Parsing with Automatically Acquired, Wide-Coverage, Robust, Probabilistic LFG Approximations

Aoife Cahill

A dissertation submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy

to the

Dublin City University

School of Computing

Supervisors: Prof. Josef van Genabith
Dr. Andy Way

September 2004

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work

Signed Aoife Cahill

(Aoife Cahill)

Student ID 97093246

Date September 2004

Contents

Abstract

Traditionally, rich, constraint-based grammatical resources have been hand-coded. Scaling such resources beyond toy fragments to unrestricted, real text is knowledge-intensive, time-consuming and expensive.

The work reported in this thesis is part of a larger project to automate as much as possible the construction of wide-coverage, deep, constraint-based grammatical resources from treebanks. The Penn-II treebank is a large collection of parse-annotated newspaper text. We have designed a Lexical-Functional Grammar (LFG) (Kaplan and Bresnan, 1982) f-structure annotation algorithm to automatically annotate this treebank with f-structure information approximating to basic predicate-argument or dependency structures (Cahill et al., 2002c, 2004a). We then use the f-structure-annotated treebank resource to automatically extract grammars and lexical resources for parsing new text into f-structures.

We have designed and implemented the Treebank Tool Suite (TTS) to support the linguistic work that seeds the automatic f-structure annotation algorithm (Cahill and van Genabith, 2002) and the F-Structure Annotation Tool (FSAT) to validate and visualise the results of automatic f-structure annotation.

We have designed and implemented two PCFG-based probabilistic parsing architectures for parsing unseen text into f-structures: the pipeline and the integrated model. Both architectures parse raw text into basic, but possibly incomplete, predicate-argument structures ("proto f-structures") with long distance dependencies (LDDs) unresolved (Cahill et al., 2002c).

We have designed and implemented a method for automatically resolving LDDs at f-structure level based on a finite approximation of functional uncertainty equations (Kaplan and Zaenen, 1989) automatically acquired from the f-structure-annotated treebank resource (Cahill et al., 2004b).

To date, the best result achieved by our own Penn-II induced grammars is a dependency f-score of 80.33% against the PARC 700, an improvement of 0.73% over the best hand-crafted grammar of (Kaplan et al., 2004). The processing architecture developed in this thesis is highly flexible: using external, state-of-the-art parsing technologies (Charniak, 2000) in our pipeline model, we achieve a dependency f-score of 81.79% against the PARC 700, an improvement of 2.19% over the results reported in Kaplan et al. (2004).

We have also ported our grammar induction methodology to German and the TIGER treebank resource (Cahill et al., 2003a).

We have developed a method for treebank-based, wide-coverage, deep, constraint-based grammar acquisition. The resulting PCFG-based LFG approximations parse the Penn-II treebank with wider coverage (measured in terms of complete spanning parse) and parsing results comparable to or better than those achieved by the best hand-crafted grammars, with, we believe, considerably less grammar development effort. We believe that our approach successfully addresses the knowledge-acquisition bottleneck (familiar from rule-based approaches to AI and NLP) in wide-coverage, constraint-based grammar development. Our approach can provide an attractive, wide-coverage, multilingual, deep, constraint-based grammar acquisition paradigm.

Acknowledgements

I wish to acknowledge everyone who has helped me with this thesis. Firstly, I would like to express my deepest gratitude to Josef van Genabith, without whom, this thesis would not exist. Josef has always been patient and understanding and I really appreciate that his door was always open to me. He has been a wonderful mentor, full of helpful and insightful comments and most of all encouragement and enthusiasm. In this respect, I would also like to thank Andy Way. Andy has always supported and reassured me, whenever I had any doubts and his pragmatic approach to everything has been an inspiration.

Thanks also to all the members of the National Centre for Language Technology, and the staff and post-graduate students of the School of Computing at DCU for all their helpful suggestions whenever I presented my work. Special thanks to Mary, Mick and Ruth for listening to my crazy half-baked ideas and discussing them with me. I would also like to thank Michelle, Nano and everybody else in CAPG for keeping me sane. I am grateful to Dalen and Tom for all their technical expertise, they have taught me so much!

My time at the University of Stuttgart in the summer of 2003 was thoroughly enjoyable and extremely rewarding. I would especially like to thank Helmut Schmid, Martin Forst, Christian Rohrer and Michael Schielen for all their help and support while I was there.

I would like to thank everybody at the Natural Language Theory and Technology group at the Palo Alto Research Center, I have always enjoyed my visits and come home with a plethora of ideas. I would especially like to thank Stefan, Tracy, Ron and Dick for the fruitful discussions we have had. Also, thanks to Dick for providing us with his evaluation software.

I would also like to thank all of my friends outside of DCU for their constant friendship and support over the course of this Ph.D. It is impossible to name everybody, but special thanks to Rachael, Nico, Paul, Jerry, Brian, Valerie, Frances, Claudia, Siobhán and Sarv for helping me to keep my head out of the clouds and reminding me that there is life outside of thesis-writing!

Finally, I would like to thank my parents Maria and Michael, my sisters Fiona and Róisín, and my best friend Barry for their unquestioning belief in me. Without their confidence in me, I would never have had the courage to go through with this. Thank

you (especially Barry) for putting up with me, even in my worst humours and helping me through the tough times.

I am grateful to Enterprise Ireland for their financial support of our project (Basic Research Grant SC/2001/186), enabling us to carry out the work reported in this thesis.

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
AVM	Attribute-Value Matrix
CCG	Combinatory Categorial Grammar
CF-PSG	Context-Free Phrase Structure Grammar
CFG	Context-Free Grammar
CNF	Chomsky Normal Form
DOP	Data-Oriented Parsing
FU	Functional Uncertainty
FUG	Functional Unification Grammar
GF	Grammatical Function
GPSG	Generalized Phrase Structure Grammar
HPSG	Head-driven Phrase Structure Grammar
LDD	Long-Distance Dependency
LFG	Lexical-Functional Grammar
LHS	Left Hand Side
NLP	Natural Language Processing
PCFG	Probabilistic Context-Free Grammar
POS	Part Of Speech
RHS	Right Hand Side
SVO	Subject-Verb-Object
TAG	Tree Adjoining Grammar
VSO	Verb-Subject-Object
WSJ	Wall Street Journal
XTAG	X-windows based TAG development environment

Chapter 1

Introduction

Parsing (determining the syntactic structure of a string) is an important step in natural language processing, as syntactic structure strongly determines semantic interpretation in the form of predicate-argument structures, dependency relations or logical form representations. For a substantial number of linguistic phenomena such as topicalisation, wh-movement in relative clauses and interrogative sentences, however, there is an important difference between the location of the (surface) realisation of linguistic material and the location where this material should be interpreted semantically. Resolution of such long-distance dependencies (LDDs) is, therefore, crucial in the determination of accurate predicate-argument structure, deep dependency relations and the construction of proper meaning representations such as logical forms (Johnson, 2002). Modern unification/constraint-based grammars such as Lexical-Functional Grammar (LFG) (Kaplan and Bresnan, 1982; Bresnan, 2001; Dalrymple, 2001) or Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) capture deep linguistic information including LDDs, predicate-argument structure, or logical form.

Developing deep unification/constraint-based grammars is a highly knowledge-intensive task and grammars are typically hand-crafted. Scaling rich, unification/constraint-based, computational grammatical resources beyond small fragments to unrestricted text is time-consuming and expensive, involving person-years of concerted grammar and lexicon development (Butt et al., 1999, 2002). Few hand-crafted, deep unification grammars have in fact achieved the coverage and robustness required to parse a corpus of (say) the size

and complexity of the Penn treebank: Riezler et al. (2002) show how a deep, carefully hand-crafted LFG is successfully scaled to parse the Penn-II treebank (Marcus et al., 1994) with discriminative (log-linear) parameter estimation techniques.

The situation is familiar from other knowledge-intensive engineering tasks in traditional rule-based, "rationalist" approaches in Artificial Intelligence (AI) and Natural Language Processing (NLP): it is an instance of the famous knowledge-acquisition bottleneck.

At the same time, much recent work in NLP has been corpus based, following what has been referred to as an "empiricist" research tradition. Treebank resources are available for increasing numbers of languages and treebank-based, probabilistic grammar induction and parsing is a cutting-edge research paradigm (Charniak, 1996; Johnson, 1999; Charniak, 2000; Klein and Manning, 2003). Such approaches are attractive as they achieve coverage, robustness and performance while incurring very low grammar development cost. With a number of notable exceptions (Collins, 1999; Johnson, 2002; Hockenmaier, 2003), however, most of the induced grammars are "shallow", i.e. they do not map text to information and even these exceptions are substantially less detailed than current unification/constraint-based grammars such as LFG and HPSG in the rationalist paradigm.

This situation poses a research question: is it possible to combine rationalist and empirical research methods to induce rich, wide-coverage, constraint-based grammars from treebanks?

This thesis presents a method of extracting wide-coverage, robust, probabilistic context-free grammar (PCFG)-based LFG approximations from automatically f-structure-annotated treebanks, two flexible PCFG-based processing architectures for parsing with such resources, a method for automatically resolving LDDs at f-structure level based on a finite approximation of functional uncertainty equations extracted from the f-structure-annotated treebank resource and extensive evaluation of all these resources.

Our approach requires an f-structure-annotated treebank. We have designed and developed an algorithm which automatically annotates the Penn-II treebank with f-structure information (Cahill et al., 2002a; Burke et al., 2004b). The Penn-II treebank uses traces, coindexation and functional tags to represent a certain amount of deep linguistic information. Our algorithm exploits this information together with categorial, configurational and

head information to automatically annotate the Penn-II treebank with abstract syntactic functional (LFG f-structure) information approximating to basic predicate-argument structures.

In order to successfully develop the automatic f-structure annotation algorithm, additional support tools are required. We designed and implemented the Treebank Tool Suite (TTS) (Cahill and van Genabith, 2002) to support the population of annotation matrices which constitute the linguistic basis for the automatic f-structure annotation algorithm. Once the automatic f-structure annotation algorithm has been implemented, we require tools to visualise and validate the output of automatic annotation. For this purpose, we designed and implemented the F-Structure Annotation Tools (FSAT).

PCFG parsing is a core technology used in this dissertation. A number of grammar transformations have been proposed to improve the quality of PCFG-based parsing, most notably those of Johnson (1999) and Klein and Manning (2003). In addition, there are a number of possible pre-processing steps that are usually carried out before automatically extracting a PCFG from a treebank. We thoroughly investigate pre-processing steps and grammar transformations and the way in which they interact in order to determine which combinations will produce the highest quality parse trees.

We have used our automatic f-structure annotation algorithm to develop two PCFG-based parsing architectures that parse raw text into f-structures (Cahill et al., 2002c). In the **pipeline** architecture, we first extract a PCFG from the unannotated Penn-II treebank to parse new text. We then automatically annotate the most probable parse with LFG f-structure equations using our f-structure annotation algorithm. The equations are collected and passed to a constraint solver to generate an f-structure. In the **integrated** model, we first annotate the Penn-II trees with f-structure equations and extract an annotated PCFG from the annotated treebank. We treat strings consisting of CFG categories followed by one or more f-structure equations as monadic categories for grammar extraction and parsing. We then parse with the annotated grammar and choose the f-structure-annotated tree with the highest probability. We collect the f-structure equations and pass them to a constraint solver to generate an f-structure.

Both architectures produce "proto f-structures": basic, but possibly incomplete, predicate-

argument structures with long-distance dependencies unresolved. Like in most other current PCFG-based parsing technology (Johnson, 1999; Charniak, 2000; Klein and Manning, 2003), linguistic material is interpreted purely locally where it occurs in the tree. In this thesis, we show how LDDs can be resolved at f-structure level based on a finite approximation of functional uncertainty equations (Kaplan and Zaenen, 1989; Dalrymple, 2001) and LFG subcategorisation frames automatically acquired from the f-structure-annotated treebank resource (Cahill et al., 2004b; O'Donovan et al., 2004). Unlike (Collins, 1999; Johnson, 2002), in our approach LDDs are resolved on the level of f-structure representation, rather than in terms of empty productions and co-indexation on parse trees. In order to reliably determine the quality of the f-structures generated by our methodology, we evaluate against three gold standards: the DCU 105 (Cahill et al., 2002a), the automatically generated 2416 f-structures for the original trees in section 23 of the Penn-II treebank in a CCG-style experiment (Hockenmaier, 2003), and the PARC 700 Dependency Bank (King et al., 2003). Currently our own induced grammars achieve preds-only f-structure/dependency f-scores of 80.33% and 81.24%, evaluating against the PARC 700 and DCU 105, respectively. We achieve a preds-only f-score of 79.38% against the automatically generated 2416 f-structures for the original trees in section 23 of the Wall Street Journal (WSJ) part of the Penn-II treebank.

We compare our work to other similar work on the resolution of LDDs. Collins (1999) only deals with wh-movement in relative clauses. It is difficult to carry out a satisfactory comparison, but we perform an experiment to compare our work at f-structure level. Our method of resolving LDDs at f-structure level performs better than Collins' Model 3 trees with traces, evaluating against the DCU 105, the PARC 700 and the automatically generated 2416 f-structures for the original trees in section 23. In a post-processing approach, (Johnson, 2002) inserts empty nodes and their antecedents into parse trees in order to capture long-distance dependencies. Again, it is difficult to compare our work directly, but we carry out an experiment at f-structure level, evaluating against the DCU 105, the PARC 700 and the automatically generated 2416 f-structures for the original trees in section 23. In all experiments, our method of resolving LDDs at f-structure level yields higher results than Johnson's (2002) method of adding empty nodes and co-indexation to Charniak's

(2000) parser output to capture LDDs. Given the trees generated by Charniak's (2000) parser, our method of resolving LDDs at f-structure achieves a preds-only f-score of 80.65% against the DCU 105. Generating f-structures from these trees with empty nodes added by Johnson's (2002) software achieves an f-score of 79.52%. Kaplan et al. (2004) evaluate their hand-crafted LFGs against a subset of the PARC 700 with a reduced feature-set and achieve an f-score of 79.6%. For the same experiment, our own best PCFG achieves an f-score of 80.33%. Using the output of Charniak's (2000) parser in our pipeline model, our method achieves an f-score of 81.79%, an improvement of 2.19% on the results of (Kaplan et al., 2004).

It is well known that PCFG-based approximations of unification grammars do not provide an adequate probability model (Abney, 1997), as sometimes probability mass is lost when the parser produces a most probable parse, but this parse cannot generate an f-structure. Given this, it is perhaps surprising that our automatically induced deep LFG resources and PCFG-based approximations outperform the best hand-crafted wide-coverage constraint-based grammars and sophisticated processing approaches.

