LR(0) parsing. The intermediate results of an LR parser, the LR states, are sets of LR items closed under prediction, defined by the function *closure*. The function *goto* computes the set of items that results from a state by shifting the dot in the items over a symbol X. The schema defines three deduction rules. Rule (I) generates the initial state consisting of the set of all items $[\bullet\alpha \to A]$ predicting all productions of the grammar. Rule (Sh) obtains a new state from a state by *shifting* a terminal. Rule (Re) reduces a number of states to a new state upon the complete recognition of a production $B_1 \dots B_m \to B$. It is clear that the function *closure* corresponds to the predict rule (P) in Earley, that (Sh) corresponds to (S) and that (Re) corresponds to (C). A goto-graph is a precomputation of the goto function. Figure 2 shows a goto-graph for the grammar of Example 2.4.

Definition 6.1 (LR(0) Parsing) LR items are Earley items without indices. The items used in LR parsing are sets of LR-items with a pair of indices.

$$\mathcal{I}_{LR} = \{[\alpha \bullet \beta \to A] \mid \alpha\beta \to A \in \mathrm{P}(\mathcal{G})\} \quad \mathcal{I} = \{[\Phi, i, j] \mid \Phi \subseteq \mathcal{I}_{LR}\}$$

The closure of a set of items Φ is the smallest set of items containing Φ and closed under prediction, i.e.,

$$\frac{}{\Phi \subseteq \mathrm{closure}(\Phi)}$$

$$\frac{[\alpha \bullet B\beta \to A],\ \gamma \to B \in \mathrm{P}(\mathcal{G})}{[\bullet\gamma \to B] \in \mathrm{closure}(\Phi)}$$

Given a symbol X the goto function maps a set of items to the closure of the set obtained by shifting all items with X.

$$\mathrm{goto}(X, \Phi) = \mathrm{closure}(\{[\alpha X \bullet \beta \to A] \mid [\alpha \bullet X\beta \to A] \in \Phi\})$$

Given these functions an LR parser is defined[1] by the following deduction rules.

$$\frac{}{[\{[\bullet\alpha \to A] \mid \alpha \to A \in \mathcal{G}\}, 0, 0]} \quad \text{(I)}$$

$$\frac{[\Phi, h, i],\ [a, i, i+1]}{[\mathrm{goto}(a, \Phi), h, i+1]} \quad \text{(Sh)}$$

$$\frac{[\Phi^{[\alpha \bullet B\beta \to A]}, h, i],\ [\Phi_1^{[B_1 \bullet \ldots B_k \to B]}, i, i_1],\ \ldots\ [\Phi_k^{[B_1 \ldots B_k \bullet \to B]}, i, i_k]}{[\mathrm{goto}(B, \Phi), h, i_k]} \quad \text{(Re)}$$

□

In the same way that an LR parser is derived from the Earley schema an LR parser can be derived from the optimized parsing schema of Definition 5.6 by adapting the closure and goto functions.

Definition 6.2 (LR(0) parsing modulo Q) Items are only predicted if they do not lead to a conflict.

$$\frac{[\alpha \bullet B\beta \to A],\ \gamma \to B \in \mathrm{P}(\mathcal{G}),\ [\alpha[\gamma \to B]\beta \to A] \notin Q}{[\bullet\gamma \to B] \in \mathrm{closure}(\Phi)}$$

Given a production $\gamma \to B$ the goto function maps a set of items to the closure of the set obtained by shifting all items with $\gamma \to B$ for which that does not lead to a conflict.

$$\mathrm{goto}(\gamma \to B, \Phi) = \mathrm{closure}(\{[\alpha B \bullet \beta \to A] \mid [\alpha \bullet B\beta \to A] \in \Phi \wedge [\alpha[\gamma \to B]\beta \to A] \notin Q\})$$

□

Note that the goto function has to be parameterized with the production that is recognized instead of with just the symbol. (For the (Sh) rule the old goto function is used.) Figure 3 shows the goto-graph for the disambiguated grammar from Example 5.4.

6.1 SLR(1) Parsing

The LR(0) goto graph in Figure 3 contains conflicts that are easy to prevent with the SLR(1) (Simple LR(1)) extension of LR(0) parsing due to DeRemer (1971). The SLR algorithm is based on the observation that a reduction is only useful if the next symbol in the string can follow the symbol that is recognized by the reduction, i.e., the right hand-side of the production that is reduced. This is expressed in the following adaptation of the LR(0) parsing schema of Definition 6.1. The function $\mathrm{first}(\alpha, \Psi)$ yields the set of symbols that can start a phrase derived from a string of symbols α followed by a symbol from the set Ψ. The expression $\mathrm{follow}(B, \Psi)$ denotes the set of symbols that can follow symbol B in a phrase that is followed by a symbol from the set Ψ. The reduce rule now only applies if a production has been recognized *and* the next symbol in the string can follow the right-hand side of the production.

[1] This definition gives the intermediate results of an LR parser, not its exact control flow.

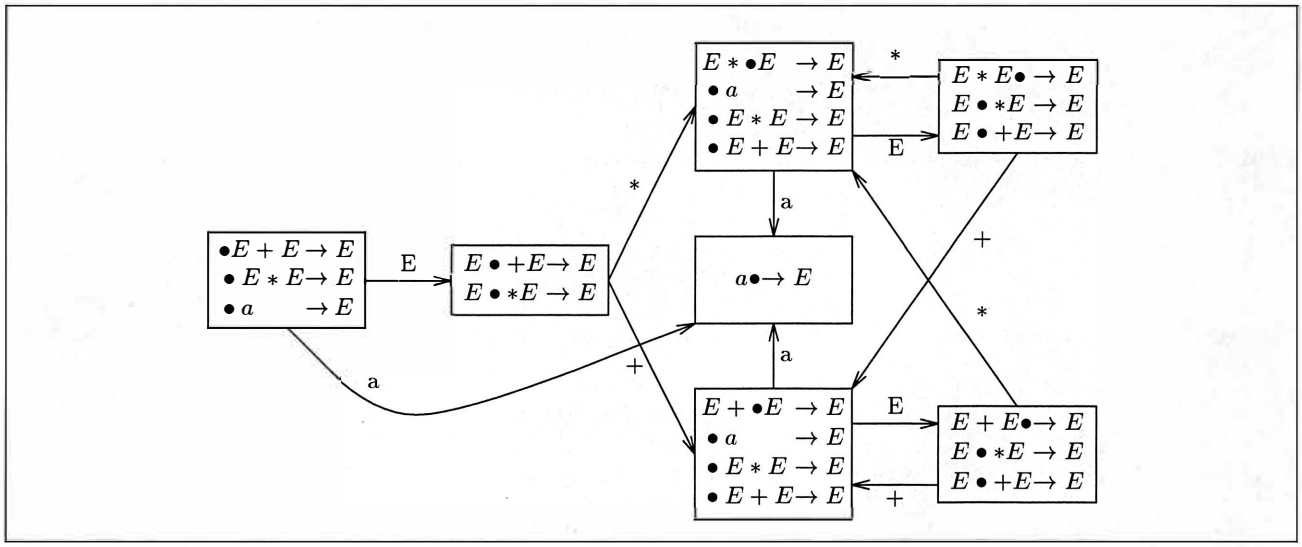


Figure 2: LR(0) goto graph for the grammar of Example 2.4

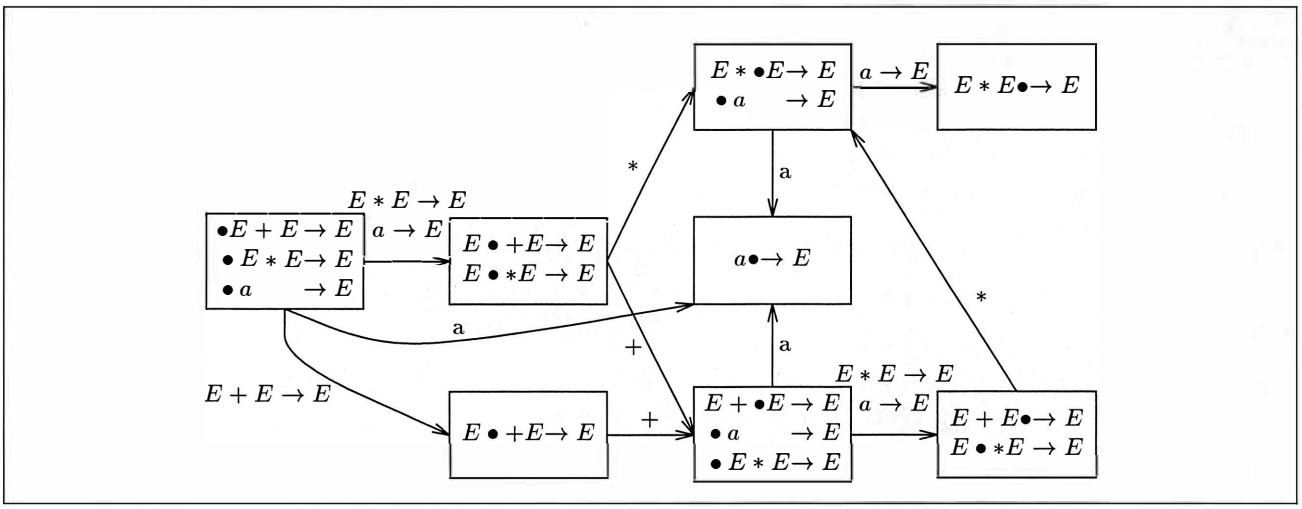


Figure 3: LR(0) goto graph for the grammar of Example 5.4.

Definition 6.3 (SLR(1) Parsing) The set of first symbols of a phrase generated by a string α followed by an element from Ψ, is the smallest set first(α, Ψ) such that

$$\begin{aligned} \text{first}(\epsilon, \Psi) &= \Psi \\ \text{first}(a\alpha, \Psi) &= \{a\} \\ \text{first}(A\alpha, \Psi) &= \bigcup_{\beta \to A \in \mathrm{P}(\mathcal{G})} \text{first}(\beta\alpha, \Psi) \end{aligned}$$

The set of symbols that can follow a symbol B in a phrase generated by $\mathcal{G}$ followed by a symbol from Ψ is the smallest set such that

$$\frac{\alpha B \beta \to A \in \mathrm{P}(\mathcal{G})}{\text{follow}(B, \Psi) \supseteq \text{first}(\beta, \text{follow}(A, \Psi))}$$

state	a	$*$	$+$	\$	1	2	3	4
0	s 1				3	3	4	2
1		r 1	r 1	r 1				
3		s 8	s 5	acc				
4			s 5	acc				
5	s 1				7	7		
7		s 8	r 3	r 3				
8	s 1				9			
9		r 2	r 2	r 2				

(1) $a \rightarrow E$
(2) $E * E \rightarrow E$
(3) $E + E \rightarrow E$
(4) $E\,\$ \rightarrow S$

(2) > (3)
(2) L (2)
(3) L (3)

Figure 4: SLR(1) table for the grammar of example 5.4. s n denotes *shift to state* n, r n denotes reduce with production n, acc denotes *accept*. The right part of the table contains the goto entries for the productions. This parse table corresponds to the goto graph of figure 3.

The reduce rule of the schema in Definition 6.1 is restricted by requiring that the next symbol in the string is an element of the follow set of B.

$$\frac{[\Phi^{[\alpha\bullet B\beta\rightarrow A]}, h, i],\ [\Phi_1^{[B_1\bullet\ldots B_k\rightarrow B]}, i, i_1],\ \ldots,[\Phi_k^{[B_1\ldots B_k\bullet\rightarrow B]}, i, i_k],\ [a, i_k, i_k+1],\ a \in \text{follow}(B, \{\$\})}{[\text{goto}(B, \Phi), h, i_k]}$$

□

The SLR(1) schema can be adapted in the same way as the LR(0) schema to account for priority conflicts (or subtree exclusion). However, the definition of follow above is too weak for this extended schema. For instance, in the grammar of Example 5.4, the token $*$ is an element of the follow set of E. However, $*$ can not follow an E if it is a $E + E \rightarrow E$, i.e., if a reduction is done with $E + E \rightarrow E$, no action for $*$ is possible. The following parsing schema optimizes the SLR(1) parsing schema by defining the follow set for a production instead of for a symbol and adapting the reduce rule accordingly. Figure 4 shows the SLR(1) table for the grammar of Example 5.4.

Definition 6.4 (SLR(1) Parsing Modulo Q) This schema defines SLR(1) parsing modulo a set Q of parse trees of the form $[\alpha[\beta \rightarrow B]\gamma \rightarrow A]$ using the definition of the closure and goto functions from the parsing schema in Definition 6.2 and the definition of first from Definition 6.3.

$$\frac{\alpha B\gamma \rightarrow A \in \text{P}(\mathcal{G}),\ [\alpha[\beta \rightarrow B]\gamma \rightarrow A] \notin Q}{\text{follow}(\beta \rightarrow B, \Psi) \supseteq \text{first}(\beta, \text{follow}(\alpha B\gamma \rightarrow A, \Psi))}$$

The reduce rule is adapted to the new definition of follow.

$$\frac{[\Phi^{[\alpha\bullet B\beta\rightarrow A]}, h, i],\ [\Phi_1^{[B_1\bullet\ldots B_k\rightarrow B]}, i, i_1],\ \ldots,[\Phi_k^{[B_1\ldots B_k\bullet\rightarrow B]}, i, i_k],\ [a, i_k, i_k+1],\ a \in \text{follow}(B_1\ldots B_k \rightarrow B, \{\$\})}{[\text{goto}(B_1\ldots B_k \rightarrow B, \Phi), h, i_k]} \quad \text{(Re)}$$

□

6.2 Discussion

Conventional methods for disambiguating grammars that apply to LR parsing disambiguate the grammar by solving conflicts in an existing LR table. The classical method of Aho *et al.* (1975) uses associativity and precedence information of a limited form—a linear chain of binary operators that have non-overlapping operator syntax—to solve shift/reduce conflicts in LR tables. The method is based on observations on how such conflicts should be solved given precedence information, without a real understanding of the cause of the conflicts. Aasa (1991, 1992) describes filtering of sets of parse trees by means of precedences. Thorup (1994a) describes a

method that tries to find a consistent solution for all conflicts in an LR table starting from, and producing a set of excluded subtrees.