Substantial treebanks (or dependency banks) are now available for many languages (including English, Japanese, Chinese, German, French, Czech, Turkish), while others are currently under construction (Arabic, Bulgarian) or near completion (Spanish, Catalan). We show how our method of acquiring large-scale, probabilistic LFG approximations can be migrated to German, a topologically different language (Cahill et al., 2003a), using the TIGER treebank resource (Brants et al., 2002). We successfully extract PCFG-based LFG approximations for German and parse unseen German text into LFG f-structures. Currently we achieve an f-score of 71% against a manually constructed gold standard of 100 dependency structures.

This thesis presents a method for automatically acquiring large-scale, robust, probabilistic English and German LFG approximations. We present a method for resolving long-distance dependencies at f-structure level, enabling our parsers to produce deep linguistic representations. We show that, although our PCFG approximations do not provide a strictly adequate probability model, we achieve parsing results for English on the WSJ section of the Penn-II treebank equal to or better than those achieved by the best state-

of-the-art hand-crafted grammars.

This thesis is structured as follows:

Chapter 2 describes and evaluates an automatic f-structure annotation algorithm for English and the Penn-II treebank.

Chapter 3 describes a suite of tools used in the development and the application of the automatic f-structure annotation algorithm presented in Chapter 2.

Chapter 4 outlines basic probabilistic context-free parsing models, and gives a brief discussion on alternative approaches to probabilistic parsing.

Chapter 5 describes a number of pre-processing steps and grammar transformations that can be applied to a treebank before extracting a PCFG. Each is examined in detail and an extensive evaluation of each transformation and pre-processing step and how they interact is carried out.

Chapter 6 presents two parsing architectures (pipeline and integrated) for parsing into LFG f-structures. We evaluate the basic, but possibly partial, predicate-argument structures ("proto f-structures") generated. We then describe our method of resolving long-distance dependencies at the level of f-structure to produce "proper f-structures" and extensively evaluate the proper f-structures produced.

Chapter 7 compares our method of generating deep linguistic representations to other related work, in particular to the work of Collins (1999),Johnson (2002) and Riezler et al. (2002); Kaplan et al. (2004) and present results using external state-of-the-art parsers (Collins, 1999; Charniak, 2000) in our pipeline processing architecture.

Chapter 8 describes how we adapt the methodology developed for English to German and the TIGER treebank. We carry out evaluation of the German f-structures. We perform and evaluate a morphological case-simulating grammar transformation.

Chapter 9 concludes and outlines some areas of future work.

Chapter 2

Automatic F-Structure Annotation

2.1 Background and Motivation

Deep grammars relate strings to information, represented in terms of logical forms, deep dependency or predicate-argument-adjunct structures. Deep unification or constraint-based grammars are usually hand-crafted, time-consuming and expensive to develop, and rarely achieve the coverage of state-of-the-art treebank-based probabilistic grammars. Much current parsing technology is treebank based, with grammars automatically induced from available treebank resources. The Penn-II treebank (Marcus et al., 1994), for example, has been used to automatically extract probabilistic context-free grammars (PCFGs) (Charniak, 1996), combinatory categorial grammars (CCGs) (Hockenmaier and Steedman, 2002), tree-adjoining grammars (TAGs) (Xia, 1999) and lexicalised grammars (Charniak, 1997). Extracting grammars from treebanks is fast, cheap and provides wide-coverage grammars. The main disadvantage with this approach, however, is that the analyses provided by these grammars are mostly "shallow". They do not map strings into meaning representations and, with a few notable exceptions, do not attempt to resolve long-distance dependencies.

This poses a research question: can "deep", probabilistic constraint-based grammars, such as lexical functional grammar(LFG), be acquired from treebanks? The answer is yes,

if an f-structure annotated treebank is at our disposal. Unfortunately, however, such a resource does not exist, so we have to create one. First generation treebanks represented mainly surface syntactic information in the form of CFG parse trees, while many second generation treebanks contain a certain amount of deep linguistic information to support the computation of "meaning" representations. The Penn-II treebank, for example, uses traces, coindexation and functional tags to represent deep linguistic information. We have designed an f-structure annotation algorithm which exploits this information together with categorial and configurational information to automatically annotate the Penn-II treebank with abstract syntactic functional information approximating to basic predicate-argument structures (Cahill et al., 2002a; Burke et al., 2004b).

In this chapter, I will first outline Lexical Functional Grammar and why we have chosen to annotate the Penn-II treebank using this formalism. I will briefly outline some previous approaches to automatic f-structure annotation and present our own annotation algorithm, including a description of the architecture and a thorough evaluation of our system.

2.2 Lexical Functional Grammar

LFG (Kaplan and Bresnan, 1982; Bresnan, 2001; Dalrymple, 2001) belongs to the family of unification or constraint-based grammars. I will first briefly describe constraint-based grammars in general. I will then go on to describe LFG in more detail, and finally I will argue that LFG is a very suitable framework for our automatic annotation project.

2.2.1 Unification (or Constraint-Based) Grammars

Constraint-based grammars extend context-free formalisms with the addition of *Feature Structures* or Attribute Value Matrices (AVMs). An AVM is a graphical representation of finite hierarchical sets. Alternatively, an AVM is a representation of a minimal, canonical model satisfying a logical description involving Boolean combinations of expressions in an equality logic (Johnson, 1988; Kaplan, 1995). Values of features in an AVM can be either complex or atomic, i.e values can, but must not, be other AVMs. Figure 2.1 shows an AVM induced by the conjunction over a set of terms (constraints) in an equality logic. Members

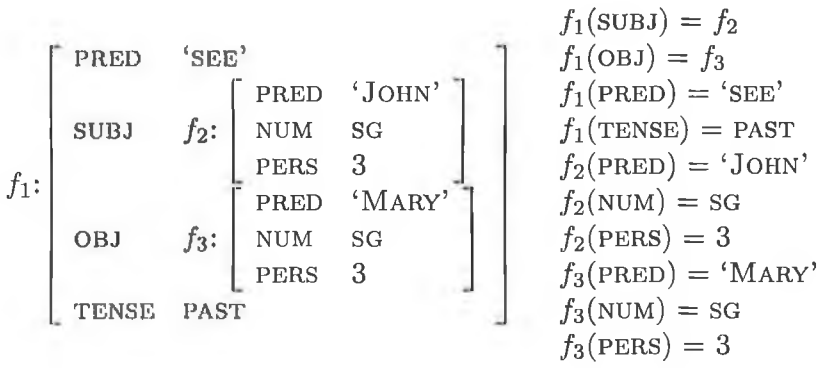


Figure 2.1: An example AVM satisfying a set of terms (constraints)

of the unification grammar family include LFG (Kaplan and Bresnan, 1982), Functional Unification Grammars (FUG) (Kay, 1985), PATR-II (Shieber, 1984), Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1985) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994).

2.2.2 LFG

Minimally, LFG (Kaplan and Bresnan, 1982; Bresnan, 2001; Dalrymple, 2001) involves two levels of representation: c(onstituent)-structure and f(unctional)-structure. C-structure captures language-specific phenomena such as word order and the grouping of constituents into larger phrases in the form of context-free trees. F-structure represents abstract syntactic functions such as SUBJ(ect), OBJ(ect), PRED(icate), COMP(lement), XCOMP(lement), OBL(ique), ADJUNCT etc. in the form of recursive attribute-value structures. F-structures approximate to predicate-argument-modifier representations, simple logical forms (van Genabith and Crouch, 1996; Cahill et al., 2003b) or deep dependency relations.

C-structure and f-structure representations are related in terms of "functional annotations" of the form $\uparrow \ldots = \downarrow \ldots$ on tree nodes, i.e. attribute-value structure equations (or more generally: disjunctive, implicational, negative and set membership constraints) describing f-structures. Figure 2.2 shows an example c- and f-structure for the sentence *John saw Mary*. Each node in the c-structure is annotated with f-structure equations, e.g. $\uparrow$ SUBJ= $\downarrow$. The uparrows ($\uparrow$) point to the f-structure associated with the mother node, downarrows ($\downarrow$) to that of the local node. In a complete parse tree these $\uparrow$ $\downarrow$ meta variables

are instantiated to unique tree node identifiers and a set of constraints (a set of terms in the equality logic) is generated which (if satisfiable) generates an f-structure.

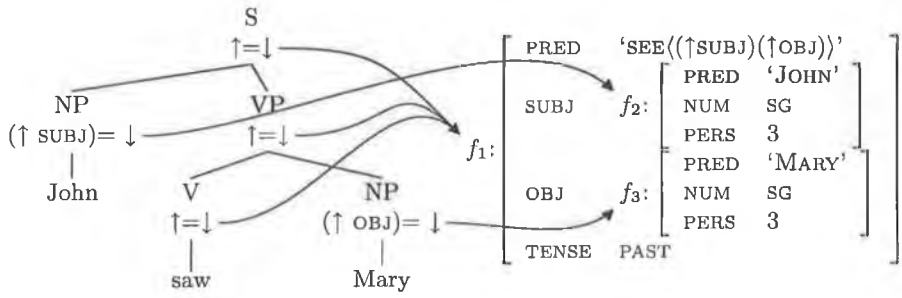


Figure 2.2: C- and f-structures for the sentence *John saw Mary*

2.2.3 Why the LFG Framework?

LFG is particularly attractive for multilingual grammar development as the level of f-structure representation abstracts away from certain (but not all) particulars of language-specific surface realisation (Butt et al., 1999, 2002). At the same time LFG provides a precise, flexible, computationally tractable and non-transformational interface between c-structure and f-structure representation for both parsing and generation (Butt et al., 2002).

There are two specific reasons for using the LFG framework in our automatic annotation project. The first is that while languages differ with respect to surface representations, they may encode the same (or very similar) abstract syntactic (and semantic) predicate-argument structures. Figure 2.3 illustrates this point. Irish is typologically a VSO-language, while English is an SVO-language. The same proposition expressed in Irish and English exhibits markedly different c-structure configurations but is associated with isomorphic (up to the values of PRED nodes) f-structure representations. This is a very useful property for applications such as machine translation (Kaplan et al., 1989).

The second reason is that unlike other constraint-based grammar formalisms, LFG has enjoyed a substantial body of work on automatic f-structure annotation architectures summarised in (Cahill et al., 2002c; Frank et al., 2003). These approaches automatically annotate (treebank or parse-generated) trees with f-structure equations to generate f-structures for those trees.

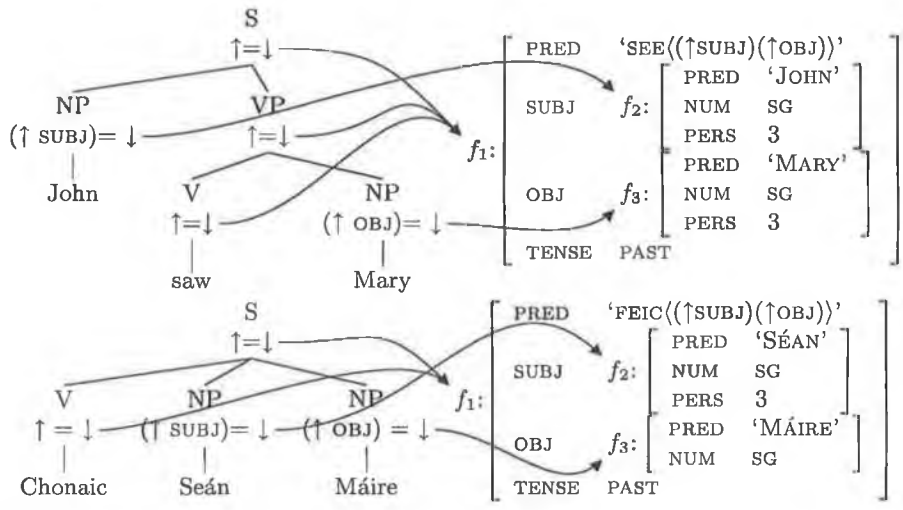


Figure 2.3: C- and f-structures for an English and corresponding Irish sentence

Our work builds on this research and develops more robust and large-scale systems. Our algorithm is theoretically formalism-independent and could, for example, also be applied to developing a large-scale HPSG grammar. Miyao et al. (2003) extract an HPSG for the Penn-II treebank, while earlier work by Tateisi et al. (1998) derives an HPSG from the hand-crafted English XTAG (Egedi et al., 1994). In this dissertation, however, we only deal with LFG.

2.3 Previous Work on Automatic Annotation

It would be desirable to have a substantial treebank, such as Penn-II, annotated with f-structure information as a resource for extracting probabilistic unification grammar resources. The large number of context-free rule types that occur in the Penn-II treebank (>17000) make it unfeasible to manually annotate each rule type with f-structure information. Therefore we would like to be able to automatically annotate such a large resource. Automatically generating f-structures from c-structure is not a completely new idea. Essentially there have been three approaches to date, each of which I will describe in more detail below:

- Annotation algorithms (Lappin et al., 1989),
- Regular expression-based annotation (Sadler et al., 2000),

- Flat, set-based tree description rewriting (Frank, 2000).

There are two ways to derive an f-structure from a tree: **direct** transformation or **indirect** annotation. The direct method recursively and destructively transforms a treebank tree into an f-structure. The indirect method only ever adds information: it annotates the treebank tree with f-structure annotations. These annotations are then collected and passed to a constraint solver which resolves the equations and, if the equations are consistent, outputs an f-structure.

2.3.1 Annotation Algorithms

The earliest approach to automatically identifying functional grammatical categories such as SUBJ, OBJ, etc in phrase structure trees is probably due to Lappin et al. (1989). Nodes in trees are identified which correspond to grammatical functions. Their motivation was to generate a set of grammatical function-based transfer rules as part of a machine translation project.

Unpublished work (1996) by Ron Kaplan (p.c.) reports a *direct* automatic f-structure transformation algorithm to generate f-structures for an LFG-DOP project (Bod and Kaplan., 1998). His approach was to transform a tree (from the ATIS corpus) into an f-structure by walking through the tree and restructuring the tree into an f-structure.

2.3.2 Regular Expression-Based Annotation

A regular expression-based, *indirect* automatic annotation method is described in Sadler et al. (2000). This involves extracting a context-free phrase structure grammar (CF-PSG) from a treebank fragment. F-structure annotation principles are stated in terms of regular expressions matching CF-PSG rules. By applying regular expression-based annotation principles to the rules that are extracted, and using these annotated rules to re-match the original trees, f-structures can be generated for these trees. The number of annotation principles is appreciably smaller than the number of extracted CFG rule types since the regular expression-based annotation principles capture linguistic generalisations.