All these methods fail on grammars that are inherently non-LR(k), i.e., for which there is no complete solution of all conflicts in any LR table for the grammar. An example is the grammar

```
syntax
                -> L
  L [\ \t\n]    -> L
  "a"           -> E
  E L "*" L E -> E {left}
  E L "+" L E -> E {left}
priorities
  E L "*" L E -> E >
  E L "+" L E -> E
```

that models arithmetic expressions with layout. The tokens of expressions can be separated by any number of spaces, tabs or newlines, which requires unbounded lookahead. Such grammars are the result of integrating the lexical syntax and context-free syntax of a language into a single grammar as is proposed in Visser (1997b). Parsers for such grammars are called scannerless parsers because the tokens they read are the characters from the input file. This grammar is disambiguated completely (it has no ambiguous sentences) with priorities, resulting in an LR table that contains some LR-conflicts, but that does not produce trees with priority conflicts. In combination with a nondeterministic interpreter, e.g., Tomita's generalized LR algorithm (Tomita, 1985), of the parse tables this gives an efficient disambiguation method for languages on the border of determinism.

Thorup (1994b) describes a transformation on grammars based on a set of excluded subtrees to disambiguate a grammar. This method could be used to generate conflict free parse tables as far as possible. Because such a transformation introduces new grammar symbols, more states and transitions are needed in the parse table than for the original grammar. Since the method defined above also introduces some extra states, it would be interesting to compare the LR tables produced by both methods.

7 Multi-set Filter

The multi-set ordering on parse trees induced by a priority declaration solves ambiguities not solvable by priority conflicts. A certain class of ambiguities solved by the multi-set order does not need the full power of multi-sets, only a small part of both trees are actually compared. Based on this observation an optimization of the Earley schema that partially implements the multi-set filters can be defined.

Definition 7.1 (Multi-sets) A *multi-set* is a function $M : \mathrm{P}(\mathcal{G}) \to \mathbf{N}$ that maps productions to the number of their occurrences in the set. The union $M \uplus N$ of two multi-sets M and N is defined as $(M \uplus N)(p) = M(p) + N(p)$. The empty multi-set is denoted by $\emptyset$, i.e., $\emptyset(p) = 0$ for any p. We write $p \in M$ for $M(p) > 0$. A multi-set with a finite number of elements with a finite number of occurrences can be written as $M = \{p_1, p_1, \ldots, p_2, \ldots\}$, where $M(p)$ is the number of occurrences of p in the list. A parse tree t is interpreted as a multi-set of productions by counting the number of times a production acts as the signature of a subtree of t, where $\alpha \to A$ is the signature of $[t_\alpha \to A]$. □

The following definition due to Jouannaud and Lescanne (1982) defines an ordering on multi-sets.

Definition 7.2 (Multi-set Order) Given some priority declaration $\mathrm{Pr}(\mathcal{G})$, the order $\prec^{\mathrm{Pr}(\mathcal{G})}$ on multi-sets is defined as

$$M \prec^{\mathrm{Pr}(\mathcal{G})} N \iff$$
$$M \neq N \wedge \forall y \in M : M(y) > N(y) \Rightarrow \exists x \in N : y >^{\mathrm{Pr}(\mathcal{G})} x \wedge M(x) < N(x)$$

□

Definition 7.3 (Multi-set Filter) Given a priority relation $\mathrm{Pr}(\mathcal{G})$, the multi-set filter $\mathcal{F}^{\prec^{\mathrm{Pr}(\mathcal{G})}}$ is defined by

$$\mathcal{F}^{\prec^{\mathrm{Pr}(\mathcal{G})}}(\Phi) = \{t \in \Phi \mid \neg\exists s \in \Phi : s \prec^{\mathrm{Pr}(\mathcal{G})} t\}$$ □

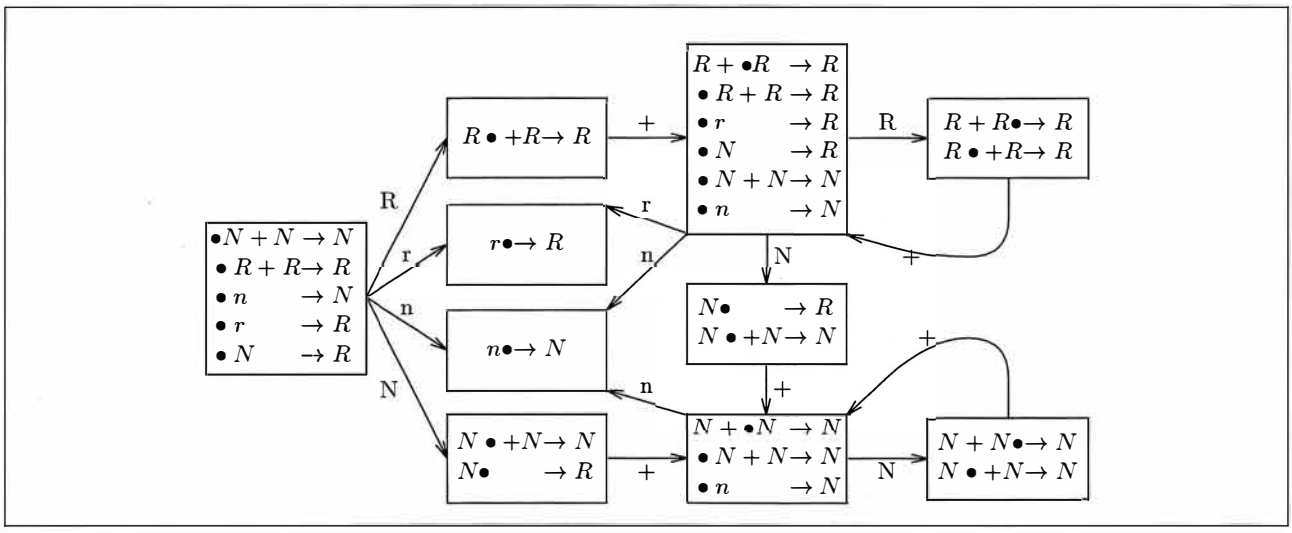


Figure 5: Goto graph for the grammar of Example 7.4

The motivation for this filter is that it prefers parse trees that are constructed with the smallest possible number of productions of the highest possible priority.

Example 7.4 Consider the grammar

```
syntax
  "n"      -> N
  N "+" N -> N
  N        -> R
  "r"      -> R
  R "+" R -> R
```

that describes the language of 'naturals' and 'reals' with an overloaded addition operator. The sentence $n + n$ can be parsed as $[[n \to N] + [n \to N] \to N]$ and as $[[[n \to N] \to R] + [[n \to N] \to R] \to R]$. This ambiguity can be solved, choosing either the first or the second tree, by declaring one of the priority rules

```
N "+" N -> N > R "+" R -> R
```

or

```
R "+" R -> R > N "+" N -> N
```

Note that with the second priority rule, the production $N + N \to N$ is only used as a parse tree in a context where no R is allowed. Therefore, the first priority rule is assumed in further examples. □

The multi-set order is too strong for this kind of disambiguation. To solve the ambiguity there is no need to compare the complete trees, as the multi-set order does. Comparing the patterns $[[N + N \to N] \to R]$ and $[[N \to R] + [N \to R] \to R]$ is sufficient. The goto graph corresponding to the Earley parser for the example grammar (Figure 5) shows that the partial phrase $n+$ causes a conflict (in the left-most state at the bottom row) after completing the production $[n \to N]$. The parser can either shift with + or complete with the chain rule of $N \to R$. However, only after having seen what follows the + a decision can be made. In the following adaptation of the Earley parsing schema the cause of these early decision problems is solved by not predicting and completing chain production, but instead storing them in items.

Definition 7.5 (Earley modulo Chain Rules) Let V_C be the set containing all chain symbols $[C \to B]$, where B and C are nonterminals in context-free grammar $\mathcal{G}$ and $B = C$ or $C \to_\mathcal{G} B_1 \to_\mathcal{G} \cdots \to_\mathcal{G} B_m \to_\mathcal{G} B$, $(m \geq 0)$. Symbols $[A \to A]$ and A are identified. A production with chain symbols $[C \to B]$ in its left-hand side is identifiable (member of grammar, priority relation) with a production where the chain symbols are replaced

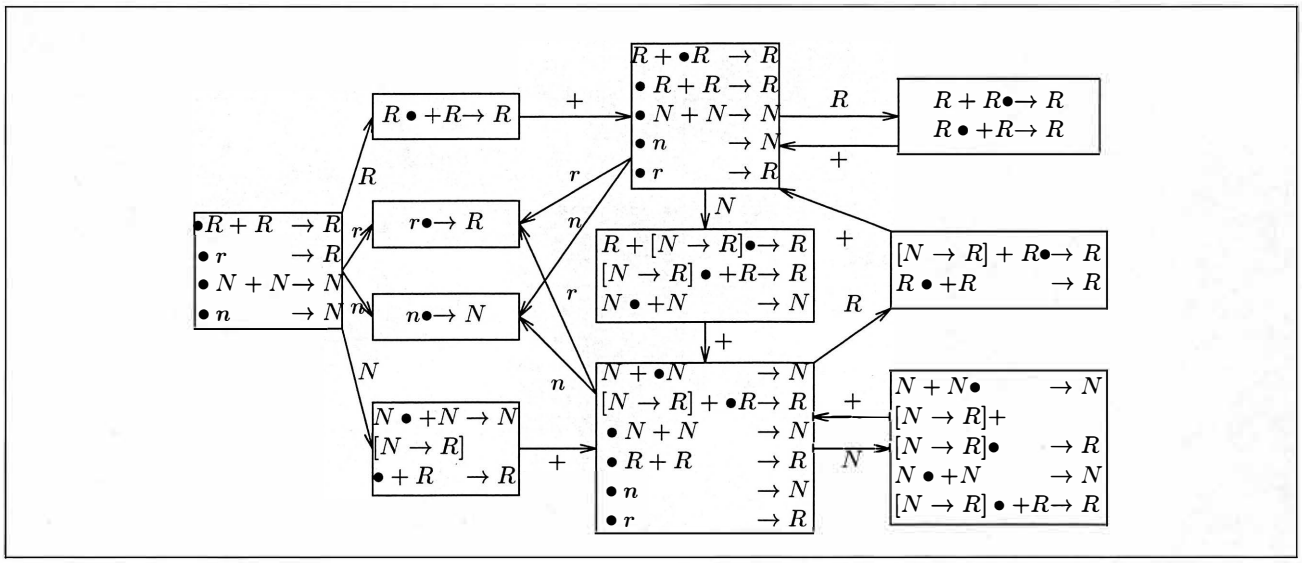


Figure 6: Goto graph for grammar of Example 7.4 corresponding to parsing schema in Definition 7.5. The item $[[N \to R] + [N \to R]\bullet \to R]$ is present if the negative premise of rule (C2) is absent.

with their heads B. The (I) and (S) rules are as usual.

$$\frac{\alpha\beta \to A \in \mathcal{G},\ |\alpha\beta| \neq 1 \vee \alpha\beta = a \in V_T,\ 0 \leq i \leq j}{[\alpha \bullet \beta \to A, i, j] \in \mathcal{I}}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i],\ [C \to B] \in V_C}{[\bullet\gamma \to C, i, i]} \tag{P}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i],\ [\gamma\bullet \to C, i, j],\ |\beta| > 0}{[\alpha[C \to B] \bullet \beta \to A, h, j]} \tag{C1}$$

$$\frac{[\alpha \bullet B \to A, h, i], [\gamma\bullet \to C, i, j], \neg[\alpha'\bullet \to A', h, j],\ \alpha' \to A' > \alpha B \to A}{[\alpha[C \to B]\bullet \to A, h, j]} \tag{C2}$$

□

The negative premise $\neg[\alpha\bullet \to A, i, j]$ in combination with the condition $A' \to \alpha' > A \to \alpha B$ is used in rule (C2) to express that an item $[\alpha[C \to B]\bullet \to A, h, j]$ can be derived from $[\alpha \bullet B \to A, h, i]$ and $[\gamma\bullet \to C, i, j]$ only if no item $[\alpha'\bullet \to A', h, j]$ can be derived such that $A' \to \alpha'$ has higher priority than $A \to \alpha B$.

With the introduction of negative premises we leave the domain of parsing schemata as defined in Sikkel (1993) and this deserves a more thorough investigation than is possible in the scope of this paper. However, two points about this feature can be observed: (1) As used here the notion has a straightforward implementation in an LR-like compilation scheme: first construct the complete set of items and then choose the maximal items from it. (2) The priority relation $>$ on productions is irreflexive by definition, which entails that rule (C2) has no instantiation of the form $I_1, I_2, \neg I_3 \vdash I_3$ that would make the schema inconsistent.

Example 7.6 Figure 6 shows the goto graph for the grammar of Example 7.4 according to the parsing schema in Definition 7.5. The shift/reduce conflict between the items $[N \bullet +N \to N]$ and $[N\bullet \to R]$ is changed into a reduce/reduce conflict between the items $[N + N\bullet \to N]$ and $[[N \to R] + [N \to R]\bullet \to R]$. If the negative premise of rule (C2) is taken into account the item $[[N \to R] + [N \to R]\bullet \to R]$ can not be derived, and is not present in the goto-graph. The conflict is solved. □

The method does not help for grammars where the ambiguity is not caused by chain rules, for instance consider the following example due to Kamperman (1992)

```
syntax
  E E     -> E
```

```
    "-" E   -> E
    E "-" E -> E
  priorities
    E E     -> E >
    "-" E   -> E >
    E "-" E -> E
```

It defines expressions formed by concatenation, prefix minus and infix minus.

The methods developed in this paper can be combined into a parsing schema that handles both priority conflicts and the partial implementation of multi-set filters by adding the subset exclusion conditions to the (P), (C1) and (C2) rules of the parsing schema in Definition 7.5. As a bonus this combined parsing schema handles priority conflicts modulo chain rules.

8 Conclusions

In this paper two disambiguation methods specified as a filter on sets of parse trees were considered. These filters were used to optimize parsers for context-free grammars by adapting their underlying parsing schema.

The first optimization uses priority conflicts to prevent ambiguities. The resulting Earley parsers modulo priority conflicts are guaranteed not to produce trees with priority conflicts, even for grammars with overlapping operators, layout in productions or other problems that need unbounded lookahead. In combination with a GLR interpreter of the parse tables this gives an efficient disambiguation method for languages with unbounded lookahead. The second optimization covers a subset of the ambiguities solved by multi-set filters. Together these optimizations can be used in the generation of efficient parsers for a large class of ambiguous context-free grammars disambiguated by means of priorities.

Parsing schemata provide a high-level description of parsing algorithms that is suitable for the derivation of new algorithms. The introduction of negative items was needed to express the optimization for the multiset filter and needs more research. This first experiment in implemenation of disambiguation methods from formal specifications encourages research into a fuller optimization of multiset filters and application of this approach to other disambiguation methods.