2.3.3 Set-Based Tree Description Rewriting

The flat, set-based tree description rewriting method of automatically annotating trees with f-structure descriptions developed by Frank (2000), can be seen as a generalisation of the regular expression-based technique of Sadler et al. (2000). Here the idea is that each tree is translated into a flat set description with terms from a tree description language. Annotation principles are then defined in terms of rules employing a rewriting system originally developed for transfer-based machine translation architectures (Kay et al., 1994). In certain circumstances, the principles can be applied order independently, or in a particular cascading order. One of the advantages of this method is that tree fragments of arbitrary depth can be considered, whereas in the regular expression-based method, tree depth is limited to 1 (i.e. CFG rules).

The methodologies presented in Sadler et al. (2000) and Frank (2000) have been 'proof-of-concept' and to date have only been applied to small subsets of the AP and Susanne corpus of the order of 100–200 trees. This, however, is not to claim that they cannot be scaled to a complete treebank.

2.4 Our Annotation Algorithm

In the work reported here, we have chosen the algorithmic approach to automatic annotation and developed an indirect annotation algorithm to annotate the >48,000 parse-annotated strings (without FRAG(ment) or X(unknown) constituents) in the Wall Street Journal (WSJ) section of the Penn-II treebank. The algorithm recursively traverses each tree and annotates each node in the tree with f-structure information. The annotation algorithm was developed in two stages. The first stage produces what we refer to as "proto" f-structures which represent basic predicate-argument structure, but do not resolve long-distance dependencies (LDDs). LDDs were treated in the second stage. The annotation algorithm is in a continuous state of development to further improve annotation results. Here, I will outline the core infrastructure and current results. For more detail on the annotation algorithm, see McCarthy (2003) and Burke (forthcoming). I will describe how we generate f-structures from f-structure-annotated trees and how we evaluate the f-structures

our annotation algorithm produces. Finally, I will briefly describe the implementation of the system.

2.4.1 Proto F-structures

Proto f-structures capture basic predicate-argument structure. Long-distance dependency phenomena such as topicalisation and wh-movement are partially represented in terms of TOPIC, TOPICREL and FOCUS functions but are not resolved as arguments of the PREDS subcategorising for the "moved" linguistic material. In order to support the maintainability and extensibility of the linguistic information encoded in the algorithm, we separate the linguistic data from the algorithm itself and adopt a modular design. To produce proto f-structures, the following three modules are required. I will discuss them in greater detail below.

- Left-Right Context
- Coordination
- Catch-All and Clean-Up

Left-Right Context

The algorithm considers the elements of a local subtree of depth one (i.e. a CFG rule), and, if there is no coordination element (i.e. CC or CONJP on the RHS or UCP on the LHS),[1] left-right context annotation principles are applied. Coordination is treated separately in order to keep the left-right context annotation principles simple and perspicuous. The algorithm first locates the head daughter h of a local subtree, effectively creating a tripartition of left-context $l_1 \ldots l_n$, followed by head h, followed by right context $r_1 \ldots r_m$: LHS $\rightarrow$ $l_1 \ldots l_n$ h $r_1 \ldots r_m$. We use a modified version of Magerman's (1994) head-finding rules to locate the head.[2] The complete set of modified rules is listed in Table 2.1. To find the head of a constituent, the algorithm first identifies the mother category in the "Category"

[1] See Appendix A for a description of the POS tags used in the Penn-II Treebank

[2] Some of Magerman's original rules were changed in order to give better results. For example, MD (modal verb) was moved from after VP to the start of the list of possible heads for VP, since in our scheme, the modal verb is always the head of a VP. More detail about the changes made can be found in McCarthy (2003).

column. If the value in the "direction" column is "Left", it searches the daughters from left to right (head initial), otherwise, from right to left (head final). For each daughter X in the list of values in the table, it scans the children of the local constituent for the first occurrence of category X. If X occurs, that child is the head. If no child matches, the algorithm uses the first (or last, for head-final) child as the head. Asterisks (****) indicate that any categories after the asterisks should only be chosen as the head if there are no other matching categories. If there is a daughter whose category is one other than those occurring after the asterisks, choose the left-most or right-most of these daughters according to the search direction. Otherwise, choose the left-most or right-most daughter as the head. The head of an NP node is determined by a separate list of rules. To locate the head of an NP node, the algorithm first searches from right to left looking for the first node whose category label begins with N and satisfies the following conditions: (i) the category label does not have any Penn-II functional tags and (ii) if the node has label NP, it must not be preceded by punctuation. If no category is found, the information in Table 2.1 is used to find the head.

Category	Direction	Values
ADJP	Right	% QP JJ VBN VBG ADJP $ JJR JJS DT FW IN **** RBR RBS RB
ADVP	Left	RBR RB RBS FW ADVP CD **** JJR JJS JJ NP
CONJP	Left	CC RB IN
FRAG	Left	
INTJ	Right	
LST	Left	LS :
NAC	Right	NN NNS NNP NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW
NP	Right	EX $ CD QP PRP VBG JJ JJS JJR ADJP DT FW RB SYM PRP$ **** PRN POS
PP	Left	IN TO FW
PRN	Left	
PRT	Left	RP
QP	Right	$ % CD NCD QP JJ JJR JJS DT
RRC	Left	VP NP ADVP ADJP PP
S	Right	TO VP SBAR ADJP UCP NP PP-PRD ADJP-PRD NP-PRD
SBAR	Right	IN S SQ SINV SBAR FRAG X
SBARQ	Right	SQ S SINV SBARQ FRAG X
SINV	Right	MD IN VBZ VBD VBP VB AUX VP S SINV ADJP NP
SQ	Right	MD VBZ VBD VBP VB AUX VP SQ
UCP	Left	CC S **** ADVP RB PRN
VP	Left	MD VBD VBN VBZ VB VBG VBP POS AUX AUXG VP TO ADJP JJ NP
WHADJP	Right	JJ ADJP
WHADVP	Left	WRB
WHNP	Right	NN NNS NNP NNPS NP WDT WP WP$ WHADJP WHPP WHNP
WHPP	Left	IN TO FW
X	Left	

Table 2.1: Our complete list of head-finding rules, based on a version of Magerman's (1994) rules.

Once we have identified the head of each constituent, we use categorial and configu-

Left Context	Head	Right Context
DT: $\uparrow$ SPEC:DET $= \downarrow$	NN, NNS ...	NP: $\downarrow \in \uparrow$ APP
QP: $\uparrow$ SPEC:QUANT $= \downarrow$	$\uparrow = \downarrow$	PP: $\downarrow \in \uparrow$ ADJUNCT
JJ, ADJP: $\downarrow \in \uparrow$ ADJUNCT		S, SBAR:$\uparrow$ RELMOD $= \downarrow$

Table 2.2: Simplified sample NP annotation matrix

rational information to assign annotations to the other constituents in the tree. We do this by consulting a left-right annotation matrix for the mother node of each local subtree. A simplified sample matrix for NP rules is shown in Table 2.2. The matrix specifies, for example, that a DT node to the left of the head node of an NP constituent should get the annotation $\uparrow$ SPEC:DET $= \downarrow$.

Populating the left-right context annotation matrices is done by hand using linguistic expertise. Given the scale of the Penn-II treebank, with more than 17,000 rule types, it is impossible to analyse each rule individually. Instead we only look at the most frequent rule types and extract generalisations. For each LHS category, we analyse the most frequent rule types such that the token occurrence of these rule types in the corpus covers at least 85% of all occurrences of rules expanding that particular LHS. To give an example, instead of looking at >6,000 different NP rule types in the Penn-II treebank, we only consider the 102 most frequent ones to populate the NP annotation matrix (only 1.7% of the total NP rule types). These 102 NP rule types constitute 85% of all NP rule tokens. For example, the rule NP $\rightarrow$ JJ NNS is a very common rule occurring 9,553 times in the basic grammar we extracted from the Penn-II treebank. However, the rule NP $\rightarrow$ JJ JJ JJ JJ NNS only occurs once. While analysing the first rule, we come up with the generalisation that a JJ (adjective) to the left of a head in an NP rule should receive the annotation $\downarrow \in \uparrow$ ADJUNCT. This generalisation also applies to the less frequent rule providing a reasonable analysis for all elements of the rule. The fact that we can sample so few rules and still achieve good results is due to an interesting Zipfian property of treebanks. For each rule LHS category, a small number of very frequently occurring rules expand those categories, while there exists a large number of much less frequent rules, many of which may occur only once or twice in the whole treebank. In total we have constructed 23 left-right context annotation matrices for the 23 atomic (without Penn-II tags) LHS categories

in Penn-II. The matrices are then generalised in that they also apply to rules with LHSs with Penn-II tags, such as NP-TMP, NP-LOC etc. More detail on the annotation matrices can be found in McCarthy (2003) and Burke (forthcoming).

Coordination

Due to the comparatively flat analyses in the Penn-II treebank, coordinate constructions can be difficult to analyse. In order to keep our Left-Right Context annotation principles simple and perspicuous, they are only applied if the local tree does not contain coordination. We provide two sets of coordination annotation principles, depending on the type of coordination. For like-constituent coordination, we build sets of linguistically similar categories depending on the category of the LHS mother node, e.g. {NN, NNS, NNP} etc for NP nodes. Using these *similarity sets*, we are able to determine the elements of a constituent that should be part of the coordination set, and which remaining constituents should be analysed using the left-right context annotation principles. Figure 2.4 shows the annotation of a coordinated VP structure. The VPs belong in the similarity set of VP, and so receive the annotation $\downarrow \in \uparrow$conj. The annotation for the NP is found in the left-right context annotation matrix for VP, which says that an NP to the right of the head in a VP receives the annotation $\uparrow$obj=$\downarrow$. Unlike-constituent coordination is marked in the Penn-II treebank by means of the UCP category. Figure 2.5 shows a UCP structure for the string *207 and growing*. We treat this differently to like-constituent coordination with CC and CONJP constituents. Unlike-constituent coordination is more complex and difficult to analyse. Fortunately it does not occur very often, with 288 type and 590 token occurrences in the Penn-II treebank. For more detail on how we deal with coordination, see McCarthy (2003) and Burke (forthcoming).

Catch-All and Clean-Up

This step exploits the Penn-II functional information tags present in some of the node categories of the Penn-II treebank. Initially, the algorithm looks at the category labels and any functional tags on nodes that have received no f-structure annotation. By default, if the node has any functional tag information (e.g. -TMP, -LOC ...), we assign it

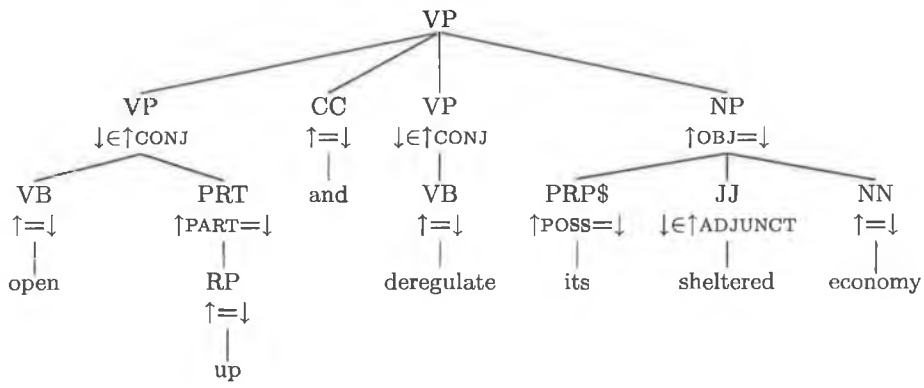


Figure 2.4: Annotating VP coordination with similarity sets

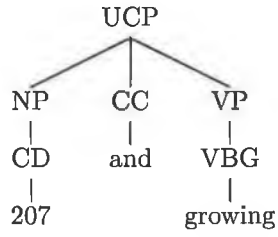


Figure 2.5: Unlike-constituent coordination in the Penn-II treebank

the annotation $\downarrow \in \uparrow$ ADJUNCT, meaning that the local node's f-structure is an element of the adjunct set of its mother node's f-structure. If the category label on the node is PRN (parenthetical), we also mark it as an adjunct. Sometimes it is necessary to overwrite default annotations on nodes. For example, PP-CLR always gets annotated as an oblique argument (i.e. $\uparrow$OBL=$\downarrow$), overwriting any default annotation it might have previously received. Figure 2.6 shows an example structure with a PP-CLR node receiving the annotation $\uparrow$OBL=$\downarrow$. -CLR indicates a close relationship to the verb and we assume that a prepositional phrase that is closely related to the verb is in fact an oblique argument of that verb. Our left-right context annotation matrices sometimes overgeneralise. This is also corrected in the Catch-All and Clean-Up phase. For example, the VP matrix specifies that an NP occurring to the right of the head in a VP rule receives the annotation $\uparrow$OBJ=$\downarrow$. However, if there are two NP arguments of the verb (i.e. direct and indirect), they cannot both receive the annotation $\uparrow$OBJ=$\downarrow$, since, as well as being incorrect, the constraint solver would not be able to resolve the equations and would fail to generate an f-structure. The

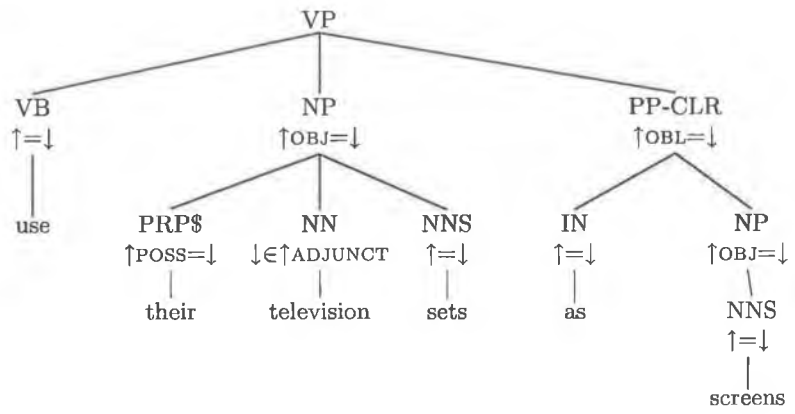


Figure 2.6: Annotating PP-CLR nodes as oblique arguments

Catch-All and Clean-Up phase corrects the overgeneralisation and rewrites the annotation of the second NP argument (the indirect object) as ↑OBJ2=↓.