The deductive parsing approach of Shieber *et al.* (1995) and its implementation in Prolog could be used to prototype such optimized schemata. Deductive parsing consists in computing the closure of a set of axiom items under the inference rules of a schema, resulting in all items derivable for a sentence. Compiling the inference rule of a schema into a parse table for a specific grammar increases the efficiency of an algorithm, since work is shifted from the parser into the parser generator. It seems feasible to generalize the compilation of Earley rules into LR tables to other schemata, thus obtaining a very declarative method for creating new parser generators.

Acknowledgements I thank Mark van den Brand and several referees for their comments on this paper. This research was supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO). Project 612-317-420: Incremental parser generation and context-dependent disambiguation, a multi-disciplinary perspective.

References

Aasa, A. (1991). Precedences in specifications and implementations of programming languages. In J. Maluzynski and M. Wirsing, editors, *Programming Language Implementation and Logic Programming*, volume 528 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag.

Aasa, A. (1992). *User Defined Syntax*. Ph.D. thesis, Department of Computer Sciences, Chalmers University of Technology and University of Göteborg, S-412 96 Göteborg, Sweden.

Aho, A. V., Johnson, S. C., and Ullman, J. D. (1975). Deterministic parsing of ambiguous grammars. *Communications of the ACM*, **18**(8), 441–452.

DeRemer, F. L. (1971). Simple LR(k) grammars. *Communications of the ACM*, **14**, 453–460.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, **13**(2), 94–102.

Earley, J. (1975). Ambiguity and precedence in syntax description. *Acta Informatica*, **4**(1), 183–192.

Heering, J., Hendriks, P. R. H., Klint, P., and Rekers, J. (1989). The syntax definition formalism SDF – reference manual. *SIGPLAN Notices*, **24**(11), 43–75.

Jouannaud, J.-P. and Lescanne, P. (1982). On multiset orderings. *Information Processing Letters*, **15**(2), 57–63.

Kamperman, J. (1992). A try at improving the second disambiguation phase in SDF. technical note.

Klint, P. and Visser, E. (1994). Using filters for the disambiguation of context-free grammars. In G. Pighizzini and P. San Pietro, editors, *Proc. ASMICS Workshop on Parsing Theory*, pages 1–20, Milano, Italy. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control*, **8**, 607–639.

Rekers, J. (1992). *Parser Generation for Interactive Environments*. Ph.D. thesis, University of Amsterdam. ftp://ftp.cwi.nl/pub/gipe/reports/Rek92.ps.Z.

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *The Journal of Logic Programming*, pages 3–36.

Sikkel, K. (1993). *Parsing Schemata*. Ph.D. thesis, Universiteit Twente, Enschede.

Sikkel, K. (1994). How to compare the structure of parsing algorithms. In G. Pighizzini and P. San Pietro, editors, *Proc. ASMICS Workshop on Parsing Theory*, pages 21–39, Milano, Italy. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano.

Sikkel, K. (1997). *Parsing Schemata. A Framework for Specification and Analysis of Parsing Algorithms*, volume XVI of *Texts in Theoretical Computer Science. An EATCS Series*. Springer-Verlag, Berlin / Heidelberg / New York.

Thorup, M. (1994a). Controlled grammatic ambiguity. *ACM Transactions on Programming Languages and Systems*, **16**(3), 1024–1050.

Thorup, M. (1994b). Disambiguating grammars by exclusion of sub-parse trees. Technical Report 94/11, Dept. of Computer Science, University of Copenhagen, Denmark.

Tomita, M. (1985). *Efficient Parsing for Natural Languages. A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers.

Visser, E. (1997a). A family of syntax definition formalisms. Technical Report P9706, Programming Research Group, University of Amsterdam.

Visser, E. (1997b). Scannerless generalized-LR parsing. Technical Report P9707, Programming Research Group, University of Amsterdam.

NEW PARSING METHOD USING GLOBAL ASSOCIATION TABLE

Juntae Yoon and **Seonho Kim** and **Mansuk Song**
{queen,pobi,mssong}@december.yonsei.ac.kr
Department of Computer Science
Yonsei University, Seoul, Korea

Abstract

This paper presents a new parsing method using statistical information extracted from corpus, especially for Korean. The structural ambiguities are occurred in deciding the dependency relation between words in Korean. While figuring out the correct dependency, the lexical associations play an important role in resolving the ambiguities. Our parser uses statistical cooccurrence data to compute the lexical associations. In addition, it can be shown that sentences are parsed deterministically by the global management of the association. In this paper, the global association table(GAT) is defined and the association between words is recorded in the GAT. The system is the hybrid semi-deterministic parser and is controlled not by the condition-action rule, but by the association value between phrases. Whenever the expectation of the parser fails, it chooses the alternatives using a chart to remove the backtracking.

1 Introduction

The association of words takes an important role in finding out the dependency relation among them. The associations among words or phrases are indicators of the lexical preference. Many works have shown that the association value computed with statistical information makes good results in resolving structural ambiguities(Hindle, 1993; Magerman, 1995; Collins, 1996). Statistical information has led recent researches for syntactic analysis not to the problem of recognizing sentences by given grammar but to that of finding the correct one in multiple parse trees.

A chart parser has been used to produce all possibilities when a sentence is analysed. However, it generates too many structures trying to find a correct one. While reading a sentence, in many cases, a reader can make decisions without examining the entire sentence. A deterministic parser has been worked with the determinism hypothesis for natural language parsing(Marcus, 1980; Faisal et al., 1994). The deterministic parser makes errorneous results because of the limited lookahead, however.

This paper presents a new parsing method that uses the lexical association for parsing sentences semi-deterministically. First, a global association table(GAT) is defined to record and manage the association. As all the associations can be globally observed through the GAT, the parser can obviate the error caused by the limited lookahead. The associations among words are estimated on the basis of lexical association calculated using data acquired from corpus. Next, a parsing algorithm is described using the GAT. The parser selects the action according to the association among the nodes presented by the GAT. That is, the parser is controlled not by condition-action rules, but by the associations between phrases. It merges one phrase with another phrase that has the highest association value, or will wait until it meets the most probable candidate indicated by the GAT. To recapitulate, our system is the parser with the lookahead buffer of sentence length. Experiments show that it doesn't lose accuracy as well as it is as efficient as the deterministic parser.

2 The Characteristics of Korean

2.1 Structures of Korean sentences

Korean is an agglutinative language and has different features from an inflectional language such as English. A sentence consists of a sequence of *eojeols* composed of a content word and functional words. A content word

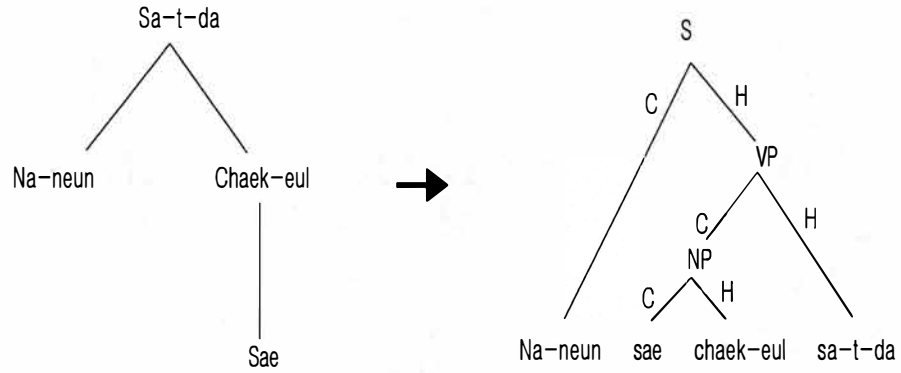


Figure 1: the dependency tree and the binary tree for ex 1)

determines the characterization of a phrase. For instance, an eojeol whose content word is a verb, an adjective, or a copula, functions as a predicate. A functional word directs the grammatical function of an eojeol. For examples, '*eul/reul*' is the postposition(functional word) that makes a nominal eojeol an object. '*Seupnika*' is the sentential ending for a question and '*t/eot*' converts a predicate to the past tense in Korean.

ex 1) *Na-neun sae chaek-eul sa-t-da.*
I new book bought.
— I bought a new book.

'*Na-neun*' is the subject of the above sentence, and '*chaek-eul*', the object.

Second, Korean is an SOV('*Subject Object Verb*' order) language, where the head always follows its complements. In Korean, a head eojeol follows its complement eojeols. A new phrase is generated when one or more eojeols are merged, and the head of the phrase is always the last eojeol of the phrase.

ex 2) *Keu-ga norae-reul booreu-myu hakkyo-e ga-t-da.*
He a song singing to school went
— He went to school singing a song.

Both '*booreu-myu*' and '*ga-t-da*' have verbs as their content words. Predicative eojeols are the heads for nominal eojeols and follow the eojeols, '*keu-ga*', '*norae-reul*', and '*hakkyo-e*' , respectively.

Third, the grammatical dependency relation is determined decisively by functional words. For example, '*in the box*' in English can modify both a noun and a verb. In contrast, in order that '*sangja*', which means *box*, modifies a noun, it has to have the postposition, '*ui*' in Korean. Whenever it has another postpositions, it is the complement of a verb. There is syntactic levels in Korean, and the dependency relation of an eojeol is fixed according to the level.

2.2 Syntactic analysis

The dependency tree of a sentence is built up through parsing. For the operation of the deterministic parser such as CREATE, the dependecy tree is represented by the binary tree for phrase structure that has two children nodes for a complement and a head. The complement node is the dependent node of the dependency tree and the head node is the head of the dependent node. Consequently, the parser uses binary grammar described with the feature structure about the morphemes that constitute an eojeol. The parent node inherits the feature of its head node. Since the head follows its complement in Korean, the root node of the parse tree inherits the feature of the last eojeol in the sentence.

(Figure 1) shows the dependency tree and our parse tree of the sentence given in example 1. '*Na-neun*(I)' is a nominal eojeol and dependent on the predicative eojeol, '*sa-t-da*(bought)'. The feature of '*sa-t-da*' is placed in the root node of the parse tree because it is the head of the sentence.

saengsan/NN(production)	soodan/NN(method)
dambae/NN(tobacco)	nongsa/NN(farming)
sooyo/NN(demand)	byunhwa/NN(variation)
france/NN(France)	moonhak/NN(literature)

Figure 2: Examples of co-occurrence pairs of two nouns

3 Global Association Table

The global association table(GAT) has the association between words or eojeols that have dependency relation. The association can be estimated in various ways; for example, if it is likely that a word depends on the word nearest to it, the estimation function would be given as follows.

$$Assoc(\epsilon_i, \epsilon_j) = 1/d$$

Let the row and the column of the GAT represent eojeols occurring to the left-hand side and to the right-hand side, respectively, in the parsing process. The left-hand side eojeol is a complement, and the right-hand side, the candidate for its head. $GAT(i, j)$ indicates the degree of association in case the ith eojeol is dependent on the jth eojeol. Because the head follows its complement in Korean, and the table is a triangular matrix.

3.1 Estimation function for association

Two kinds of co-occurrence data were extracted from 30 million eojeol corpus. One is for compound noun analysis, the other is for dependency analysis of verb and noun. The associations of modifier-head relations such as an adverb and a verb, or a pre-noun and a noun, are estimated by distance. Distance measure is also used for the case there is no co-occurrence data, which is caused by data sparseness. The distance has been shown to be the most plausible estimation method without any linguistic knowledge(Collins, 1996; Kurohashi et al., 1994). First of all, co-occurrence pairs of two nouns were collected by the method presented in (Pustejovksy et al., 1993). Let N be the set of eojeols consisting of only a noun and NP the set of eojeols composed of a noun with a postposition. From $\epsilon_1, \epsilon_2, \epsilon_3$ ($\epsilon_1 \notin N$, $\epsilon_2 \in N, \epsilon_3 \in NP$), we can obtain complete noun compounds, (n_2, n_3) such that n_2 and n_3 are the nouns that belong to the eojeols, ϵ_2 and ϵ_3, respectively. The parser analyzes compound nouns, based on the complete noun compounds. (Figure 2) shows an example of compound noun pairs.

The association between nouns is computed using the co-occurrence data extracted by the above method. Let

$$N = \{n_1, \ldots, n_m\}$$

N be the set of nouns. Given $n_1, n_2 \in N$, association score, $Assoc$, between n_1 and n_2 is defined to be

$$\begin{aligned} Assoc_{NN}(n_1, n_2) &= P(n_1, n_2) \\ &= \frac{freq(n_1, n_2)}{\sum_i \sum_j freq(n_i, n_j)} \end{aligned} \quad (1)$$

As mentioned above, the distance measure is suggested without any cooccurrence data. Therefore, these estimators are sequentially applied for two eojeols in the following way. Here, ϵ_i and ϵ_j are the ith and the jth nominal eojeols, and n_i and n_j the nouns that belongs to the nominal eojeols.

$$\begin{aligned} &If\ Assoc_{NN}(n_i, n_j) \neq 0 \\ &\quad Assoc(\epsilon_i, \epsilon_j) = Assoc_{NN}(n_i, n_j) \\ &else\ Assoc(\epsilon_i, \epsilon_j) = 1/d \end{aligned}$$

Because the associations are calculated and compared for all ϵ_j on which ϵ_i have the possiblity to be dependent, the compound noun analysis is based on the dependency model rather than the adjacency model(Kobayasi et al., 1994; Lauer, 1995). Because the two estimate functions are used, the extra-comparison routine is required. It will be explained in the next section.

Second, the co-occurrence pairs of nominal eojeols and predicative eojeols were extracted by the partial parser from a corpus. (Figure 3) shows an example of the triples generated from the text. In (Figure 3), the triple,

gada/VB(go)	gajok/NN(family)	ga(SUBJ)
gada/VB(go)	keu/PN(he)	ga(SUBJ)
gada/VB(go)	kang/NN(river)	e(TO)
masida/VB(drink)	maekjoo/NN(beer)	reul(OBJ)
masida/VB(drink)	mool/NN(water)	reul(OBJ)
masida/VB(drink)	neo/PN(he)	ga(SUBJ)

Figure 3: Examples of triples extracted from the text

⟨*masida*/VB, *mool*/NN, *reul*/OBJ⟩ indicates that the verb, '*masida*', and the noun, '*mool*' which mean 'drink' and 'water' respectively, co-occur under the grammatical circumstance, '*reul*' which is the postposition that makes a noun an object.

The association between a verb and a noun is evaluated based on the triples obtained by the above method. Let

$$V = \{v_1, \ldots, v_l\},\ N = \{n_1, \ldots, n_m\}$$
$$S = \{ga, reul, e, \ldots\}$$

V, N, S be the sets of predicates, nouns and syntactic relations respectively. Given $v \in V, s \in S, n \in N$, association score, $Assoc$, between v and n with syntactic relation s is defined to be

$$Assoc_{VN}(n, s, v) = \lambda_1 P(n, s|v) + \lambda_2 P(s|v) \quad (\lambda_1 \gg \lambda_2) \tag{2}$$

The conditional probability, $P(n, s|v)$ measures the strength of the statistical association between the given verb, v, and the noun, n, with the given syntactic relation, s. That is, it favors those that have more co-occurrences of nouns and syntactic relations for verbs. However, the above formula, including the maximum liklihood estimator, suffers from the problem of data sparseness. To back off the estimation, it is introduced the probability, $P(s|v)$ that means how much the verb requires the given syntactic relation.

The association measure based on the distance between two eojeols is used without any co-occurrence data. These estimators are applied sequentially in the following way. Let us suppose that ϵ_i be the ith eojeol and ϵ_j the jth eojeol. In addition, n_i and s_i are the noun and the postposition in the nominal eojeol, ϵ_i, and v_j the verb in the predicative eojeol, ϵ_j, respectively.

$$If\ Assoc_{VN}(n_i, s_i, v_j) \neq 0$$
$$Assoc(\epsilon_i, \epsilon_j) = Assoc_{VN}(n_i, s_i, v_j)$$
$$else\ Assoc(\epsilon_i, \epsilon_j) = 1/d$$

3.2 Making GAT

The association value of two eojeols is recorded in the GAT only when the eojeols have dependency relation. Above all, the dependency relation of two eojeols is checked, therefore. For two eojeols to have dependency relation indicates that they have a possibility to be combined in parsing process. For example, a nominal eojeol with the postposition for case mark depends on a predicative eojeol that follows them. Second, if a dependency relation can be assigned to two eojeols, the association value is calculated using the estimators described in the previous section.

The association is represented by a pair, ⟨*method, association-value*⟩. If a sentence consists of n eojeols, the GAT used is the $n \times n$ triangular matrix. As mentioned in the previous section, each eojeol has its own syntactic level in Korean, and an eojeol can be combined with either a predicate or a noun. This follows that an eojeol doesn't have dependency relation to the nominal eojeol, whenever it is dependent on the predicative eojeol, *vice versa*. Because the different estimators are applied for the analysis of compound noun and predicate-argument, any collision doesn't take place in the comparison of the association. $Assoc_{NN}$ is used as the estimator for compound noun and $Assoc_{VN}$, for predicate-argument. The GAT is sorted by the association to look up the most probable phrase in the parsing process. Thus, the global association table is implemented by the global association list. The algorithm to generate the GAT is represented in (Figure 4).

for each eojeol ϵ_i $0 <= i <= n-2$

1. for each eojeol ϵ_j $i+1 <= j <= n-1$

if (depend_on (ϵ_i, ϵ_j))

compute $g_i(j) = < method, Assoc(\epsilon_i, \epsilon_j) >$

2. sort $g_i(i+1), \ldots, g_i(n-1)$ and refer it to $G(i)$

Figure 4: The algorithm for making GAT

	0	1	2	3	4	5	6
0	-	(2,0.1)	(1,1/2)	-	(1,1/3)	-	-
1	-	-	(1,1)	-	(1,1/2)	-	-
2	-	-	-	(2,0.11)	-	(1,1/2)	(2,0.02)
3	-	-	-	-	(2,0.15)	-	-
4	-	-	-	-	-	(1,1)	(2,0.52)
5	-	-	-	-	-	-	(1,1)
6	-	-	-	-	-	-	-

Table 1: The global association table(GAT) for the example sentence, ex 3

The following example is represented by (Table 1),

ex 3) (0)*computer* (1)*hwamyon-ui* (2)*gusuk-e* (3)*natana-n* (4)*sutja-ga* (5)*paru-ge* (6)*olaga-t-da.*
(0)computer (1)of screen (2)in the corner (3)appeared (4)the number (5)fast (6)scrolled up
— The number to appear in the corner of computer screen scrolled up fast.

In (Table 1), '-' mark means that two eojeols have no dependency relation. The first element of the pair is the method of the measurement and the second is the association value. The pair, (1, 1/2) in GAT(0,2), indicates that the measure by distance is 1/2. The pair (2, 0.11) in GAT(2,3) means that the association value is 0.11 and estimated with co-occurrence relation. The *method* has the priority for the comparsion of the association. Therefore, (2,0.02) is greater than (1,0.5) because *method* of the first is greater than that of the second. Since the row of the table is sorted for parsing, GAT[2] can be represented in the form of a list of eojeols as follows.

GAT[2] — ⟨3,(2,0.11)⟩ — ⟨5,(2,0.02)⟩ — ⟨6,(1,1/2)⟩

The association list in the above lets the parser know that the eojeol ϵ_2 has the possibility to merge with the eojeol, ϵ_3, ϵ_5 or ϵ_6, and the most probable one is ϵ_3. The function, $max(G(i))$ is defined to return the most probable candidate for the head of the ith eojeol, ϵ_i, in the GAT.

4 Parsing Algorithm

4.1 Parsing algorithm

The parser presented here consists of a stack and a buffer. A two-item lookahead buffer is enough to make decisions in regard to Korean. The grammatical structures lie in the parsing stack and a set of actions are operated on the buffer. Unlike the deterministic parser where the set of rules directs the operation, this system parses by the association value of the GAT.

Since a head follows its complement in Korean, the head of a phrase is the last eojeol of the phrase. A phrase is generated when two eojeols or two phrases are merged. In this case, Head Feature Inheritance takes care of the assignment of the same value as the head feature. Suppose an eojeol, ϵ_1, and an eojeol, ϵ_2, merge and a new phrase P_1 be generated, as shown in (Figure 5). As the head of P_1 is ϵ_2, the parser uses the subscription of ϵ_2 as the index to the GAT, that is, 2.

Basic operations are CREATE, ATTACH, and DROP. However, its operation is conditioned not by rule matching but by the value of the GAT as shown in the following description. The function, $position(max(G(i)))$ returns the sentential position of the most probable candidate for the head of the ith eojeol, ϵ_i.

CREATE If the most probable candidate for the head of the eojeol, ϵ_i, is ϵ_j, that is, $j = position(max(G(i)))$,

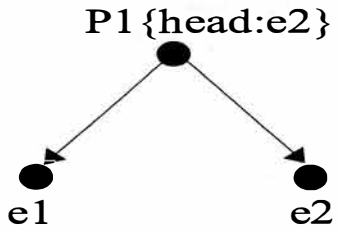


Figure 5: the index to which the parent node refers

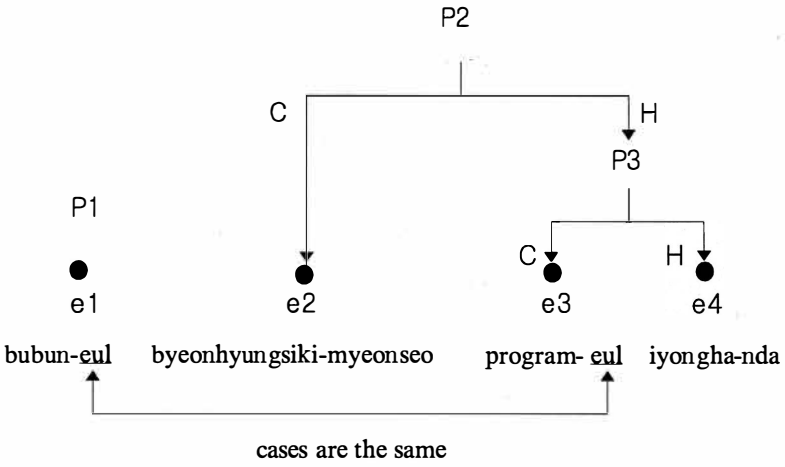


Figure 6: example of failure of the prediction

then merge ϵ_i(or the phrase where the last eojeol is ϵ_i) with ϵ_j(or the phrase where the last eojeol is ϵ_j), and generate a new phrase.

ATTACH If the phrase where the ϵ_j is the last eojeol is not the most probable candidate for the head of the eojeol ϵ_i, that is, $\epsilon_j \neq max(G(i))$ then wait until ϵ_i meets the most probable candidate indicated by the GAT.

DROP DROP operation is accompanied with CREATE operation in our system because the complement precedes the head and thus the top node of the stack must be dropped and checked for dependency immediately after a new node is generated.

The GAT provides the parser with the prediction of the best candidate for the head of the ith eojeol, ϵ_i. This is easy because the GAT is already sorted; however, the expectation is not always correct because the value of the GAT is calculated whenever there is a possible dependency relation between one eojeol and another. That is, the parser constructs the GAT as preparsing and it may happen that the two eojeols or phrases which have the possiblity to have dependency relations cannot be merged in parsing. The violation of the '*one case per clause*' principle, is the case.

ex 4) *bubun-eul*(part/OBJ) *byeonhyeongsiki-myeonseo*(change) *program-eul*(program/OBJ)
iyongha-nda(use).
→ The part being changed, the program is used.

In (Figure 6), ϵ_1 and ϵ_3 are nominal eojeols, and ϵ_2, ϵ_4 are predicative eojeols apiece. The phrase P_1 consists of ϵ_1, and the phrase P_2 consists of three eojeols, ϵ_2, ϵ_3, ϵ_4. Let the most probable candidate, suggested by the GAT, for both ϵ_1 and ϵ_3 be ϵ_4. However, ϵ_1 and ϵ_3 have the same grammatical case because they contain

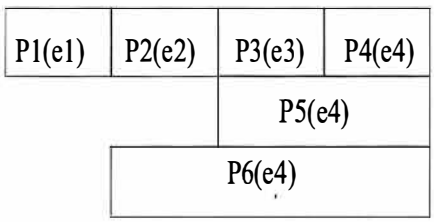


Figure 7: the content of chart and selection for the next candidate

```
For phrases P_i and P_j, let their heads ε_i and ε_j respectively.
if (lookahead = nil and there is one parse tree in the stack)
        return SUCCESS
else if (GAT(●) = NULL)
        return FAIL
else
        if (position(G(i)) = j)
        begin
                if (isunifiable(P_i, P_j) = TRUE)
                        CREATE;
                else ALTER;
        end
        else ATTACH;
```

Figure 8: The parsing algorithm using the GAT

the postpositions marking the same case. The phrase P_1 and the phrase P_2 cannot be merged because of the violation of '*one case per clause*' principle. This means the prediction of the GAT is incorrect, and consequently an analysis with the alternatives is required. If the next candidate is ϵ_2, the grammatical structure in the buffer must be erroneous. The chart presents the phrase suitable for the alternative execution into the buffer. The chart allows the parser to store the partial structure to remove the backtracking. ALTER operation occurs in this case.

ALTER is required if an eojeol ϵ_i cannot be merged with the eojeol ϵ_j which is the prediction of the candidate for the head of ϵ_i.

The operation being executed, the structure in the lookahead buffer is backed up into the chart. When ALTER operation is needed, another candidate taken from the chart, has to be put in the buffer. The next candidate, $C(\epsilon_i)$ is chosen in the following way. Let i be the left-hand position and k the right-hand position of the errorneous prediction in the GAT.

$$C(\epsilon_i) = \text{the phrase that the left-hand position is } i \text{ and the right-hand position is } max(i+1 \leq j \leq k) \text{ in nodes in the chart.}$$

Then, the phrase P_2 including ϵ_2 is the next candidate in (Figure 7). The parsing algorithm with the GAT is described in (Figure 8).

4.2 Parsing

The complexity of making the GAT is $O(N^3 log_2(N))$, where N is the number of eojeols. This is due to the sorted $n \times n$ table. The average complexity of the parser is linear, according to the experiments.

	OP	Stack Top		First Lookahead	
		Constituents	Head	Constituents	Head
1	A	(computer)	computer	(hwamyon-ui)	hwamyon-ui
2	B	(computer hwamyon-ui)	hwamyon-ui	(han)	han
3	A	(han)	han	(gusuk-e)	gusuk-e
4	A	(computer hwamyon-ui)	hwamyon-ui	(han gusuk-e)	gusuk-e
5	A	((computer hwamyon-ui) (han gusuk-e))	gusuk-e	(natana-n)	natana-n
6	A	(((computer hwamyon-ui) (han gusuk-e)) natana-n)	natana-n	(sutja-ga)	sutja-ga
7	B	((((computer hwamyon-ui) (han gusuk-e)) natana-n) sutja-ga)	sutja-ga	(paru-ge)	paru-ge
8	A	(paru-ge)	paru-ge	(olaga-t-da)	olaga-t-da
9	A	((((computer hwamyon-ui) (han gusuk-e)) natana-n) sutja-ga)	sutja-ga	(paru-ge olaga-t-da)	olaga-t-da

Figure 9: an example of analyzing the sentence in (ex 3). (A) Create & Drop operation (B) Attach operation

	Chart Parser first S found	Parser Using GAT
The Total Number of Generated Nonterminals	2,561,613	10,582
The Average Number of Generated Nonterminals	6404	26.5

Table 2: The number of nodes generated by test parsers

(Figure 9) represents the analysis steps of the sentence in (ex 3). The head on the stack top is the complement, and the candidate for the head of it lies in the head part of the buffer. In the seventh row of the figure, the ATTACH operation is executed by the GAT in (Table 1), because the lookahead is not the best candidate for the head of the complement on the stack top. The eojeol, 'sutja-ga', has to wait until it meets its best candidate. A new phrase are created in the row (9). The eojeol, 'olaga-t-da' is the best candidate for the eojeol, 'sutja-ga', which was estimated by the GAT. (Figure 10) represents the parse tree of ex 3). The sentence is written in Korean.

5 Experimental Results

For testing purposes, 400 sentences were randomly extracted from 3 million corpus. First, our parser is compared to the chart parser to show the efficiency of our algorithm. The number of the nodes generated by each parser is represented in (Table 2). Because of ths size of the searching space, the results from the chart parser are calculated whenever the first S is found. The average number of the prediction failure is 0.26 per sentence. That is, The parser has to search for the alternative in the chart once in four sentences. This makes the complexity of the parser a constant. (Figure 11) shows the occurrence of ALTER operation over the number of words. The average number of ALTER is about 0.36 for the sentences with more than 20 words, which means our parser is efficient.