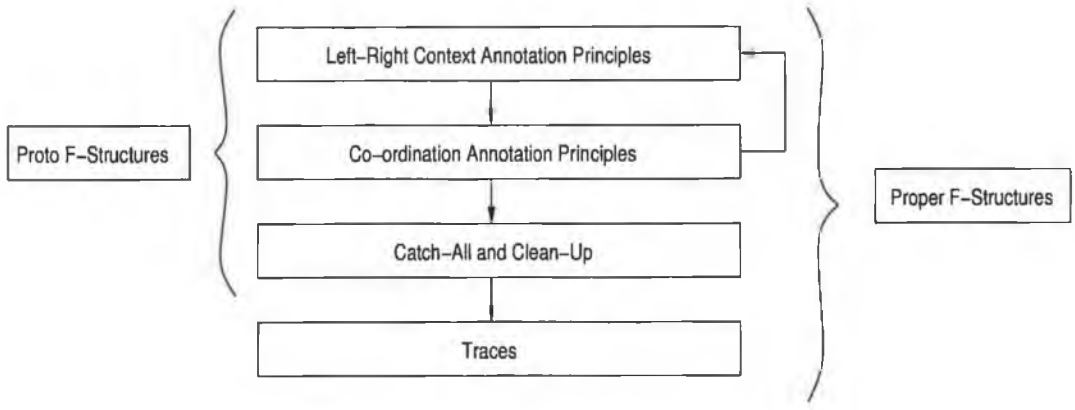


Figure 2.7: Outline of algorithm to generate proto and proper f-structures.

2.4.2 Proper F-structures

The first three components of the f-structure annotation algorithm generate proto-f-structures with long-distance dependencies unresolved. In order to produce proper f-structures that capture long-distance dependencies such as topicalisation and wh-movement, we exploit trace information encoded in the Penn-II treebank trees. Figure 2.7 outlines the extended annotation algorithm, with an additional "Traces" component that deals with long-distance dependencies.

Trace Information

Non-local dependencies are encoded in the Penn-II treebank by means of traces. Figure 2.8 illustrates a tree containing traces. In this instance, the trace is used to capture A movement (i.e. movement to argument position). The indices on the WHNP-3 and *T*-3 node labels indicate that these constituents are linked. The WHNP should be interpreted semantically where the *T*-3 label is located in the tree. To link these nodes in the automatically generated f-structure, the empty node is assigned the equation ↑SUBJ = ↑TOPICREL, indicating that the pronoun "who" should be semantically interpreted as the subject of the verb *write*. Using trace information we successfully capture linguistic phenomena such as passive, fronted constituents, wh-questions, A and A′ movement. We also annotate arbitrary PRO (a phonetically null subject in non-finite verb constructions where the subject is unknown). The encoding of phenomena such as passive is important in LFG since the "surface-syntactic" grammatical functions such as SUBJ and OBJ differ from "logical" grammatical roles. Figure 2.9 shows an f-structure for the passive sentence *An agreement was brokered by the U.N.*. The surface subject of the sentence is not the logical subject of the verb *broker*, it is in fact the logical object. The logical subject of the verb is *the U.N.* which is the object of the oblique agent of the verb in the f-structure. Given that the f-structure indicates passive voice, however, it is possible to generate the correct logical form for this sentence.

2.4.3 From Annotated Trees to LFG F-Structures

Once we have successfully annotated each node in a tree with f-structure information, we collect the equations and convert them into a PROLOG format. This is done by replacing the UP and DOWN in the node annotations with the required tree node number which is read off the tree. The equations are then passed to a PROLOG constraint solver, which is based on and extends the constraint solver in Gazdar and Mellish (1989). Our constraint solver can deal with equality constraints, disjunction and simple set-valued feature constraints. Since our f-structure annotations currently do not involve disjunctions, there should ideally be one and only one f-structure produced for each set of equations. Figure 2.10 shows the tree of Figure 2.8 after automatic annotation. The equations on each node are read off the

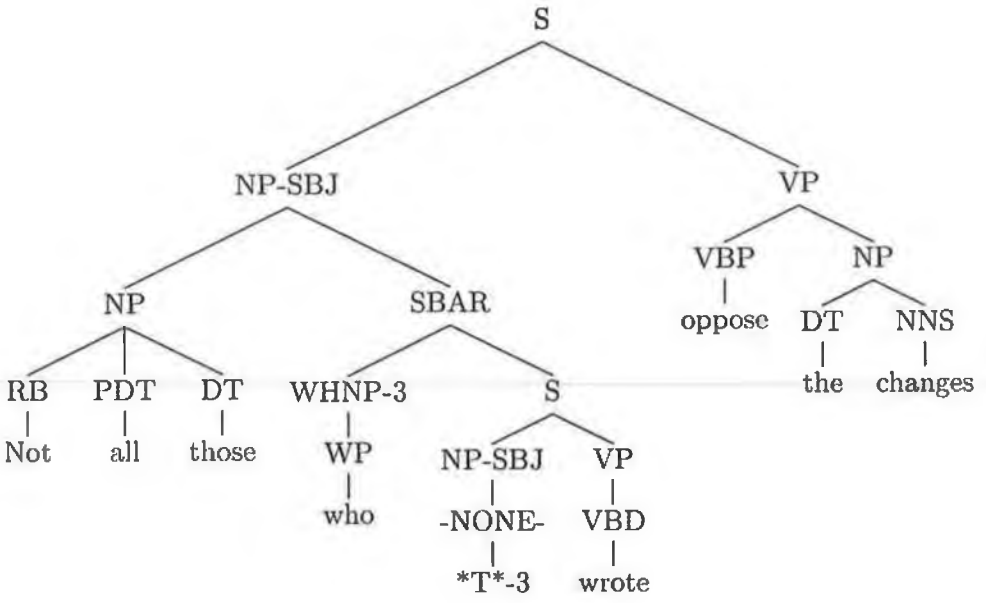


Figure 2.8: A tree containing traces

An agreement was brokered by the U.N.

$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`AGREEMENT'} \\ \text{NUM} & \text{SG} \\ \text{SPEC} & \text{A} \end{bmatrix} \\ \text{PRED} & \text{`BROKER}\langle\uparrow\text{SUBJ}, \uparrow\text{OBL}_{ag}\rangle\text{'} \\ \text{PASSIVE} & + \\ \text{OBL}_{ag} & \begin{bmatrix} \text{PRED} & \text{`BY}\langle\uparrow\text{OBJ}\rangle\text{'} \\ \text{OBJ} & \begin{bmatrix} \text{PRED} & \text{`U.N.'} \\ \text{NUM} & \text{SG} \\ \text{SPEC} & \text{THE} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2.9: The f-structure for the passive sentence *An agreement was brokered by the U.N.*

tree and passed to a constraint solver which generates the f-structure in the same figure.

2.4.4 Evaluation

We provide two methods of evaluating our automatic f-structure annotation algorithm – qualitative and quantitative. I will outline these methods and current results below.

Quantitative Evaluation

Quantitative evaluation effectively evaluates the coverage of our annotation algorithm. We measure this in two ways: f-structure fragmentation and category annotation coverage. When a node in a tree fails to receive an annotation, this results in a separate f-structure fragment for the material below this node. This f-structure is not connected with the f-structure for the rest of the tree. Therefore the more nodes that fail to get an annotation, the more f-structure fragments will be produced for any particular tree. It sometimes happens that no f-structure is produced for an annotated tree. This occurs if there are feature-value clashes (inconsistent annotations) that the constraint solver cannot resolve. Table 2.3 illustrates the progress we have made in reducing the number of f-structure fragments and feature-value clashes since the beginning of the project. Currently over 99.8% of the trees receive one complete f-structure. Only 85 trees do not produce an f-structure because of feature-value clashes. 2 trees produce two f-structure fragments.

# f-str.	(Cahill et al., 2002a)		(Cahill et al., 2002b)		(Cahill et al., 2003b)		current	
frags	# sent	percent	# sent	percent	# sent	percent	# sent	percent
0	2701	5.576	166	0.343	120	0.25	85	0.178
1	38188	78.836	46802	96.648	48304	99.75	48337	99.818
2	4954	10.227	387	0.799	0	0	2	0.004
3	1616	3.336	503	1.039	0	0	0	0
4	616	1.271	465	0.960	0	0	0	0
5	197	0.407	70	0.145	0	0	0	0
6	111	0.229	17	0.035	0	0	0	0
7	34	0.070	8	0.017	0	0	0	0
8	12	0.024	6	0.012	0	0	0	0
9	6	0.012	0	0	0	0	0	0
10	4	0.008	0	0	0	0	0	0
11	1	0.002	0	0	0	0	0	0

Table 2.3: Automatic proto-f-structure annotation fragmentation results

An alternative way to look at the coverage achieved by the automatic f-structure

```
(S
    (NP-SBJ[up-subj=down]
        (NP[up=down]
            (RB[down-elem=up:adjunct] Not[up-pred='not'])
            (PDT[up-spec:det=down] all[up-pred='all'])
            (DT[up=down] those[up-pred='those'])
        )
        (SBAR[up-relmod=down]
            (WHNP-3[up-topicrel=down,up-topicrel:index=3]
                (WP[up=down] who[up-pred=pro,up-pron_form='who'])
            )
            (S[up=down]
                (NP-SBJ[up-subj=down,up-subj=up:topicrel]
                    (-NONE- *T*-3)
                )
                (VP[up=down]
                    (VBD[up=down] wrote[up-pred='write',up-tense=past])
                )
            )
        )
    )
    (VP[up=down]
        (VBP[up=down] oppose[up-pred='oppose',up-tense=pres])
        (NP[up-obj=down]
            (DT[up-spec:det=down] the[up-pred='the'])
            (NNS[up=down] changes[up-pred='change', up-num=pl,up-pers=3])
        )
    )
    (. .)
)
```

```
subj : adjunct : 1 : pred : not
       spec : det : pred : all
       pred : those
       relmod : topicrel : index : 3
                           pred : pro
                           pron_form : who
                subj : index : 3
                       pred : pro
                       pron_form : who
                pred : write
                tense : past
pred : oppose
tense : pres
obj : spec : det : pred : the
      pred : change
      num : pl
      pers : 3
```

Figure 2.10: The tree presented in Figure 2.8 with f-structure equations automatically added to each node. These equations are collected and passed to a constraint solver which generates the f-structure shown.

annotation algorithm is to examine what percentage of each category does not receive an annotation. A summary of the current results is presented in Table 2.4. It shows that there is just one daughter of an S node and one daughter of an SBAR node in the entire treebank that do not receive an annotation.[3] Every other node receives an annotation 100% of the time. This corresponds directly to the fragmentation results presented in Table 2.3, where there are two trees that receive two f-structure fragments. Each tree contains one of the unannotated nodes, leading to a fragmented f-structure.

LHS Category	#Unannotated	#Total RHS nodes	% Annotated
ADJP	0	32162	100
ADVP	0	30712	100
CONJP	0	878	100
INTJ	0	167	100
LST	0	64	100
NAC	0	1270	100
NP	0	817076	100
NX	0	3739	100
PP	0	234009	100
PRN	0	3453	100
PRT	0	3191	100
QP	0	33775	100
RRC	0	113	100
S	1	232159	99.99
SBAR	1	62290	99.99
SBARQ	0	562	100
SINV	0	7669	100
SQ	0	1217	100
UCP	0	1899	100
VP	0	403103	100
WHADJP	0	131	100
WHADVP	0	2615	100
WHNP	0	9650	100
WHPP	0	936	100

Table 2.4: Automatic proto-f-structure annotation coverage results

Tables 2.3 and 2.4 give us an indication of how much of the Penn-II treebank we are able to automatically annotate, but we also need to evaluate how accurate our annotations are: having near 100% coverage is irrelevant if the quality of the annotations is poor. Likewise, obtaining highly accurate annotations is not very useful if the coverage is low.

[3]These two cases involve unusual tree structures, about which we cannot make any generalisations. For more discussion, see Burke (forthcoming).

Qualitative Analysis against the DCU 105

In this section, I describe a set of experiments designed to measure the quality of the f-structures produced by our methodology. The qualitative measures compare the f-structure annotations generated by our automatic annotation procedure against those contained in DCU 105, a manually constructed gold standard set of f-structures for 105 Penn-II trees selected at random from section 23 of the WSJ section of the treebank. Appendix B gives the list of gold standard sentences. The average string length is 23.98 words, with the shortest string 2 words, and the longest 45 words. The trees have been manually annotated, and after a number of iterations, refined to provide a set of complete, correct annotations. The task that our automatic annotation method is confronted with is to match as many of these correct annotations as possible. We use two measures to evaluate the quality of our automatic annotation algorithm: we perform the standard `evalb`[4] test to compare the automatically annotated trees with the manually annotated reference trees, as well as calculating precision and recall on the dependencies computed from the f-structures according to the method and evaluation software presented in Riezler et al. (2002) and Crouch et al. (2002).

`evalb` Evaluation

`evalb` is a bracket scoring program designed by Sekine & Collins which reports precision, recall, non-crossing brackets and tagging accuracy for given data. The main advantage for us in using `evalb` is that it provides a quick and cheap way of evaluating the automatically annotated trees against the manually annotated ones. In order to use `evalb`, we treat a string consisting of a CFG category followed by one or more f-structure annotations (e.g. `VP[up-xcomp=down,up-subj=down-subj]`) as an atomic node identifier. We calculate precision, recall and f-score (harmonic mean), as defined in (2.1), (2.2) and (2.3) respectively. The results of evaluating the automatically annotated 105 sentences against the gold standard are given in Table 2.5. Currently we achieve 81.42% precision, 77.7% recall and 79.52% f-score.

The major disadvantage with using `evalb` is that it is an extremely severe and coarse-

[4] Available at: `http://www.cs.nyu.edu/cs/projects/proteus/evalb/`

grained evaluation metric, in that for any given node the set of equations produced automatically for that node must be identical to the set of manually created annotations, and what is more, they must appear in the same order. For `evalb`, therefore, the annotations `[2,1,3]` and `[1,3,2]` are different (here `1, 2` and `3` represent f-structure equations). Similarly, partial but correct annotations (e.g. `[1,3]` against `[1,2,3]`) are scored as full mistakes by `evalb`.

Some good results may not be recognised by `evalb`. Given this, in the next section we calculate precision and recall *directly* on descriptions of f-structures, or rather dependency relations derived from f-structures.

Bracketing Recall	81.42
Bracketing Precision	77.70
F-Score	79.52

Table 2.5: The result of evaluation using `evalb`

$$\text{Precision} = \frac{\text{number of correct constituents in proposed parse}}{\text{number of constituents in proposed parse}} \tag{2.1}$$

$$\text{Recall} = \frac{\text{number of correct constituents in proposed parse}}{\text{number of constituents in treebank parse}} \tag{2.2}$$

$$\text{F-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3}$$

Dependency Structure Evaluation

In order to calculate precision and recall directly on descriptions of f-structures, we use the evaluation methodology and software presented in Crouch et al. (2002) and Riezler et al. (2002). Each f-structure is represented as a set of terms of the form: `relation(argument,argument)`. As an example, consider the sentence *John saw Mary.* From the f-structure in Figure 2.2, we extract a flat set of terms, as in (1).

(1) `subj(see~0, john~1), obj(see~0, mary~2), num(john~1,sg),`
`pers(john~1,3), num(mary~2,sg), pers(mary~2,3),`
`tense(see~0,past)`

We calculate precision, recall and f-score for complete f-structures and preds-only f-structures encoding basic predicate-argument-modifier relations. A preds-only f-structure contains only paths that end in a `pred:value` pair. The results are presented in Tables 2.6 and 2.7. Currently we achieve an f-score of 96.5% on all grammatical functions, and an f-score of 94.21% preds-only against the DCU 105. Table 2.7 presents a breakdown by function of the preds-only evaluation. It shows that, for example, we achieve an f-score of 97% for SUBJ and OBJ and an f-score of 94% for XCOMP. We consider these results to be very promising. They show that our automatic annotation methodology is more often (slightly more) partial than incorrect. Furthermore, these results confirm our hypothesis that many correct automatic annotations are discounted by `evalb`.

	All GFs	Preds only
Bracketing Recall	96.34	93.95
Bracketing Precision	96.67	94.47
F-Score	96.50	94.21

Table 2.6: The result of evaluation using `relation(argument, argument)` against the DCU 105

2.4.5 Implementation

The automatic annotation algorithm was implemented in Java. In order to support maintainability and extensibility of the algorithm, it was modularised as much as possible to keep the linguistic data separate from the particulars of the implementation of the algorithm. Each parse-tree is represented as a recursive `Node` object. This makes it easy to traverse trees, to compute the local context and to add annotations to each node as required. As it was written in Java, the algorithm could easily be integrated into the suite of tools described in Section 3.3.

Dependency	Precision	Recall	F-Score
adjunct	911/997 = 91	911/965 = 94	93
app	19/19 = 100	19/19 = 100	100
comp	89/92 = 97	89/102 = 87	92
coord	176/184 = 96	176/191 = 92	94
det	267/271 = 99	267/269 = 99	99
focus	1/1 = 100	1/1 = 100	100
obj	453/463 = 98	453/468 = 97	97
obj2	1/1 = 100	1/2 = 50	67
obl	45/49 = 92	45/61 = 74	82
obl2	2/2 = 100	2/2 = 100	100
oblag	12/13 = 92	12/12 = 100	96
poss	75/78 = 96	75/83 = 90	93
quant	43/47 = 91	43/61 = 70	80
relmod	42/44 = 95	42/50 = 84	89
subj	400/415 = 96	400/414 = 97	97
topic	12/12 = 100	12/13 = 92	96
topicrel	42/46 = 91	42/52 = 81	86
xcomp	145/161 = 90	145/146 = 99	94

Table 2.7: Precision and recall on preds-only descriptions of f-structures by grammatical function for the DCU 105

2.5 Summary

In this chapter I have outlined the motivation for developing a large treebank resource that has been annotated with functional information approximating to predicate-argument structure. I have presented our algorithm for automatically annotating the Penn-II treebank with LFG f-structure information. Our algorithm is modular, with four main components: Left-Right Context Rules, Coordination, Catch-All and Clean-Up, and Traces. Each component assigns f-structure equations to tree nodes, which are collected and passed to a constraint solver which produces an f-structure. Left-right context annotation matrices are compiled manually and express generalisations based on most frequent rule types. Coordination is treated separately, as this simplifies the statement of the left-right context annotation matrices. The Catch-All and Clean-Up component assigns annotations to unannotated nodes and overwrites some annotations in certain situations. Finally, the Traces component translates traces and coindexation in the trees to corresponding reentrancies in f-structure.

I have presented a number of quantitative and qualitative evaluation methods and

experiments. Quantitative evaluation does not involve a gold-standard, while qualitative evaluation does. Our quantitative methods measure the coverage of our automatic annotation algorithm. In contrast to counting annotated versus unannotated nodes and f-structure fragmentation, measuring unresolvable f-descriptions gives a first indication of the (lack of) quality as opposed to mere coverage of the automatic annotation results.

Quantitative evaluation is cheap and easy to implement. Qualitative evaluation involves the manual construction of a 'gold-standard' fragment (DCU 105) against which the output of automatic annotation is evaluated. We have constructed a reference fragment consisting of 105 manually annotated trees randomly selected from section 23 of the WSJ section of the Penn-II treebank. We have presented two variants for qualitative evaluation. The first reuses the standard `evalb` evaluation software available from the probabilistic parsing community. Evaluation involving `evalb` in this manner is extremely strict, and the results obtained using `evalb` certainly provide lower bounds. In order to provide a more fine-grained evaluation, our second qualitative evaluation variant involves the translation of generated f-structures into a flat set of terms describing test and reference f-structures. Precision and recall are then computed on these term descriptions, following Riezler et al. (2002).

To summarise the particular results obtained in our automatic f-structure annotation experiments to date, 48337 Penn-II sentences (99.81% of the 48,440 trees without FRAG and X constituents) receive a complete f-structure. 85 trees are not associated with any f-structure. Using `evalb`, we obtained 77.7% precision and 81.42% recall. Following Crouch et al. (2002), we calculated precision and recall directly on sets of term descriptions of f-structures. For the preds-only set of equations, we obtain a precision of 94.47%, and recall of 93.95% and 96.67% and 96.34% for all grammatical functions. These results show that our automatic annotation methodology is more often partial than incorrect.

Chapter 3

Tools and Infrastructure

This chapter presents the tools created to support the linguistic development of the automatic annotation algorithm. Tools are required to search and visualise the treebank resource, to support the population of the automatic annotation matrices, to carry out the annotation, and to visualise the output generated by the annotation algorithm. The tool suites are web-based, platform-independent, can be accessed by standard browsers and were developed using Java and Perl. I will first outline the motivation for the development of such tools. I will then describe the tools in more detail, and finally I will state the advantages of the tool suite developed.

3.1 Background and Motivation

In order to successfully write an algorithm to automatically annotate the Penn-II treebank with f-structure information, extra support tools are required. The first stage of the algorithm relies on manually constructed annotation matrices. These matrices contain information about categories and return an annotation based on the category itself, whether it occurs to the left or right of the head of the local subtree, and what its parent category is. For example, Table 2.2 (page 16) gives a simplified sample matrix for NP. The table states *inter alia* that a DT to the left of a head under an NP node should receive the annotation $\uparrow$ SPEC:DET $= \downarrow$.

To facilitate the linguistic work on populating these matrices, we developed the Tree-

bank Tool Suite (TTS)[1] (Cahill and van Genabith, 2002). This allows context-free rule- and yield-based search on the treebank.

Once the linguistic basis for the automatic annotation algorithm is established, the algorithm has to be implemented and applied, and we need to validate and visualise the result of annotating the trees in the Penn-II treebank. To this end we developed the F-Structure Annotation Tools (FSAT).[2] By visualising the result of the automatic annotation algorithm, we can identify problems and mistakes and provide important feedback for the development cycles. Figure 3.1 outlines the development cycles and our tools infrastructure interacting in order to transform the phrase-structure annotated Penn-II treebank into the f-structure-annotated Penn-II treebank. TTS supports the development of the linguistic basis of the automatic f-structure annotation algorithm, while FSAT applies the algorithm and visualises the results. I will outline the different tool suites in more detail in the sections below.

3.2 TTS: Treebank Tool Suite

A treebank is a corpus of parse-annotated text. The Penn-II treebank contains over 1 million words and over 50,000 sentences. It is straightforward to extract the underlying context-free grammar from the treebank following Charniak (1996). It is also possible to extract variants of the underlying grammar. We extract three grammars from the Penn-II treebank. The first is a basic grammar, with empty productions and trace information removed, and all Penn-II functional information tags attached to CFG categories stripped. The second grammar is almost identical, but the functional information supplied in Penn-II is retained. The third grammar is a head-lexicalised grammar which has been extracted using a modified version of Magerman's (1994) head-finding rules (see page 15). The most basic grammar extracted from all WSJ sections of the treebank has 17,034 rule types. Given a particular rule type from this set, we would like to be able to find and view the following information from the treebank:

[1] Available at http://www.computing.dcu.ie/~acahill/tts/
[2] Available at http://www.computing.dcu.ie/research1/servlet/DisplayTree

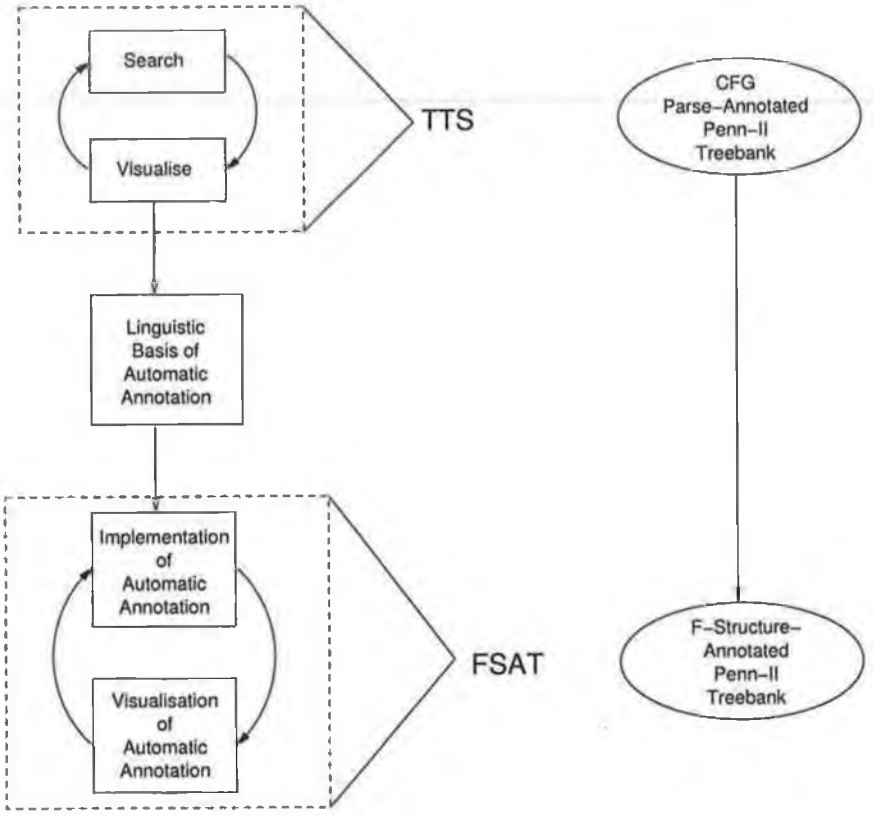


Figure 3.1: Tool suite development cycles

- trees containing instances of the rule,
- subtrees rooted by the rule,
- the strings covered by the rule, with or without their surrounding context.

The treebank is first pre-processed in order to build an index for each context-free rule linking it to rule token occurrences in treebank trees. This is to facilitate efficient runtime retrieval, an important consideration for the end-user. The Interarbora software (Calder, 2000) is used to display treebank trees graphically. Figure 3.2 outlines the architecture of the TTS tool. Grammars are extracted from a treebank and indices for each grammar are generated. The TTS uses these indices to provide the user with the choice of listing rules by frequency or to search and display information about one of the rules listed. If the user chooses a rule, they can display the yield of that rule, display the yield in context, constrain the yield of a rule to be displayed, display the entire trees containing a rule or display only the subtrees rooting a rule.

3.2.1 List Rules by Frequency

It is possible to list all the context-free rules in the extracted grammar by frequency. In addition, the user can choose to display only rules occurring more frequently than a particular threshold (Figure 3.3). It is also possible to select rules with a particular left hand side category only. By clicking on a rule, information about that rule is presented, as described in Section 3.2.2.

3.2.2 Search by Rule

When the user works with the tool (Figure 3.4), they are presented with a number of options. There is a list of context-free rules from which they can select the rule they wish to examine. This list can be further reduced to only display rules with a particular left hand side category. The user also has the option to change the grammar.