Second, the precision is given in (Table 3). The precision is defined as the ratio of the precise dependency relation between eojeols in parse trees. No label is attached because the final output is the tree that represents the dependency relation among words. Thus, the number of erroneous and correct relations is considered, which can be estimated by the number of crossing brackets(Table 3).

Crossing Brackets number of constituents which violate constituent boundaries with a constituent in the correct parse.

The cause of the incorrect analysis can be largely classified by two reasons. One of the failures is caused by statistical information. We collected the data from 30 million eojeol corpus. The total number of the data is 2

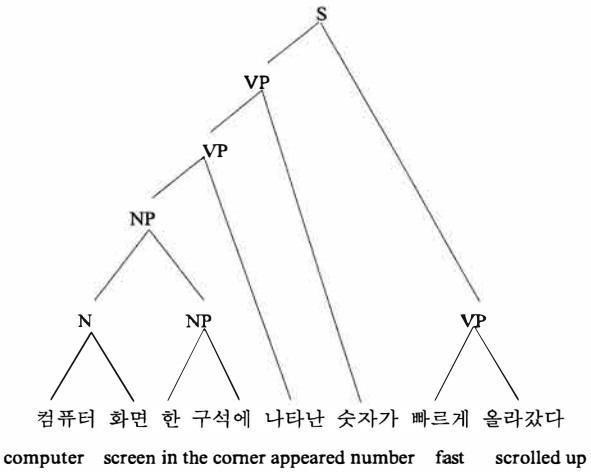


Figure 10: the parse binary tree for ex 3) (in Korean)

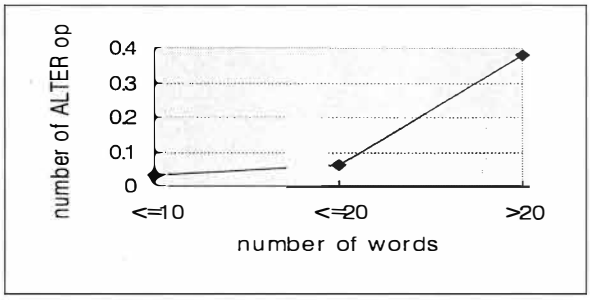


Figure 11: The number of the ALTER operation for words

CBs		0 CBs		$\leq$ 2 CBs	
(a)	(b)	(c)	(d)	(c)	(d)
378	0.95	176	44.0	323	80.8

Table 3: The precision of the parser (a) the number of crossing brackets (b) the average number of crossing brackets per sentence (c) the number of sentences (d) the percentage

million and the average frequency of the co-occurrence data is 2.5. The triples to have frequencies greater than 2 are 400,000. The frequency of most data is 1, which was the cause of the errorneous results. In addition, the association value of adjuncts and subordinate clauses is estimated by distance. The distance estimator was good but not the best. Semantic information such as thesaurus will help reduce the space of parameters.

Second, liguistic information is needed , e.g., such as light verbs or the lexical characteristics of individual words. Our parser is the hybrid system which uses both rules and statistical information. The linguistic research is prerequired for this, even if these can be partially resolved by statistical methods. However, the parser is satisfactory in spite of some erroneous results in that the association value can be computed in various ways, and the parser can be extended using this.

6 Conclusion

We have shown that it is possible to make decision semi-deterministically using the global association table. The GAT is a very effective structure in that it is triangular matrix because Korean is an SOV language and the dependency relations between words is important. It would have to be transformed for parsing English, because phrase structure grammar is needed for parsing English.

There are many possiblities for improvement. The method described for calculating the lexical association in the GAT can be modified in various ways. The GAT and the parser can be extended if the distance measure and the coordinate conjunctive structure are considered.

References

Allen, J. (1995). *Natural Language Understanding.* Benjamin Cummings.

Brill, E. (1993). *A Corpus-Based Approach to Language Learning* Department of Computer and Information Science, University of Pennsylvania.

Briscoe, T., Waegner, N. (1992). *Robust Stochastic Parsing Using the Inside-Outside Algorithm* In *Workshop notes from the AAAI statistically-based NLP Techniques Workshop.*

Charniak, E. (1993). *Statistical Language Learning* MIT Press.

Collins, M. J. (1996). *A New Statistical Parser Based on Bigram Lexical Dependencies* In *Proceedings of 34th Annual Meeting of Association for Computational Linguistics.*

Faisal, K. A. and Kwasny, S. C. (1990). *Design of a Hybrid Deterministic Parser* In *Proceedings of COLING-90.*

Framis, F. R. (1994). *An Experiment on Learning Appropriate Selectional Restrictions from a Parsed Corpus.* In *Proceedings of COLING-94.*

Gazdar G., and Mellish C. (1993). *Natural Language Processing in LISP.* Addison Wesley.

Hindle, D. and Rooth, M. (1993). *Structural Ambiguity and Lexical Relations* Computational Linguistics

Kobayasi Y., Tokunaga T., and Tanaka H. (1994). *Analysis of Japanese Compound Nouns using Collocational Information* In *Proceedings of COLING-94.*

Kurohashi S. and Nagao M. (1994). *A Syntactic Analysis Method of Long Japanese Sentences Based on the Detection of Conjunctive Structures.* Computational Linguistics

Lauer, M. (1995). *Corpus Statistics Meet the Noun Compound: Some Empirical Results* In *Proceedings of 33rd Annual Meeting of Association for Computational Linguistics.*

Marcus, M. (1980). *A Theory of Syntactic Recognition for Natural Language.* Cambridge, MA: MIT Press.

Magerman, D. M. (1995). *Statistical Decision-Tree Models for Parsing.* In *Proceedings of 33rd Annual Meeting of Association for Computational Linguistics.*

Pustejovsky, J., Bergler, S., and Anick, P. (1993). *Lexical Semantic Techniques for Corpus Analysis.* Computational Linguistics

Resnik, P. (1992). *Wordnet and Distributional Analysis: A Class-Based approach to Lexical Discovery* In *Proceedings of AAAI Workshop on Statistical Methods in NLP.*

Tomita, M. (1986). *Efficient Parsing for Natural Language* Boston: Kluwer Academic Publishers

Posters

Constraint-driven Concurrent Parsing Applied to Romanian VP

Liviu Ciortuz *
Dept. of Computer Science
University of Iasi, ROMANIA
Email: ciortuz@infoiasi.ro

Abstract

We show that LP constraints (together with language specific constraints) could be interpreted as meta-rules in (an extended) head-corner parsing algorithm using weakened ID rule schemata from the theory of HPSG [Pollard and Sag, 1994].

We present a refinement of the head-corner algorithm [van Noord, 1996], suitable for grammars containing ID/LP rules, as naturally used for high free-order languages.[1] We show that the ID schemata in the standard framework of the HPSG theory can be "weakened"/decomposed into subcomponents to allow for a fine-tuning of the ID rule application, according to the language specificity, on one hand, and – if desired – for concurrent parsing, on the other hand.

The concurrent-constraint programming (CCP) paradigm [Saraswat, 1993] lead to a powerful model [Smolka, 1995], which allows for the integration of logic (constraint), functional, and object-oriented programming, and also for smoothly passing from sequential to distributed programming, as this model adapts nicely to both these types of implementations [Mehl et al., 1995] [Haridi et al., 1997].

We use the CCP "metaphor" — 'fair interleaving of constraint-based computation processes' — and the logic characterization of attribute–value matrices [Smolka, 1992] to decompose the HPSG ID schemata into quasi-independent components. For instance, the ID schema 3 — eventually combined with the ID schema 5 — decomposes into the following "weakened" ID schemata:[2]

0.	Head – Argument Composition	(H–AC)
1.	Head – ACC Specifier	(H–S1)
1′.	Head – Direct Complement	(H–C1)
2.	Head – DAT Specifier	(H–S2)
2′.	Head – Indirect Complement	(H–C2)
3.	Head – Subject	(H–Subj)
4.	Head – Modifier	(H–Mod)

Such a decomposition is perfectly adapted for (LP) constraint-driven (concurrent) parsing of partially free-ordered phrases. For instance, the Romanian VP is characterized by

— *free* order up to the SUBJECT + COMPLEMENTS component list

— *strict* order inside the SPECIFIERS bloc (finite verb form + clitics) as given by LP constraints like:

$$(\text{Nu}) < \text{CL}_{\text{DAT}} < \text{CL}_{\text{ACC},\sim 3sf} < \text{Avea}_{\text{AUX}+} < \text{Participle} < \text{CL}_{\text{ACC, 3sf}}$$

Decomposing/weakening the HPSG ID schemata would not be needed in case *i.* we adopt for Romanian the fully lexicalized account of clitics introduced for Italian [Monachesi, 1995] [Monachesi, 1997] and French [Sag and Miller, 1997], or *ii.* one is not interested in exploring concurrent/distributed constraint parsing. Concerning the first point, we argued [Ciortuz, 1997] that in our opinion this is a hardly tenable position in case

*The author is engaged now with the LT Lab at DFKI – Saarbrücken, Germany. Email: ciortuz@dfki.de.

[1] This is the case of many human languages like Italian, Romanian, Slavic languages, etc.

[2] We restrict ourselves to what's needed to parse the Romanian transitive verbal complex. Recovering the general case is a straightforward task.

of highly inflected languages (like Romanian), because it leads to exploding with 1–2 orders of magnitude the size of the lexicon, and — independent of this argument — because clitic combination is not so arbitrary (at least for Romanian) as supposed by the papers previously mentioned, so it lacks encoding a certain generality.

Otherwise, i.e. assuming that clitics encode their (conditional) combination capacity as verb's specifiers (cf. [Ciortuz, 1997]), and exploring (the possible benefits of) concurrent parsing, one could see the application of weakened ID schemata as driven by the LP constraints and, possibly, some constraints specific to the language like (CP.C) — Clitic Pronouns Constraint, and (PA.C) — *Pe*-Accusative Constraint in Romanian.

Example for Romanian VP decomposition using the above weakened ID schemata:
the sentence

Cheia, i-am dat-o ieri lui Ion. = *The key, I gave it to John yesterday.*

is parsed as:

	Cheia,		i-		am dat		-o		ieri	lui Ion.
	•				H+AC					
(LP) →	•	<	HS2	<		<	HS1	<		
(CP.C) →	• HC1									HC2
	•								HMod	

Conclusion

We exploit the LP constraints and the possibility of (logic) "atomization" of HPSG ID schemata in order to
— account for the (partial) free-orderness inside the verbal complex in Romanian and
— explore the benefit of (as much as possible) concurrency in the VP parsing.

This approach aims to smoothly adapt HPSG head-driven parsing design primarily for a strict-order language like English to a partially free-order language like Romanian. It was elaborated while trying to provide for a clean explanation of clitics' behavior in Romanian.

References

[Ciortuz, 1997] Ciortuz, L. (1997). Mastering clitics' (dis)order. In *Informal Proceedings of The GL&GE Workshop*, Tuşnad, Romania.

[Haridi et al., 1997] Haridi, S., Roy, P. V., and Smolka, G. (July 1997). An overview of the design of Distributed Oz. In *Proceedings of PASCO '97*, Maui, Hawaii.

[Mehl et al., 1995] Mehl, M., Scheidhauer, R., and Schulte, C. (1995). An Abstract Machine for Oz. In Hermenegildo, M. and Swierstra, S. D., editors, *Programming Languages: Implementations, Logics and Programs, 7th International Symposium, PLILP'95*, Lecture Notes in Computer Science, vol. 982, pages 151–168, Utrecht, The Netherlands. Springer-Verlag.

[Monachesi, 1995] Monachesi, P. (1995). *A grammar of Italian clitics.* PhD thesis, Tilburg University. ITK Dissertation Series 1995-3 and TILDIL Dissertation Series 1995-3.

[Monachesi, 1997] Monachesi, P. (1997). Decomposing Italian clitics. In *HPSG for Romance*, Stanford. Center for the Study of Language and Information.

[Pollard and Sag, 1994] Pollard, C. and Sag, I. (1994). *Head-driven Phrase Structure Grammar.* Center for the Study of Language and Information, Stanford.

[Sag and Miller, 1997] Sag, I. and Miller, P. (1997). French clitic movement without clitics or movement. *To appear.*

[Saraswat, 1993] Saraswat, V. (1993). *Concurrent constraint programming.* The MIT Press, Cambridge, MA.

[Smolka, 1992] Smolka, G. (1992). Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87.

[Smolka, 1995] Smolka, G. (1995). The Oz programming model. In van Leeuwen, J., editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin.

[van Noord, 1996] van Noord, G. (1996). An efficient Prolog implementation of the head-corner parser.

ROBUSTNESS and EFFICIENCY in AGFL

Erik Oltmans
Dept. of Computer Science
University of Twente
Enschede, The Netherlands
oltmans@cs.utwente.nl

Caspar Derksen, Kees Koster
Dept. of Computer Science
University of Nijmegen
Nijmegen, The Netherlands
caspard@cs.kun.nl, kees@cs.kun.nl

Robust Parsing

The Condorcet project is an information retrieval project carried out at the Knowledge-Based Systems Group at the University of Twente, The Netherlands, and funded by the Dutch Technology Foundation (STW). The objective of Condorcet is to build a prototype information retrieval system that will ultimately be able to handle titles + abstracts of 30,000 documents. The system is mostly concerned with *indexing* of documents within two specific domains: mechanical properties of engineering ceramics and epilepsy. For an elaborate discussion of the system, the reader is referred to the Condorcet Annual Reports, available through the Condorcet home page: `http://wwwis.cs.utwente.nl:8080/kbs/condorcet/`. Assignment of indexing concepts to documents is done by means of a syntactic parser (CONSTRUCTER) and a semantic analyzer. As the documents contain numerous misspelled words, ill-formed constructions and new words, not all input can be parsed according to the underlying grammar. Since achieving robustness by constantly adding rules to the grammar is laborious and may be even impossible, an extra robustness device is needed in order to parse all documents. However, approaches that can be found in the literature such as [1], imply in most cases adjustments of the underlying parsing strategy: problematic fragments are skipped or completely deleted. Within Condorcet however, a knowledge-based approach is preferred, as handling ill-formed sentences with robust grammars rules would allow for more flexibility and expandability. By employing a knowledge-based approach, it is possible to develop a solution to the robustness problem that is not based on ad hoc approaches, but that acknowledges the linguistic nature of the problem. It also allows for more control over the quality of the output: in cases of an unsatisfactory result, the robust rules can be easily adapted so as to tune the ultimate parser. Robustness is thus accomplished by the linguist instead of the programmer, and by means of writing specific grammar rules rather than writing computer programs. The AGFL formalism, developed at the University of Nijmegen (`http://www.cs.kun.nl/agfl`) is used because it allows for elegantly expressing a knowledge-based approach to robustness and because of its efficient implementation.