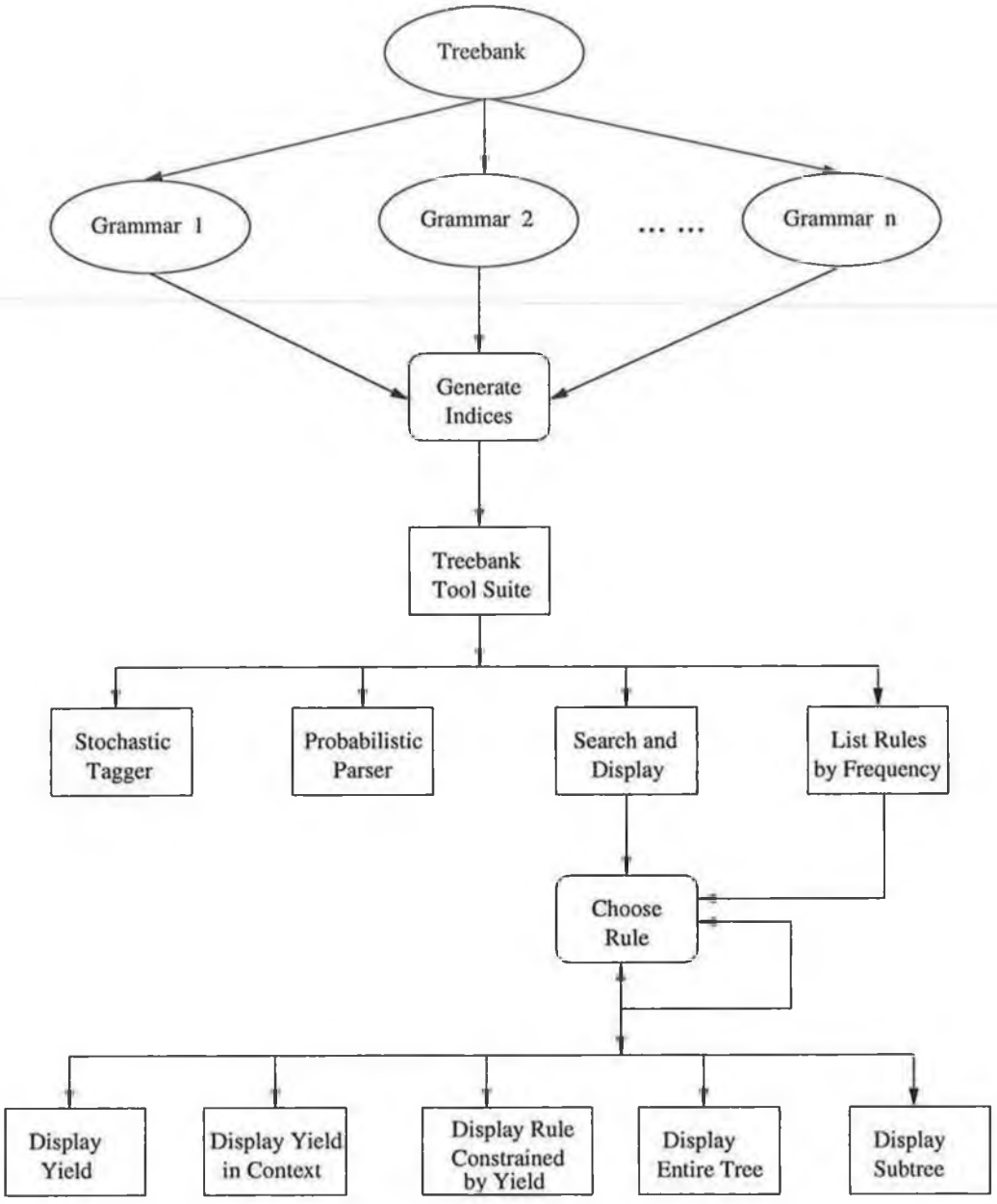


Figure 3.2: Flow chart outline of the TTS tool

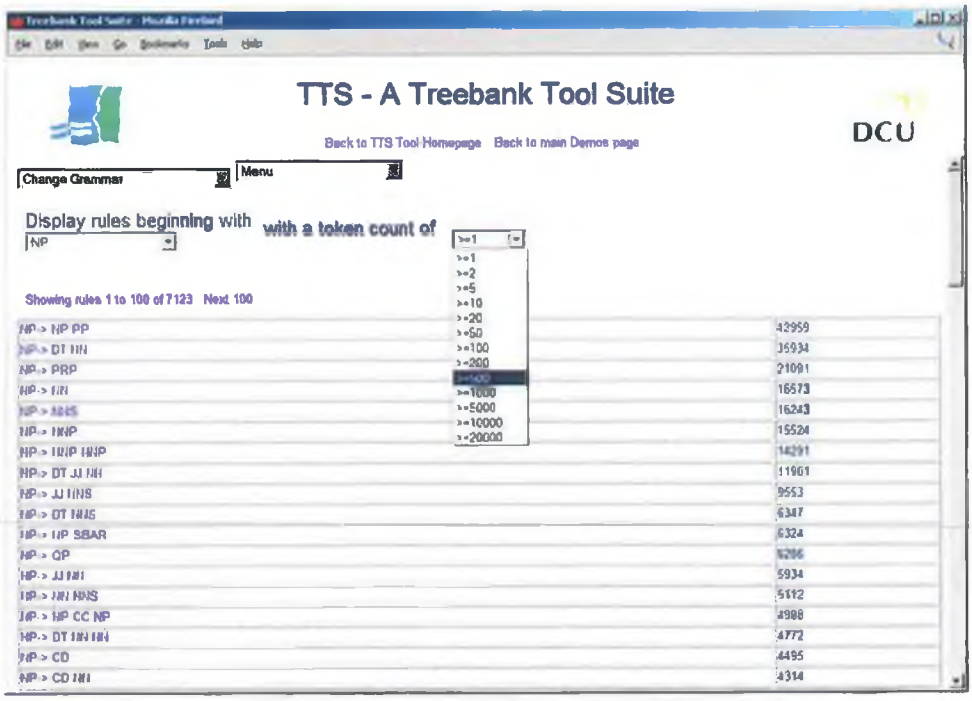


Figure 3.3: Tool to display rules by frequency (using thresholds)

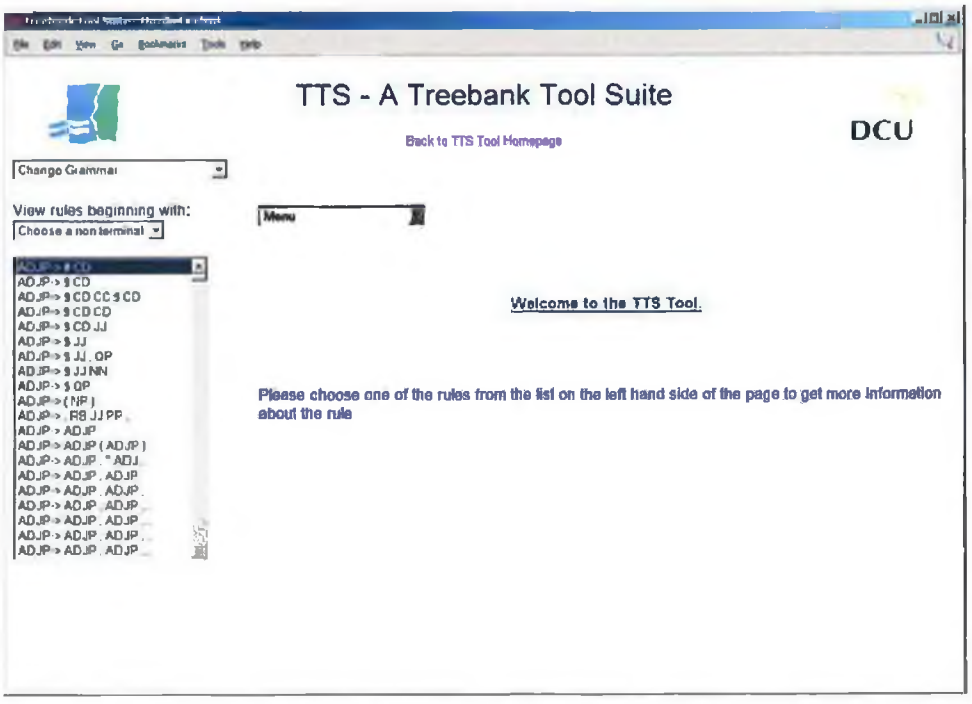


Figure 3.4: The first screen of the Treebank Inspection Tool

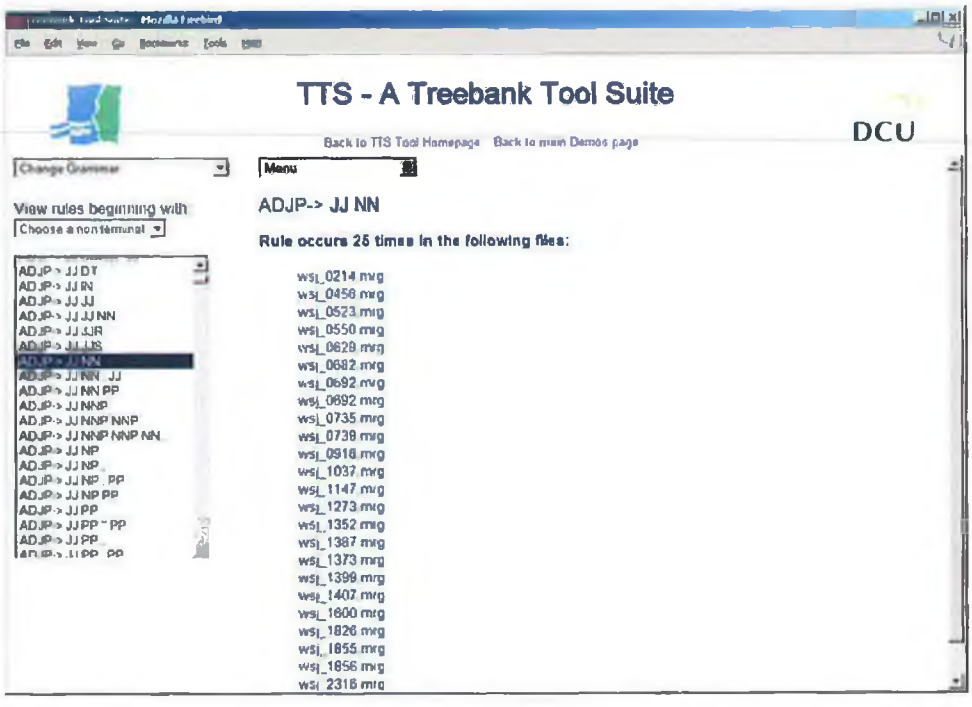


Figure 3.5: The user chooses a context-free rule and obtains information about the files containing trees with instances of that rule in them.

When the user has chosen a context-free rule, they are first presented with a list of files with trees containing instances of the rule (Figure 3.5). They are then presented with the following presentation formats:

- Display trees
- Display subtrees
- Show yield
- Show yield in context
- Show constrained yield

In order to achieve fast response-times, if there are more than 50 solutions to a user's query, they will be retrieved separately and presented in consecutive sets of 50. Figures 3.6, 3.7, 3.8 and 3.9 illustrate choosing the "display trees", "display subtrees", "show yield" and "show yield in context" options respectively. The yield with/without context options effectively turns the tool into a "yield in context" (YIC - KWIC) application. In contrast to `tgrep` (Pito, 1993), the TTS tool does not support arbitrary tree fragment (of variable depth) searches. However, we can often approximate such searches by using a TTS search option involving a context-free rule together with a terminal yield constraint. In order to select trees exhibiting object control constructions, the user can, for example, specify a `VP` $\rightarrow$ `VBD NP S` rule and require that `'asked|asks|ask'` be an element of the yield of the rule. Figures 3.10 and 3.11 illustrate the result of this query.

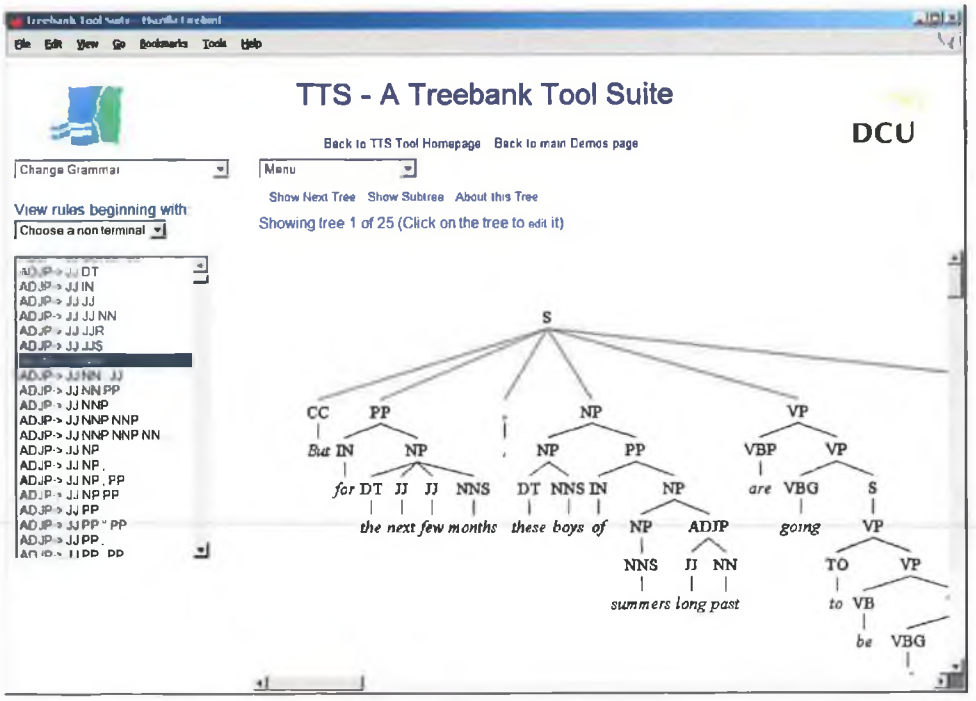


Figure 3.6: Displaying the first tree containing the rule ADJP → JJ NN

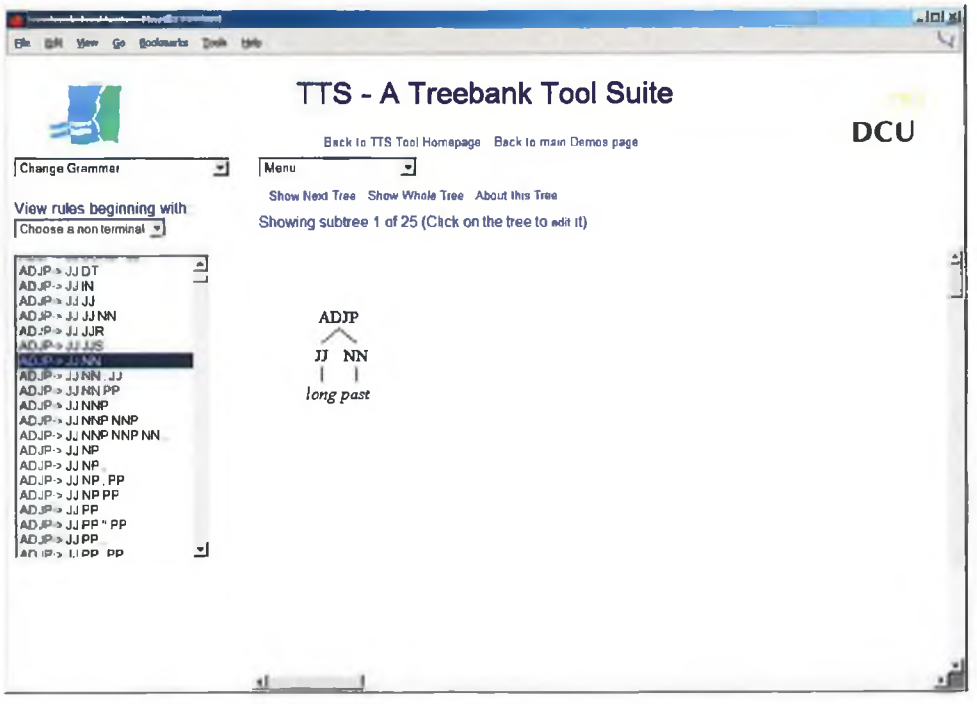


Figure 3.7: Displaying the first tree of the rule ADJP → JJ NN

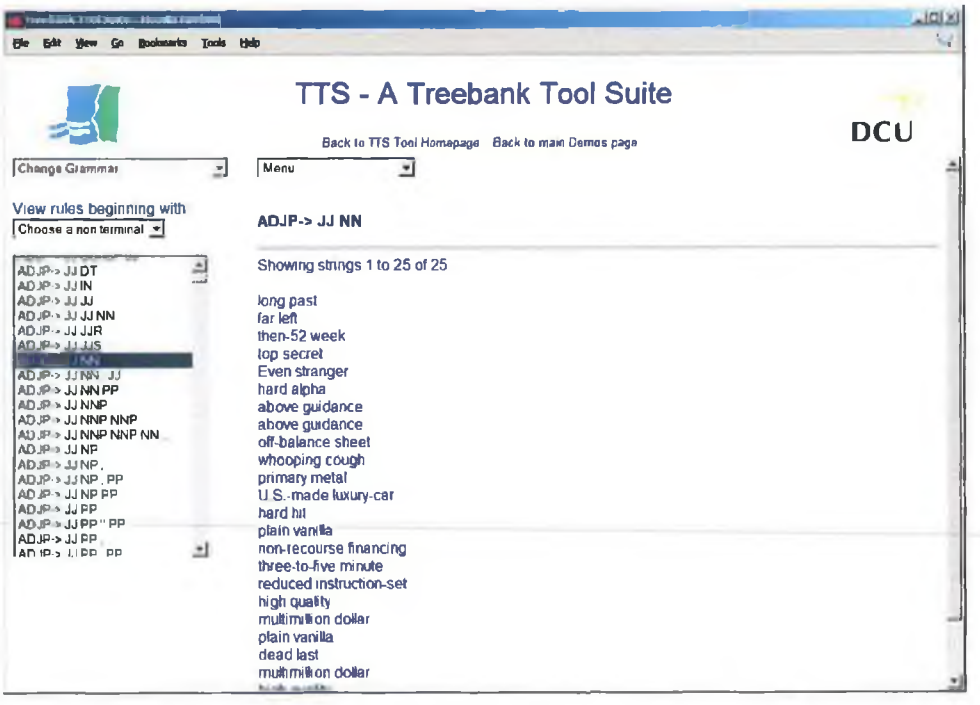


Figure 3.8: Displaying the yield of the rule ADJP → JJ NN

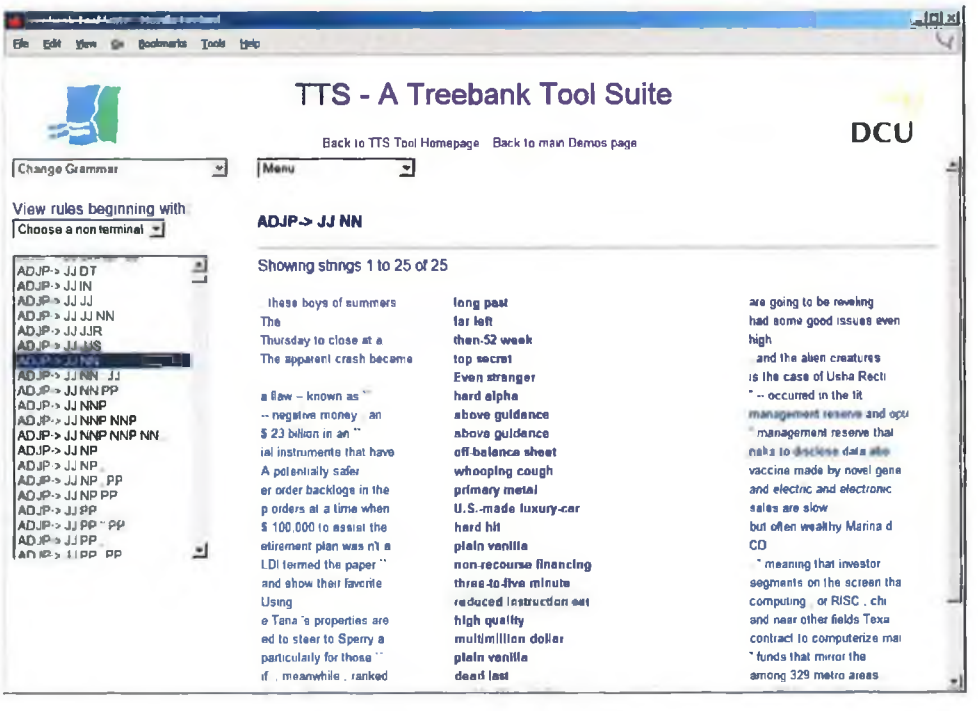


Figure 3.9: Displaying the yield (in context) of the rule ADJP → JJ NN

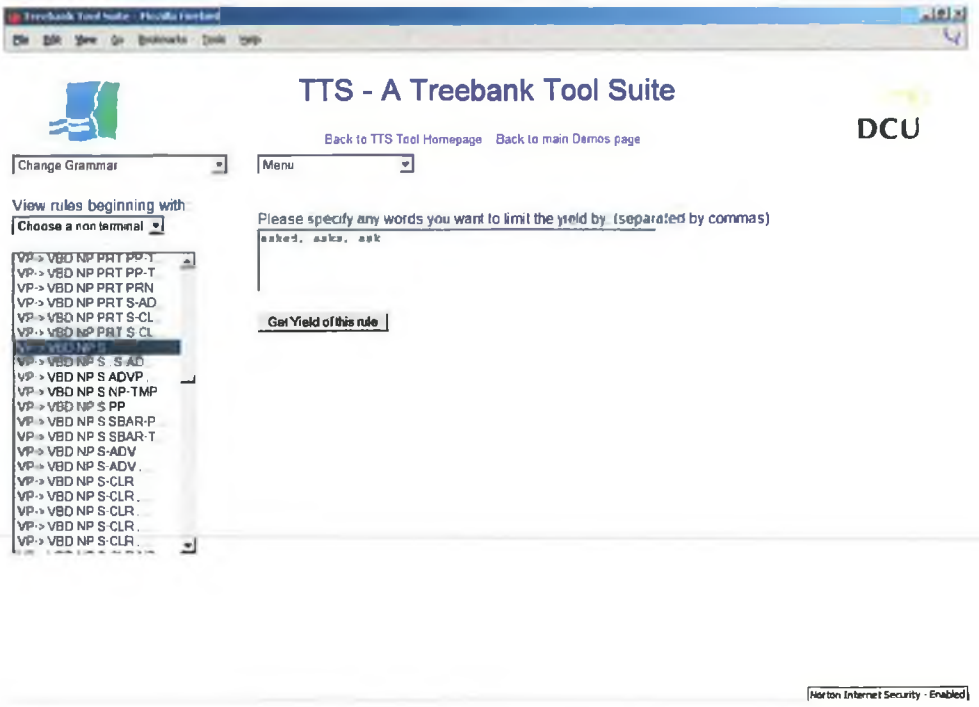


Figure 3.10: Constraining the yield of the rule VP → VBD NP S

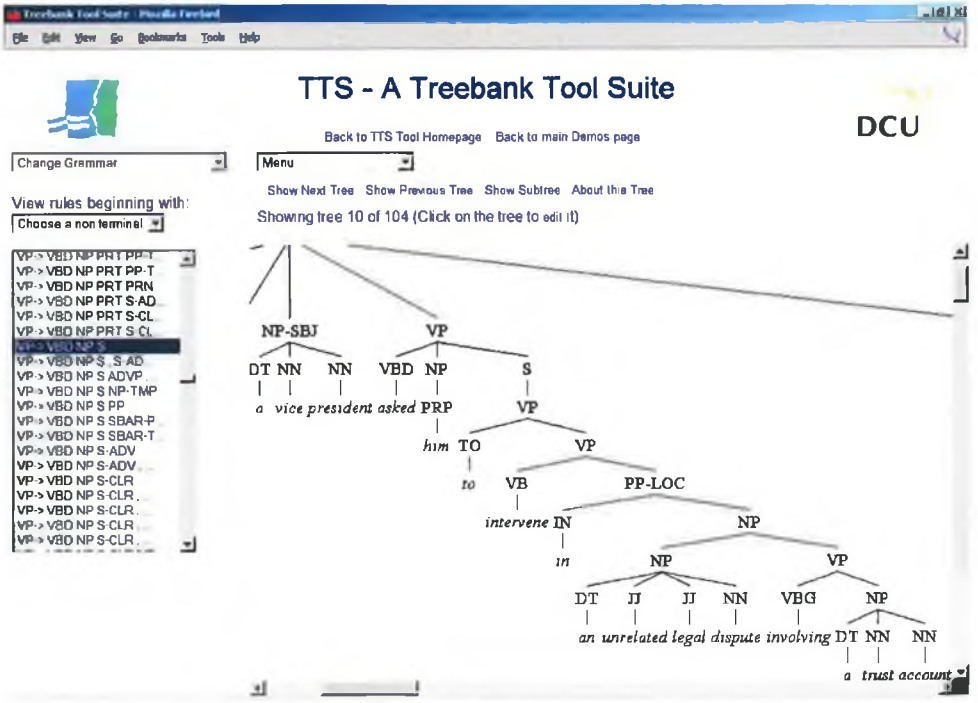


Figure 3.11: Displaying the trees of the yield of the rule VP → VBD NP S limited by the lexeme 'asked'

3.3 FSAT: F-Structure Annotation Tools

Once we began to implement the automatic f-structure annotation algorithm, we needed a tool to apply, visualise and validate the result of automatically annotating Penn-II trees with LFG f-structure information. The FSAT tool presents the following options for each tree in the treebank:

- View lexicalised tree
- View annotated trees
- View f-description equations
- View f-structure
- View subcat frames

The tool presents each tree in the treebank in succession (Figure 3.12). Alternatively, the user can select a particular file from a particular WSJ treebank section. In addition, it is also possible to input a tree manually, by pasting the bracketed form of the tree to be automatically annotated into the indicated text box. The Interarbora software (Calder, 2000) is again used to display trees graphically.

Lexicalised Trees: We use a modified version of the head-finding rules as described by Magerman (1994) (cf. Figure 2.1) to locate the head at each local subtree level. Lexical heads are displayed in brackets on the mother node (Figure 3.13).

Annotated Trees: We run the automatic f-structure annotation algorithm on the tree and display the annotated tree (Figure 3.14).

F-Description Equations: The equations from the annotated tree are collected and printed to the screen (Figure 3.15).

F-structure: The equations are sent to a PROLOG constraint solver, which returns an f-structure for the automatically annotated tree (Figure 3.16).

Semantic Forms: The f-structure is passed to the subcategorisation-frame extraction program as described in Section 6.4.1 and the resulting frames are displayed (Figure 3.17).

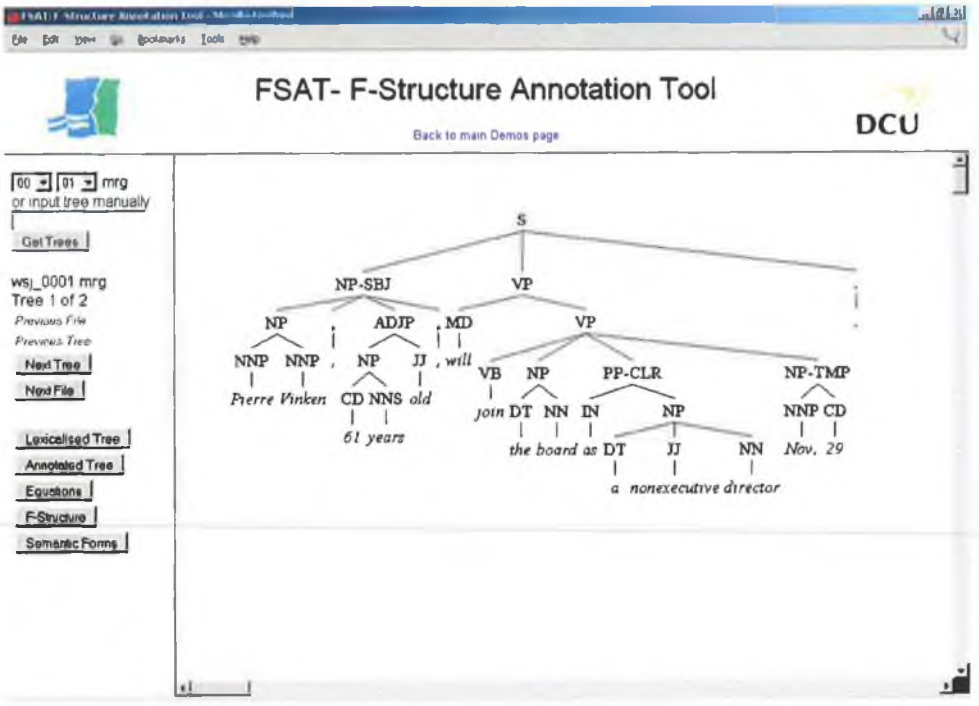


Figure 3.12: FSAT displaying the first tree in the Penn-II treebank.