Affix Grammars over a Finite Lattice

Affix grammars over finite lattices (AGFLs) are a restricted form of affix grammars in which Context Free production rules are extended with affixes (or features) for expressing agreement between parts of speech, [2]. Like in PROLOG with DCGs, these are passed as parameters to the rules of the grammar. Each affix expression at a parameter position denotes a set of features associated with the nonterminal in a certain context. In rewriting, all occurrences of a certain affix in a rule obtain the same value, being any subset of the (finite) domain of the affix, but not the empty set. *Feature ambiguity* is distinguished from *structural ambiguity* by decorating parse trees with *sets* of affix values, rather than with single valued affixes. As a result, multiple parses with the same structure are combined. Although Tomita or Earley are the preferred parsing algorithms for CF grammars, in parsing according to a grammar with features, the (exponential) determination of features may well overwhelm the parsing process, so that the efficiency of the parsing by itself is not the main issue. Therefore AGFL is implemented using Recursive Backup Parsing (RBP), a generalization of Recursive Descent Parsing to ambiguous grammars. The main drawback of RBP is that parsers exhibit exponential behavior, but by applying the optimizations described below, large-scale parsers show quasi-linear behavior for sentences of reasonable length; their exponential character becomes apparent for only a fraction of the sentences. Furthermore, top-down backtrack parsers are easy to extend with affixes, allow top-down filtering of values, [3], and parser

generation for RBP is a straightforward transcription from the grammar. Fast parser generation makes AGFL suitable for development and prototyping and finally, RB parsers are small and use memory efficiently.

Factoring out common left-factors of alternatives for the same nonterminal, possibly in combination with repeated unfolding of nonterminals, can be used for improving the time complexity of RBP, as it eliminates duplicate parsing effort. Top-down RBP by itself is not capable of dealing with left-recursion, but it is possible to remove left-recursion from a grammar, e.g. by *goal-corner transformation.* Applying look-ahead is another classical technique for improving the run-time behavior of parsers, although the time complexity of the parsing algorithm remains within the same complexity class.

Memoization is a technique for optimizing repeated computations of a function by remembering a table of values for certain arguments. However, since tuples of parameters do not have a total ordering, it is hard to search a table in which calls of parameterized nonterminals are memoized. Therefore, *negative memoization* is used, preventing the repetition of unsuccessful computations. In the mapping $memo : pos \times nonterminal \rightarrow bool \times bool$ the first boolean (*known*) indicates whether this nonterminal has been tried at this position, and the second boolean (*blocked*) indicates whether the nonterminal will fail at this position, and should be blocked. In applying this technique, each call of a nonterminal at a position where it has not been called before is first *generalized* to a call with all parameters generalized to their full domains; if the rule fails it is known that the rule will fail for all possible combinations of affixes and this fact is recorded for future use. If the call succeeds, the parameters are restricted to their original values. This optimization improves the efficiency of RBP dramatically.

Robust Parsing in Practice

For achieving knowledge-based robustness without ambiguity, the most important features in AGFL are the semi-predicate `PENALTY` and the *commit operator.* They can be used to indicate a preference of clear syntactic forms over doubtful ones. By using these features, alternatives can be ordered in such a way that *graceful degradation* is ensured, providing as much of an accurate analysis as possible: instead of failing in returning a highly structured analysis, a reliable shallow analysis is returned. This is accomplished by means of an ordered set of alternatives, in which the level of underspecification in order to describe problematic parts of the input is decreasing. If the parsing process is aborted, or the parser reaches a pre-defined time limit, the current parse (which is the best parse for that particular moment) is returned. However, if the input is not problematic, only the parse according to the last alternative will be returned, because of penalties in previous alternatives. If the input is extremely complex and the time-limit is exceeded, minimal information will at least be returned. This strategy yields a rule set that is divided into a *core grammar* and a *peripheral grammar* and it allows for full control over the quality of the output, depending on the grammar writer's intentions. The linguist only needs to express ideas with respect to robustness; computational complexity, efficiency or speed are irrelevant from the grammar writer's point of view.

Within the Condorcet parser, a robustness device has been developed according to this knowledge-based strategy. This has resulted in a parser that is capable of analyzing 100% of its input, providing information on the syntactic subject, the finite verb and all noun phrases and prepositional phrases. For now, it seems that this is useful enough but if more information were needed, the rules could simply be adjusted. CONSTRUCTER is a parser for English, generated from an AGFL grammar that consists of 68 rewrite rules extended with 14 robust rules. We report on a test involving a corpus of 84 documents (30 documents on epilepsy, 30 documents on ceramic materials and a randomly selected corpus of 24 in order to test the parser on unseen material). Most sentences in the corpus are very complex: the average sentence length is 23.8 words per sentence. 60 documents were parsed in 11 seconds, which is equal to parsing 40 sentences per second or 950 words per second (on a SUN/SPARC station). The shortest parse took 0.01 seconds, the longest 0.46 seconds. 61% of the sentences were parsed according to the core grammar and 39% were parsed robustly. The parser needed relatively more robustness (37% vs. 63%) when analyzing unseen texts, which is not surprising.

References

[1] T. Strzalkowski. Robust Text Processing in Automated Information Retrieval. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing.* Association for Computational Linguistics, 1994.

[2] C.H.A. Koster. Affix Grammars for Natural Languages. In H. Alblas and B. Melichar, editors, *Attribute Grammars, Applications and Systems*, volume 545 of *Lecture Notes in Computer Science*, pages 469–484. Springer-Verlag, Berlin, Germany, 1992.

[3] M.J. Nederhof. *Linguistic Parsing and Program Transformations.* PhD thesis, University of Nijmegen, The Netherlands, 1994.

LANGUAGE ANALYSIS IN SCHISMA

Danny Lie, Joris Hulstijn,
Hugo ter Doest, Anton Nijholt*

1 Introduction

SCHISMA is a research project that is concerned with the development of a natural language accessible theatre information and booking system. In this project two research approaches can be distinguished. One approach is devoted to theoretical research in the areas syntax, semantics and pragmatics.Research on syntax has been conducted on a unifying parsing approach [7], left and head corner grammars [6], on stochastic context-free grammars [1] and on unification grammar parsing [2][4]. Research on semantics and pragmatics has been conducted on dialogue act classification [3] and from a logical point of view [5].

The other approach is more practically oriented in that it is more focused on the realisation of a fully functional prototype of the SCHISMA system. To this purpose, in the past, the interface development system Natural Language™ has been used for implementing a SCHISMA prototype, a Wizard of Oz environment has been developed (with which a corpus of dialogues has been collected), an attempt has been made to develop a SCHISMA prototype for the WWW, and a prototype has been developed for education purposes[1]. Our most recent achievement is a fully functional SCHISMA system (implemented in Java) based on a context-sensitive string rewrite mechanism. Although the system is primitive in nature and not necessarily built on linguistic principles (rather on intuitions), we believe its performance is such that the majority of users will be satisfied. Users will adapt to the system rather than reject it because of poor performance (when compared to human-human dialogues). In the development of a spoken telephone service for the Dutch Public Transport Service a similar position is taken.

In this short paper we will give an overview of the two approaches distinguished here as far as parsing is concerned. Section 2 presents work on classical unification-based parsing, section 3 discusses the rewrite and understand approach in some detail. Section 4 considers future developments and concludes the paper.

2 Unification-based Approach

In [2] a head corner parser for typed unification grammars is designed and implemented in C++; for description of the lexicon and the grammer a specialised specification language TFS is developed. It is argued that typed unification grammars and especially the newly developed specification language are convenient formalisms for describing natural language use in dialogue systems. The system comprises a compiler that compiles TFS specifications into C++ and some specification-independent libraries (input/output, parsing and unification algorithms). After a TFS specification is developed and compiled into C++, the libraries and the specification-dependent parts of the system can be linked together.

Another, more sophisticated system for unification-based parsing is currently under development.[2] Type hierarchies, disjunctive feature structures and productions all are represented by means of DAGs. Other advantages of the system include the optional weighting of disjunctions with probabilities, flexible behaviour in case of conflicts and an incremental type hierarchy that may grow during parsing depending on type conflicts leading to new types.

*Parlevink Language Engineering, Dept. of Computer Science, University of Twente, PO Box 217 7500 AE, Enschede, The Netherlands, (lie | joris | terdoest | anijholt)@cs.utwente.nl

[1]By Joris Hulstijn; it can be downloaded from wwwseti.cs.utwente.nl/~joris/

[2]By Hugo ter Doest; a prototype can be downloaded from wwwseti.cs.utwente.nl/~terdoest/tfs/tfs.tar.gz

3 The *Rewrite and Understand* Approach

As mentioned in the introduction, there have been several attempts to design a SCHISMA dialogue system with the aim to end up with a system that is certainly not perfect but that can nevertheless be accessed by users willing to adapt themselves to the system. Our most recent achievement is a system that consist of two subsequent processes: a *rewrite* process in which natural language utterances are mapped via a sequence of context-sensitive string-to-string transformations onto some semantical normal form, and an *understanding* process, in which the interpretation of the semantic form is made dependent on the current dialogue state. We concentrate on the rewrite process here.

Two basic operations underly the rewrite process: deletion of semantically irrelevant words, and substitution of words by standard synonymous words or expressions. If we abstract from syntactic sugar of the specification language, deletions and substitutions are defined by context-sensitive rewrite rules of the form $x_1 \dots x_n \rightarrow y_0 x_{k_1} y_1 \dots y_{m-1} x_{k_m} y_m$ where $m \in 1 \dots n$ and $k_i \in 1 \dots n$ for all $i \in 1 \dots m$ where $x_1 \dots x_n$ are regular expresions[3] and $y_1 \dots y_m$ are arbitrary strings. So rewrite rules are potentially context-sensitive. A rewrite rule is applicable to a sentence if its left hand side matches a substring of the sentence; application of the rewrite rule means that the matched part of the sentence is rewritten, the surrounding parts are left as they are. A rewrite specification consists of a sequence of rewrite rules. Given a rewrite specification and a sentence s, a complete rewrite of s means that the sequence of rewrite rules is run through in the order specified applying each rule applicable in the sense described: $s \overset{r_{i_1}}{\rightarrow} s_1 \overset{r_{i_2}}{\rightarrow} \dots \overset{r_{i_n}}{\rightarrow} s_n$.

The complete rewrite process consists of two phases: a *global* phase and a *local* phase, respectively. The global phase is the same for all sentences; after the global rewrite a keyword filter decides on what local grammar(s) to apply to the sentence. If more than one local grammar applies, all of them are executed; a score accompaning the rewritten strings helps the understanding process decide on what interpretation to continue with. We developed one global grammar and 20 local grammars by closely examining the aforementioned Wizard of Oz corpus. Each of the local grammars cover a part of the semantic functions sentences may have in the SCHISMA domain. Each local grammar finally rewrites its input into a list of feature value pairs accompanied by the name of the action the system should take. The understanding process then interpretes the list in the context of the dialogue.

4 Future Work and Conclusion

In the near feature a combination of the two approaches is foreseen: the rewrite and understand approach will be adapted such that it can be applied as a preprocessor that rearranges input such that difficult problems like discontinuity and embedded sentences no longer burden the unification-based parsing process. Also the rewrite parser will be applied as a backup mechanism in case conventional parsing fails.

References

[1] Rieks op den Akker and Hugo ter Doest. Weakly restricted stochastic grammars. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 927–934, 1994.

[2] Rieks op den Akker, Hugo ter Doest, Mark Moll, and Anton Nijholt. Parsing in dialogue systems using typed feature structures. In *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague/Karlovy Vary, Czech Republic, 1995.

[3] Toine Andernach. 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*, pages 98–109, 1996.

[4] Hugo ter Doest. *Robustness and Efficiency in Unification-based Parsing Methods.* PhD thesis, University of Twente, Enschede, The Netherlands. To appear.

[5] Joris Hulstijn. Structured information states - raising and resolving issues. In *Proceedings Munich Workshop on Formal Semantics and Pragmatics of Dialogue*, Munich, Germany, 1997. University of Munich.

[6] Klaas Sikkel and Rieks op den Akker. Predictive head-corner chart parsing. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 267–275, Tilburg (The Netherlands), Durbuy (Belgium), 1993.

[7] Klaas Sikkel and Anton Nijholt. Parsing of context-free languages. In *Handbook of formal languages*, volume II Linear Modeling: Background and Application, pages 61–100. Springer, Berlin; Heidelberg; New York, 1997.

[3] Regular expressions x_i may be optional, one of a set of alternatives, required at the beginning or the end of the sentence, a *joker* * matching any word, or a *wildcard* ** matching any sequence of words.

Reducing the Complexity of Parsing by a Method of Decomposition

Caroline Lyon and **Bob Dickerson***
School of Information Sciences, University of Hertfordshire,UK

A method of automatically locating the subject of a sentence has been developed, and we may be able to take advantage of this to reduce the complexity of parsing English text. Declarative sentences can almost always be segmented into three concatenated sections: *pre-subject - subject - predicate.* The pre-subject segment may be empty; for imperative sentences the subject section is empty. Other constituents, such as clauses, phrases, noun groups, are contained within these segments, but do not normally cross the boundaries between them. Though a constituent in one section may have dependent links to elements in other sections, such as agreement between the head of the subject and the main verb, once the three sections have been located, they can then be partially processed separately, in parallel.

The tripartite segmentation can be produced automatically, using the ALPINE parser. This is a hybrid processor in which neural networks operate within a rule based framework. Readers are invited to access a prototype via telnet, to use on their own text (for details contact the authors).

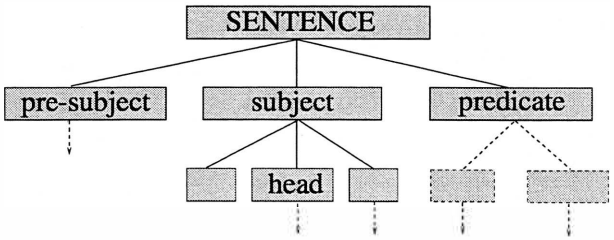


Figure 1: Decomposition of the sentence into syntactic constituents

A sentence is represented like this:

If a cooler is fitted to the gearbox, [the pipe connections of the cooler] must be regularly checked for corrosion.

ALPINE has been trained and tested on text of technical manuals ("Perkins"), and also run on other technical manuals ("Dynix" and "Trados"), see Table1. The first parsing step applies the neural processor to tag sentences and place the hypertags marking the subject. For more information see [Lyon and Frank, 1997] and other papers at `ftp://www.cs.herts.ac.uk/pub/caroline`.

This tripartite decomposition of sentences is supported by an argument from information theory [Lyon and Brown, 1997], derived from Shannon's original work with letter sequences [Shannon, 1951]. His ideas can be extended to other linguistic entities. Shannon showed that the entropy of letter sequences declines, the degree of predictability is increased, as information from more adjacent letters is taken into account.

The entropy can also be reduced if an extra character representing a space between words is introduced, producing a 27 letter alphabet. "A word is a cohesive group of letters with strong internal statistical influences" (Shannon), and the introduction of the space captures some of the structure of the letter sequence. A similar technique can be applied to text mapped onto part-of-speech tags.

* email: {C.M.Lyon,R.G.Dickerson}@herts.ac.uk Tel: +44 (0)1707 284266/4355

Text	Number of sentences	% tags + hypertags correct	% hypertags correct
Perkins	42	92.9	100
	59	91.4	100
	63	89.2	100
	67	84.4	95.5
Dynix	114	-	92.1
Trados	134	-	85.1

Table 1: Results on declarative sentences in technical manuals. From test data in Perkins corpus (2% sentences omitted) used for development, and from other texts.

Now, language can be represented at a primary level as a regular grammar, and we can apply Shannon's analysis to tag sequences. However, this is an inadequate representation: the patterns of tag sequences may be disrupted at clause and phrase boundaries. "Unlikely" tag combinations such as *noun - pronoun* and *verb - auxiliary verb* may occur at constituent boundaries in sentences like *The shirt he wants is in the wash.*

In a similar manner to the insertion of a space between words, the embedded clause is delimited by inserting hypertags, like virtual punctuation marks. The sentence is represented as

The shirt [he wants] is in the wash.

The part-of-speech tags have relationships with adjacent hypertags in the same way that they do with each other. Using this representation, one level of embedding has been modelled. We can thus represent sequences higher in the Chomsky hierarchy than regular grammars, though not fully context free.

Since the boundaries of clauses and phrases often coincide with the boundary of the subject, we expect that the insertion of hypertags to demarcate the subject will lower the entropy. If their insertion has captured some of the structure of language the bipos and tripos entropy should be reduced. This is indeed what was found on the data from the Perkins corpus. See Table 2. A tagset of 32 was used, including the hypertags.

	H_0	H_1	H_2	H_3
plain tag string	5.0	3.962	2.659	2.132
tags + subject markers	5.0	4.135	2.472	1.997

Table 2: Entropy measures for text with and without subject boundary markers. H_n is the entropy when information is taken from n adjacent tags.

This type of system could be used as a pre-processor to facilitate the processing of longer sentences by other NLP methods. Though there are arbitrary limits on the length of constituents that can be processed by ALPINE (15 words in the pre-subject, 12 words in the subject), these bounds are comparatively wide, and of course we plan to extend them. In the Trados data 9 sentences out of 134 fell outside these limits.

One of the advantages of this approach to parsing is that it lends itself to the extraction of predicate/argument structure. After the subject has been located the main verb will be found in the predicate, and then the object or complement. With the head of the subject found, we then have the raw material from which we can begin to extract the predicate/argument structure.

References

[Lyon and Brown, 1997] C Lyon and S Brown. Evaluating Parsing Schemes with Entropy Indicators. In *MOL5, 5th Meeting on the Mathematics of Language*, August 1997.

[Lyon and Frank, 1997] C Lyon and R Frank. Using Single Layer Networks for Discrete, Sequential Data: an Example from Natural Language Processing. *Neural Computing Applications*, 5 (4), 1997. To appear.

[Shannon, 1951] C E Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, pages 50–64, 1951.

Formal tools for separating syntactically correct and incorrect structures *

Martin Plátek, Vladislav Kuboň, Tomáš Holan

platek@kki.ms.mff.cuni.cz, vk@ufal.ms.mff.cuni.cz, holan@ksvi.ms.mff.cuni.cz

Faculty of Mathematics and Physics

Charles University, Prague, Czech Republic

Abstract

In this paper we introduce a class of formal grammars with special measures capable to describe typical syntactic inconsistencies in free word order languages. By means of these measures it is possible to characterize more precisely the problems connected with the task of building a robust parser or a grammar checker of Czech.

1 RFODG

This paper is actually an abstract of [3]. The main topic of [3] is the introduction of a formalism called *RFODG (Robust Free-Order Dependency Grammar)*. This formalism was developed as a tool for the description of syntactically ill-formed sentences of a language with high degree of word-order freedom. The *RFODG* serves for the description of surface syntax; it provides the base of the parsing with subsequent evaluation of *syntactic inconsistencies* (violation of a syntactic rule) and localization of *errors* (message).

Every symbol of RFODG belongs to *terminals* or other symbols (*nonterminals*), to *deletable* or *nondeletable* symbols and to *positive* or *negative* symbols. The terminals of RFODG are the lexical categories of the morpholexical analysis, next pairs of sets serve for the classification and localization of syntactic inconsistencies.

[3] contains the definition of a *DR-tree* by *G*. The *DR-tree* should map the essential part of history of deleting dependent symbols, and rewriting dominant symbols, performed by the rules applied while (bottom to up) analysis.

DR-tree is similar to a standard derivation tree of CFG ff but a node (constituent) of a *DR-tree* may cover a discontinuous subsequence of the input sentence, the terminals can be used to create any type of nodes, not only the leaves and each node has a fixed horizontal position, which is shared by exactly one leaf. This property of *DR-trees* is used to localize syntactic inconsistencies in analyzed sentences.

We say that a sentence w is *positively parsed*, if there exists *DR-tree* for w containing positive symbols only. Sentence w is *robustly parsed*, if it is parsed, but not positively parsed.

The properties of *RFODG* allow to define three complexity measures, namely *local* and *global number of gaps*, the *size of gaps* and the *degree of robustnes*.

First three measures are defined on *DR-trees* by means of a notion of *coverage*. The coverage of a node T may be intuitively described as a set of horizontal indices (the index of a terminal node is in fact its position in the sentence counted from left to right) of terminal nodes, which are derived from the node T. If the *DR-tree* is projective, the *local number of gaps*, *global number of gaps* and of course also the *sizes of gaps* are all equal to zero. Nonprojective *DR-tree* contains at least one gap.

The degree of robustness equals the number of negative symbols in a *DR-tree*.

The processing of the main phase of the system, so called grammar-checking analysis, is carried in three phases. The first phase checks whether the set of projective positively parsed trees is empty. If it is, then the second phase using and extended grammar containing also error anticipating rules and rules with relaxed constraints tests if exists a positively parsed nonprojective tree or a negatively parsed projective tree (with some limitations formulated via mentioned measures). The third phase checks the existence of a negatively parsed

*This work is supported by the Grant Agency of the Czech Republic, Grant-No. 201/96/0195 and by the RSS/HESP grant No.85/1995.

nonprojective trees. If any of the three phases finds a nonempty set of trees, the grammar checking ends and the evaluation module is called.

2 Evaluation of parses

The negative symbols can be classified into a number of groups (see [3]), the most interesting of which are the negative nonterminals signalizing an agreement inconsistency or some other type of inconsistency which can be corrected by some morphological changes of the word forms in the analyzed sentence. We denote such nonterminals as *mf-symbols*.

It is clear that there is a path leading up to the node which is assigned a pertaining *mf-symbol* from all nodes which contribute to the inconsistency signaled in that node. In order to be able to locate the source of this inconsistency we divide the set of rules into two subsets: the rules which transfer (these rules are called *mf-sensitive* or do not transfer the morphemic information *mf-insensitive*.

The report ([3]) introduces an exact definition of a notion of an *mf-component*. For the purpose of this paper it is possible to describe this notion informally as a smallest subtree of a *DR-tree*, which contains a particular *mf-symbol* and from which leads an *mf-insensitive* edge (such an edge was created by the application of an *mf-insensitive* rule).

The evaluation phase is part of the system following the grammar-checking analysis. If the grammar-checking analysis ends in the first phase, the evaluation module is not invoked because the string being analyzed is considered correct.

If the grammar-checking analysis ends after the second phase, the evaluation receives the sets of trees $TR2$. In this case the first task of the evaluation module is to check whether the non-projective positive trees may be also considered as an expression of an error in agreement in the projective readings of the analyzed string w. From the set $TR2$ the evaluation selects the subset of those trees which do not contain other negative symbols than mf-symbols. This set is denoted as *TR-mf*. The dependency trees corresponding to *TR-mf* with marked *mf-components* will be enumerated (by the contraction of *DR-trees*). These trees contain possible agreement errors.

In case *TR-mf* is empty, the evaluation will not return any warning which means that the string being analyzed is considered to be correct (and nonprojective).

If the grammar-checking analysis ends in its third phase, it issues the set of trees $TR3$ for the evaluation. If the set $TR3$ is not empty, the evaluation returns dependency trees containing negative nodes in order depending on number of negative nodes and number of mf-components.

3 Conclusion

In [3] we have summarized the current stage of our research concerning grammar-checking of Czech language. Its main result is the development of the formalism capable of localization and classification of syntactic inconsistencies in languages with a high degree of word order freedom. It also provides a base for future exact research in the field of grammar checking and robust parsing and opens a number of questions requiring further investigations.

References

[1] A.V. Gladkij: Formalnyje gramatiki i jazyki, Iz.: NAUKA, Moskva, 1973

[2] D.T.Huynh: Commutative Grammars: The complexity of Uniform word Problems, Information and Control 57, 1983, pp. 21-39

[3] T.Holan,V.Kubon, M.Platek: Formal tools for separating syntactically correct and incorrect structures. Technical report MFF UK, in press, Charles University Prague

Parsers Optimization for Wide-coverage Unification-Based Grammars using the Restriction Technique

Nora La Serna Arantxa Díaz
Department of Computer Languages and Systems, University of the Basque Country
p.k. 649, 20080 Donostia, Spain. (jiblapan@si.ehu.es)

Horacio Rodríguez
Department of Computer Languages and Systems, Universitat Politécnica de Catalunya
Pau Gargallo 5, 08028 Barcelona, Spain. (horacio@lsi.upc.es)

Abstract

This article describes the methodology we have followed in order to improve the efficiency of a parsing algorithm for wide-coverage unification-based grammars. The technique used is the restriction technique (Shieber 85), which has been recognized as an important operation to obtain efficient parsers for unification-based grammars. The main objective of the research is how to choose appropriate restrictors for using the restriction technique. We have developed a statistical model for selecting restrictors. Several experiments have been done in order to characterise those restrictors.

1. Background and Tools used

The use of linguistic material associated to the features of unification-based grammars, for guiding the parsing process may give a problem, as Shieber pointed out in (Shieber 85). This is, certain unification grammars present an infinite nonterminal domain, leading to inefficiency or even non-termination of the algorithms if standard methods of parsing are used. At the same time, Shieber proposed a solution to this problem with the *restriction* technique. However, there is an open question using restriction. It is not clear how to choose an appropriate *restrictor* (subset of the feature structures owned by the complex categories of the grammar), which assures to obtain the greatest efficiency. An inadequate choice of restrictors affects the efficiency of parsing algorithm. The research presented in this article deals with this problem i.e, what we present here is a methodology for choosing adequate restrictors with wide-coverage unification-based grammars.

In our study, a Patr-II grammar has been generated from the object grammar of the system ANLT (Alvey Natural Language Tools; Grover et al. 93 & Carroll 93). The Alvey grammar defines a wide-coverage of syntactic grammatical constructions of English; only 350 rules and 5008 entries of the grammar and lexicon respectively have been converted. We have also used the UNICORN parser developed by Gerdemann and Hinrichs (Gerdemann 91). It is an Earley-style chart-parser for unification grammars that incorporates the restriction technique.

Restriction is defined as follows: Let R be a *restrictor*, and D a dag of a unification grammar. RD is the *restricted dag* for D relative to R if RD subsumes D, and for every path p in RD, there exists a path q in R such that p is a prefix of q.

2. Methodology and Experiments Developed for Selecting Restrictors

The method we have used for selecting adequate restrictors with the wide-coverage grammar is based on the criterion of instantiation of the features. So, a statistical model for estimating the probabilities of being instantiated of the different features of the grammar along a parsing process was defined. Two important reasons justify the model: 1) each rule of the grammar has different application probability; 2) instantiation can be achieved by appearing explicitly

instantiated in the rule or by percolating through mother or sister categories. Formally, we present the model as follows:

$$Pinst\ (C_i, R_j) = \sum_k \Big(P(r_k) * (instantiated\ (C_i, R_j, r_k) + \Big(P(Cm_k, R_j) + \sum_p P(Ch_k, R_j) \Big) * un\_instantiated\ (C_i, R_j, r_k) \Big)$$

In the formula, Pinst (C_i, R_j) is the probability that the feature j, of the category i, appears instantiated; r_k is the kth rule; Cm is the mother category; and Ch is the brother category. For each rule of the grammar where the feature j, of the category i is defined, the following can be done: 1) If the feature mentioned *is instantiated*, the matrix *instantiated* (C_i, R_j, r_k) will contain the value one, and zero otherwise; 2) If the feature mentioned *is uninstantiated*, the matrix *no_instantiated* (C_i, R_j, r_k) will contain the value one, and zero otherwise. In this step, the value of the feature can be inherited from the mother or brother categories. 3) If the feature analyzed is uninstantiated, and the value of the feature can not be inherited, then the Pinst (C_i, R_j) is zero in the rule. Finally, the model generates a regular linear equations system, where the variables are the probabilities of being instantiated.

In order to prove our criterion for choosing appropriate restrictors we have performed several types of experiments (La Serna 96). The aim of them was to observe which of the proposed restrictors are adequate. The experiments consist of the analysis of a set of 30 phrases (selected randomly from the corpus) with different restrictors, basically change in length and grades of instantiation (high, intermediate, and uninstantiated). The selected measure of evaluation is the number of predictive states, so the *apropriate restrictors* are those which have the lowest number of states.

The experiments have been planned with two classes of restrictors: *static* and *dynamic*. In the first class, the restrictor is the same for all the categories of the grammar, as established in the original definition of restriction. In the second class, we have proposed that the restrictor can be different for each category of the grammar, because certain features make up adequate restrictors in some categories, but are not good candidates for others.

From the analysis of the results of all the performed experiments, we can point out the following: 1) Features that have established *appropriate restrictors*, which are instantiated in all the grammar rules. For restrictors with uninstantiated features in at least some rule, the results have not been better, in general. 2) Any combination of features of appropriate restrictors form again appropriate restrictors; however, any combination of non-appropiate restrictors makes non-efficiency process parsing. 3) In the experiments with static restrictors, the features established as appropriate restrictors have been obtained from the intersection of the best features of each category in the dynamic restrictors. 4) Finally, we observed that with the dynamic restrictors, there are more possibilities for choosing appropriate restrictors.

References

(Carroll 93) Carroll J. (1993). *Practical Unification-Based Parsing of Natural Language*. PhD thesis, Cambridge University,UK.

(Gerdemann 91) Gerdemann D. (1991). *Parsing and Generation of Unification Grammars*. PhD Thesis, University of Illinois, Technical Report CS-91-06 of The Beckman Institute.

(Grover et al. 93) Grover C., Carroll J., Briscoe T. (1993). *The Alvey Natural Language Tools Grammars(4th release)*. Technical Report N° 284, Computer Laboratory, Cambridge University,UK.

(La Serna 96) La Serna N. (1996). *Selecting Apropriate Restrictors with Wide-Coverage Unification-Based Grammars*. Research report UPV/EHU/LSI/TR15-96, University of the Basque Country, Spain.

(Shieber 85) Shieber S. (1985). *Using Restriccion to Extend Parsing Algorithms for Complex Feature Based Formalisms*. In ACL Proceedings, 23rd Annual Meeting, University of Chicago. Chicago, ILL.

Previous Workshops

IWPT'89

28-31 August 1989 - Pittsburg, PA (USA)

Unification and Classification: An Experiment in Information-based Parsing
Robert T. Kasper

Using Restriction to Optimize Unification Parsing
Dale Gerdemann

An Overview of Disjunctive Constraint Satisfaction
John T. Maxwell, Ronald M. Kaplan

A Uniform Formal Framework for Parsing
Bernard Lana

Head-Driven Bidirectional Parsing: A Tabular Method
Giorgio Satta, Oliviero Stock

Head-Driven Parsing
Martin Kay

Parsing with Principles: Predicting a Phrasal Node Before its Head Appears
Edward Gibson

The Computational Implementation of Principle-based Parsers
Sandiway Fong, Robert Berwick

Probabilistic Parsing Method for Sentence Disambiguation
T. Fujisaki, F. Jelinek., J. Cocke, E. Black, T. Nishino

A Sequential Truncation Parsing Algorithm Based on the Score Function
Keh-Yih Su, Jong-Nae Wang, Mei-Hui Su, Jing-Shin Chang

Probabilistic LR Parsing for Speech Recognition
J.H. Wright, E. N. Wrigley

Parsing Speech for Structure and Prominence
Dieter Huber

Parsing Continuous Speech by HMM-LR Method
Kenji Kita, Takeshi Kawabata, Hiroaki Saito

Parsing Japanese Spoken Sentences Based on HPSG
Kiyoshi Kogure

Probabilistic Methods in Dependency Grammar Parsing
Job M. van Zuijlen

Predictive Normal Forms for Composition in Categorical Grammars
Kent Wittenburg, Robert Wall

Parsing Spoken Language Using Combinatory Grammars
Mark Steedman

Recognition of Combinatory Categorial Grammars and Linear Indexed Grammars
K. Vijay-Shanker, David J. Weir

Handling of Ill-designed Grammars in Tomita's Parsing Algorithm
Rohman Nozohoor-Farshi

Analysis of Tomita's Algorithm for General Context-free Parsing
James R. Kipps

The Computational Complexity of Tomita's Algorithm
Mark Johnson

Probabilistic Parsing for Spoken Language Applications
Stephanie Seneff

Connectionist Models of Language
James L. McClelland

A Connectionist Parser Aimed at Spoken Language
Ajay Jain, Alex Waibel

Massively Parallel Parsing in DmDialog: Integrated Architecture for Parsing Speech Inputs
Hiroaki Kitano, Teruko Mitamura, Masaru Tomita

Parallel Parsing Strategies in NLP
Anton Nijholt

Complexity and Decidability in Left-Associative Grammar
Roland Hausser

The Selection of a Parsing Strategy for an On-line Machine Translation System in a Sublanguage Domain. A New Practical Comparison
Patrick Shann

Finite State Machines from Feature Grammars
Alan W. Black

An Effective Enumeration Algorithm of Parses for Ambiguous CFL
Tadashi Seko, Nariyoshi Yamai, Noboru Kubo, Toru Kawata

A Morphological Parser for Linguistic Exploration
David Weber

The Parallel Expert Parser: A Meaning-Oriented, Lexically-Guided, Parallel-Interactive Model of Natural Language Understanding
Geert Adriaens

Chart Parsing for Loosely Coupled Parallel Systems
Henry S.Thompson

Parallel LR Parsing Based on Logic Programming
Hozumi Tanaka, Hiroaki Numazaki

The Relevance of Lexicalization to Parsing
Yves Schabes, Aravind K. Joshi

A Framework for the Development of Natural Language Grammars
Massimo Marino

An Efficient Method for Parsing Erroneous Input
Stuart Malone, Sue Felshin

Analysis Techniques for Korean Sentences Based on Lexical Functional Grammar
Deok Ho Yoon, Yung Taek Kim

Learning Cooccurrences by Using a Parser
Kazunori Matsumoto, Hiroshi Sakaki, Shingo Kuroiwa

Parsing, Word Associations and Typical Predicate-Argument Relations
Kenneth Church, William Gale, Patrick Hanks, Donald Hindle

An Efficient, Primarily Bottom-Up Parser for Unification Grammars
Neil K. Simpkins, Peter J. Hancox

PREMO: Parsing by Conspicuous Lexical Consumption
Brian M. Slator, Yorick Wilks

Parsing Algorithms 2-Dimensional Language
Masaru Tomita

A Broad-Coverage Natural Language Analysis system
Karen Jensen

Pseudo Parsing Swift-Answer Algorithm
S. Pal Asija

A Dependency-Based Parser for Topic and Focus
Eva Hajicova

Parsing Generalized Phrase Structure Grammar with Dynamic Expansion
Navin Rudhiraja, Subrata Mitra, Harish Karnick, Rajeev Sangal

IWPT'91

13-15 February 1991 - Cancun (Mexico)

Parsing without Parser
Koiti Hasida, Hiroshi Tsuda

Parsing = Parsimonious Covering?
Venu Dasigi

The Valid Prefix Property and Left to Right Parsing of Tree-Adjoining Grammar
Yves Schabes

Preprocessing and Lexicon Design for Parsing Technical Text
Robert P. Futrelle, Christopher E. Dunn, Debra S. Ellis, Maurice J. Pescitelli, Jr.

Incremental LL(1) Parsing in Language-Based Editors
John J. Shilling

Linguistic Information in the Databases as a Basis for Linguistic Parsing Algorithms
Tatiana A. Apollonskaya, Larissa N. Beliaeva, Raimund G. Piotrowski

Binding Pronominals with an LFG Parser
Radolfo Delmonte, Dario Bianchi

A Hybrid Model of Human Sentence Processing: Parsing Right-Branching, Center-Embedded and Cross-Serial Dependencies
Theo Vosse, Gerard Kempen

Using Inheritance Object-Oriented Programming to Combine Syntactic Rules and Lexical Idiosyncrasies
Benoit Habert

An LR(k) Error Diagnosis and Recovery Method
Philippe Charles

Adaptive Probabilistic Generalized LR Parsing
Jerry Wright, Ave Wrigley, Richard Sharman

Phonological Analysis and Opaque Rule Orders
Michael Maxwell

An Efficient Connectionist Context-Free Parser
Klaas Sikkel, Anton Nijholt

Slow and Fast Parallel Recognition
Hans de Vreught, Job Honig

Processing Unknown Words in Continuous Speech Recognition
Kenji Kita, Terumasa Ehara, Tsuyoshi Morimoto

The Specification and Implementation of Constraint-Based Unification Grammars
Robert Carpenter, Carl Pollard, Alex Franz

Probabilistic LR Parsing for General Context-Free Grammars
See-Kiong Ng, Masaru Tomita

Quasi-Destructive Graph Unification
Hideto Tomabechi

Unification Algorithms for Massively Parallel Computers
Hiroaki Kitano

Unification-Based Dependency Parsing of Governor-Final Languages
Hyuk-Chul Kwon, Aesun Yoon

Pearl: A Probabilistic Chart Parser
David M. Magerman, Mitchell P. Marcus

Local Syntactic Constraints
Jacky Herz, Mori Rimon

Stochastic Context-Free Grammars for Island-Driven Probabilistic Parsing
Anna Corazza, Roberto Gretter, Giorgio Satta, Renato De Mori

Substring Parsing for Arbitrary Context-Free Grammars
Jan Rekers, Wilco Koorn

Parsing with Relational Unification Grammars
Kent Wittenburg

Parsing 2-D Languages with Positional Grammars
Gennaro Costagliola, Shi-Kuo Chang

IWPT'93

10-13 August 1993 - Tilburg (The Netherlands), Durbuy (Belgium)

Monte Carlo Parsing
Rens Bod

Transformation-Based Error-Driven Parsing
Eric Brill

Parsing as Dynamic Interpretation
Harry Bunt, Ko Van der Sloot

Compiling Typed Attribute-Value Logic Grammars
Bob Carpenter

(Pictorial) LR Parsing from an Arbitrary Starting Point
Gennaro Costagliola

A New Transformation into Deterministically Parsable Form for Natural Language Grammars
Nigel R. Ellis, Roberto Garigliano, Richard G. Morgan

A Principle-Based Parser for Foreign Language Training in German and Arabic
Joe Garman, Jeffery Martin, Paola Merlo, Amy Weinberg

An Algorithm for the Construction of Dependency Trees
Gerrit van der Hoeven

Integration of Morphological and Syntactic Analysis Based on the LR Parsing Algorithm
Tanaka Hozumi, Tokunaga Takenobu, Aizawa Michio

Structural Disambiguation in Japanese by Evaluating Case Structures based on Examples in Case Frame Dictionary
Sadao Kurohashi, Makato Nagao

An Efficient Noise-Skipping Parsing Algorithm for Context-Free Grammars
Alon Lavie, Masaru Tomita

The Use of Bunch Notation in Parsing Theory
René Leermakers

Chart Parsing of Attributed Structure-Sharing Flowgraphs with Tie-Point Relationships
Rudi Lutz

The Interplay of Syntactic and Semantic Node Labels in Partial Parsing
David D. McDonald

Increasing the Applicability of LR Parsing
Mark-Jan Nederhof, Janos J. Sarbo

Reducing Complexity in a Systemic Parser
Michael O'Donnell

Generalized LR Parsing and Attribute Evaluation
Paul Oude Luttighuis, Klaas Sikkel

A Proof-Theoretic Reconstruction of HPSG
Stephan Raaijmakers

Stochastic Lexicalized Context-Free Grammar
Yves Schabes, Richard C. Waters

Predictive Head-Corner Chart Parsing
Klaas Sikkel, Rieks op den Akker

Parsing English with a Link Grammar
Daniel D. Sleator, Davy Temperley

Evaluation of TTT Parser: A Preliminary Report
Tomek Strzalkowski, Peter G.N. Scheyen

Frequency Estimation of Verb Subcategorization Frames Based on Syntactic and Multidimensional Statistical Analysis
Akira Ushioda, David A. Evans, Ted Gibson, Alex Waibel,

Handling Syntactic Extra-Grammaticality
Fuliang Weng

Adventures in Multi-Dimensional Parsing: Cycles and Disorders
Kent Wittenburg

Probabilistic Incremental Parsing in Systemic Functional Grammar
A. Ruvan Weerasinghe, Robin P. Fawcett

IWPT'95

20-24 September 1995 - Prague, Karlovy Vary (Czech Republic)

* is a short paper

Acyclic Context-sensitive Grammars
E. Aarts

* *Parsing in Dialogue Systems Using Typed Feature Structures*
R. op den Akker, H. ter Doest, M. Moll, A. Nijholt

* *Parallel Parsing: Different Distribution Schemata for Charts*
J. Amtrup

A Fuzzy Approach to Erroneous Inputs in Context-free Language Recognitlon
P. Asveld

Parsing Non-Immediate Dominance Relations
T. Becker, O. Rambow

Yet Another O(n6) Recognition Algorithm for Mildly Context-sensitive Languages
P. Boullier

Developing and Evaluating a Probabilistic LR Parser of Part-of-Speech and Punctuation Labels
T. Briscoe T, J. Carroll

An Abstract Machine for Attribute-Value Logic
B. Carpenter, Y. Qu

A Chunking-and-Raising Partial Parser
H. Chen, Y. Lee

Distributed Parsing With HPSG Grammars
A. Diagne, W. Kasper, H. Krieger

* *Chart-based Incremental Semantics Construction with Anaphora Resolution Using λ-DRT*
I. Fischer, B. Geisert, G. Görz

Term Encoding of Typed Feature Structures
D. Gerdemann

Generic Rules and Non-Constituent Coordination
J. Gonzalo, T. Solías

A Robust Parsing Algorithm for Link Grammars
D. Grinberg, J. Lafferty, D. Sleator

An Implementation of Syntactic Analysis of Czech
T. Holan, V. Kubon, M. Plátek

Analyzing Coordinate Structures Including Punctuation in English
S. Kurohashi

* *On Parsing Control for Efficient Text Analysis*
A. Lavelli, F. Ciravegna
* *A Practical Dependency Parser*
V. Lombardo, L. Lesmo
A Labelled Analytic Theorem Proving Environment for Categorial Grammar
S. Luz-Filho, P. Sturt
A Unification-based ID/LP Parsing Schema
F. Morawietz
Parsing Without Grammar
S. Mori, M. Nagao
A Formalism and a Parser for Lexicalised Dependency Grammars
A. Nasr
Error-tolerant Finite State Recognition
K. Oflazer
A Novel Framework for Reductionist Statistical Parsing
Ch. Samuelsson
A Corpus-based Probabilistic Grammar with Only Two Non-terminals
S. Sekine, R. Grishman
Heuristics and Parse Ranking
B. Srinivas, Ch. Doran, S. Kulick
Stochastic Parse-Tree Recognition by a Pushdown Automaton
F. Tendeau
* *An HPSG-based Parser for Automatic Knowledge Acquisition*
K. Torisawa, J. Tsujii
Parsing D-Tree Grammars
K. Vijay-Shanker D. Weir, O. Rambow
The Influence of Tagging on the Results of Partial Parsing in German Corpora
O. Wauschkuhn
* *Partitioning Grammars and Composing Parsers*
F. Weng, A. Stolcke
Parsing with Typed Feature Structures
S. Wintner, N. Francez.