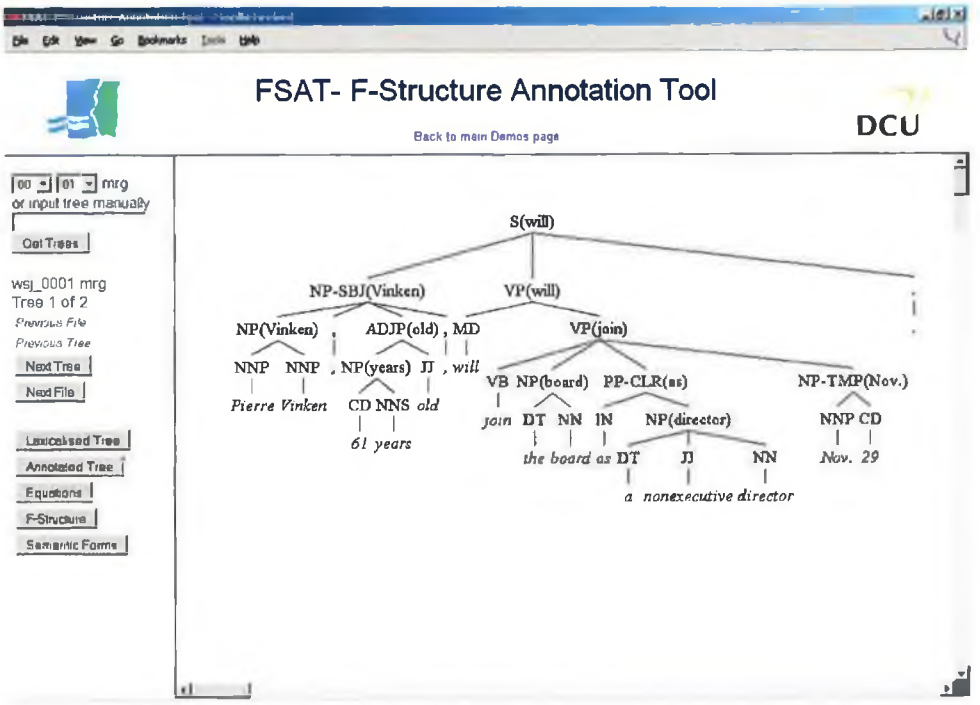


Figure 3.13: The tree in Figure 3.12 after head-lexicalisation

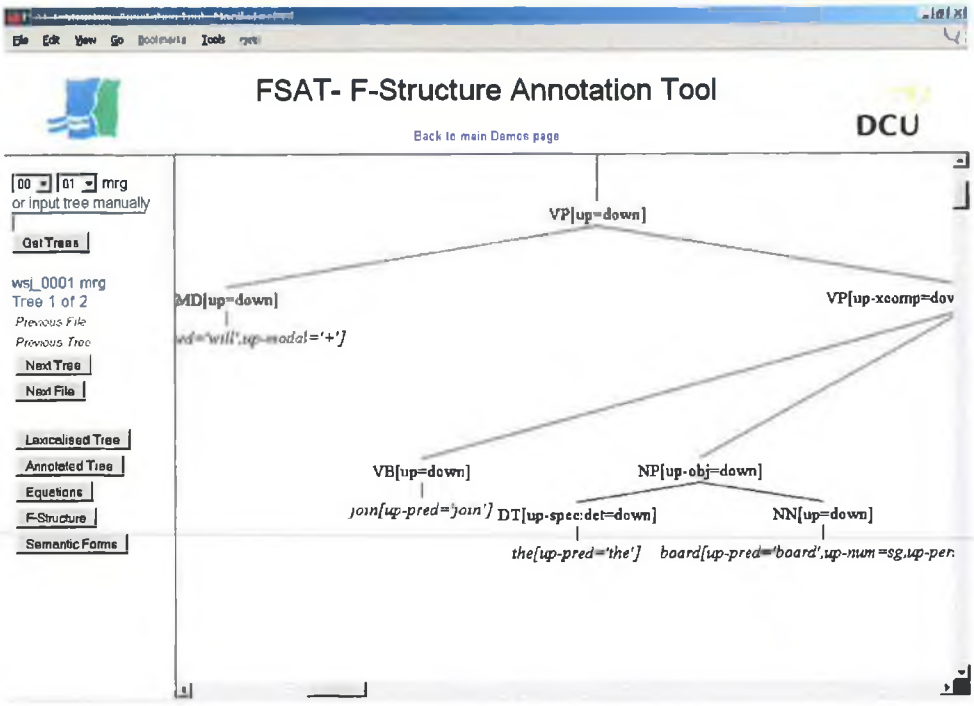


Figure 3.14: A section of the tree in Figure 3.12 after automatically f-structure annotation

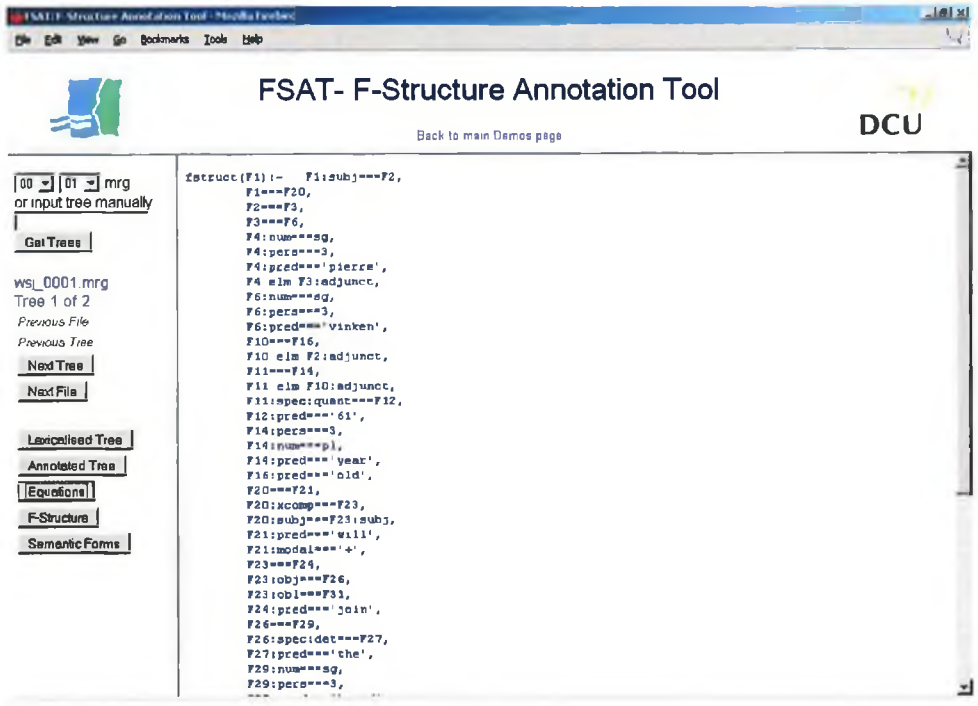


Figure 3.15: The f-description produced by the automatic annotation algorithm for the annotated tree in Figure 3.14

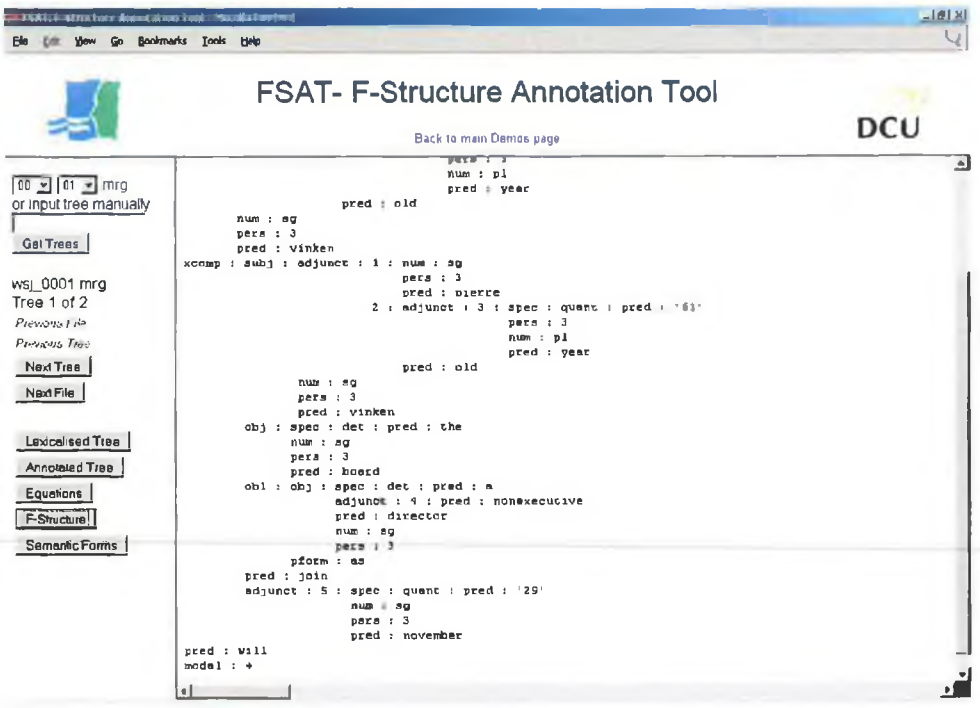


Figure 3.16: The f-structure produced by the automatic annotation algorithm for the f-description of Figure 3.15

3.4 Summary

3.4.1 Design and Implementation

We decided to write our own tool suites, since there were no tools available that matched our needs. In order to support the development of the linguistic basis of the f-structure annotation algorithm, we needed a tool that would allow us to search a treebank for occurrences of context-free rules and the data associated with them. While `tgrep` would have allowed us to perform context-free rule and tree-based searches, it did not display the results graphically or display all terminal yields covered by the rule. We also needed tools that apply and visualise output generated by the automatic f-structure annotation algorithm. We reused existing tree-graphing software (Calder, 2000) which could easily be integrated into our own software. We needed to create a platform-independent, web-based application that would support multiple users. The tools are also required to have fast response-times, since they are designed to be used online.

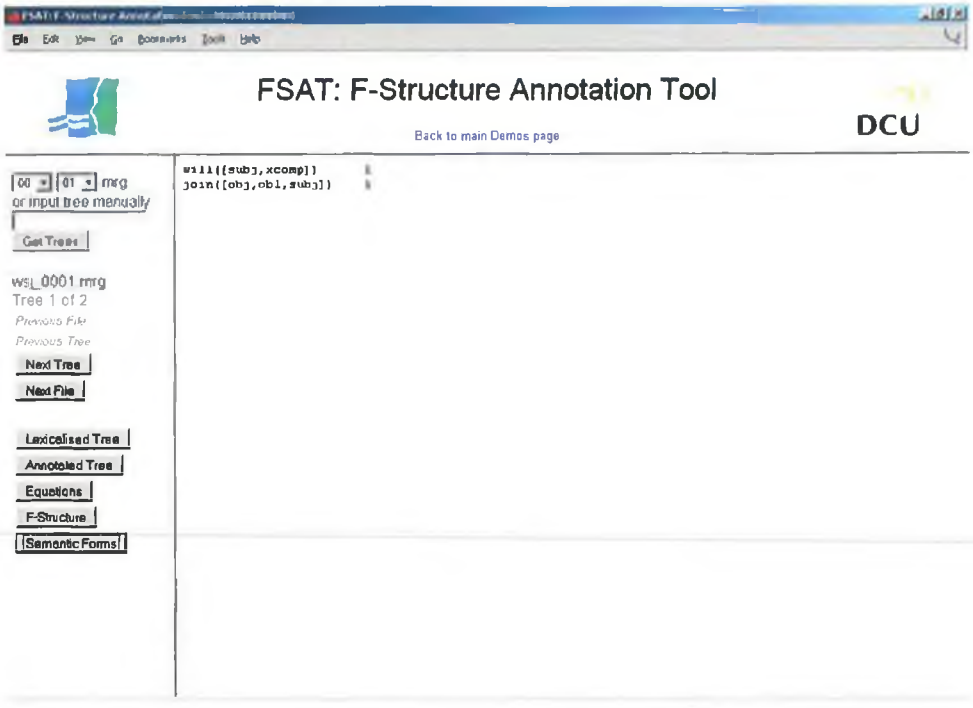


Figure 3.17: The non-empty semantic forms extracted from the f-structure in Figure 3.16

All of the tools were implemented in Java and Perl which made them perfectly suited to a web interface. The tools can be accessed by anybody that has a web browser. We take advantage of the session technology inherent in Java servlets, to allow more than one user to access the tool at the same time. We have installed the Interarbora tree-graphing software (Calder, 2000) locally so that we are not relying on external online resources at run time.

3.4.2 The Advantages of Developing the Tool Suite

The tools described above were essential to the development of the automatic f-structure annotation algorithm. TTS was used to populate the annotation matrices used by the algorithm. Without an efficient method of analysing and visualising the context-free rules underlying the Penn-II treebank, the development of these matrices would have been more tedious and time-consuming. With the treebank inspection tool, we display the most frequent rules, both overall and for individual left hand sides. We concentrated our

efforts on the most frequent rules, assuming that many of the generalisations we gathered would also generalise to the less frequent rules.

It is often obvious what annotations many of the more frequent rules should receive. However, as the frequency of the rule types decreases, it is less often the case. The Treebank Tool Suite allows us to easily find occurrences of rules in the treebank and to compute what terminal strings are covered by the rule. This makes the population of the annotation matrices easier and more efficient. Since we have easy access to actual occurrences of rules, we are less likely to make mistakes when assigning annotations.

Once the initial algorithm is in place, it is essential to be able to verify it and to make sure that there are no unwanted side effects of any changes made during the development cycles. The second set of tools described in Section 3.3 –FSAT– allow us to visualise, search and verify the annotation algorithm. We can see at a glance what annotation has been assigned to each node in the tree. It is much easier to debug a graphical representation of an annotated tree than a plain text version.

Without the tools described, the population of the annotation matrices would have been inefficient and *ad hoc*. The tools allow us to examine the results of the annotation algorithm and identify problems and mistakes. The tools described here proved an essential resource to support the development of the linguistic basis of the automatic annotation algorithm.

Chapter 4

Probabilistic Context-Free Parsing Models

4.1 Introduction

Parsing is an important step in natural language processing, as determining the syntactic structure of a string is important for semantic interpretation in the form of predicate-argument structure, deep dependency relations or logical form. In this chapter I will describe context-free grammars (CFGs), probabilistic context-free grammars (PCFGs) and a simple chart-based algorithm for parsing with such grammars. PCFGs can easily be extracted from treebanks (Charniak, 1996). Given a string, PCFGs order rank parse-trees, whereas a simple CFG can only enumerate all possible parses. CFGs have certain known weaknesses, such as their inability to take lexical information or structural context into account. The corresponding independence assumptions in PCFG models are often too strong. On the other hand, PCFGs are well understood mathematically, easy to implement and can use dynamic programming techniques and Viterbi pruning to support efficient processing. I will outline simple grammar transformation techniques to address some of the main problems of PCFGs, and briefly discuss some more complex parsing models.

4.2 Context-Free Parsing

Context-free parsing is a well-understood technology with a number of efficient algorithms designed for it. In this section I will first outline the properties of context-free grammars, and their probabilistic counterparts. I will then outline a probabilistic version of the CYK algorithm (Younger, 1967; Aho and Ullman, 1972), one of the most common parsing techniques used to parse with PCFGs.

4.2.1 Context-Free Grammars

The classification provided by the Chomsky hierarchy (Figure 4.1) shows that the class of languages defined by context-free grammars is equivalent to the class defined by push-down automata. Formally, a context-free grammar is a four-tuple $\langle \Sigma, V, S, P \rangle$, where:

- Σ is a finite, non-empty set of terminals (the alphabet);
- V is a finite, non-empty set of non-terminal symbols (category labels) such that $\Sigma \bigcap V = \emptyset$;
- $S \in V$ is the start symbol;
- P is a finite set of production rules, $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \bigcup \Sigma)^*$.

Language class	Grammar	Automaton
3	Regular	NFA or DFA
2	Context-Free	Push-Down Automaton
1	Context-Sensitive	Linear-Bounded Automaton
0	Free (Unrestricted)	Turing Machine

Figure 4.1: The Chomsky hierarchy of language classes, grammars and automata.

4.2.2 Probabilistic Context-Free Grammars

Probabilistic context-free grammars extend CFGs by associating a probability with each production rule. Formally, a PCFG is defined as a five-tuple $\langle \Sigma, V, S, P, D \rangle$ where:

- Σ is a finite, non-empty set of terminals (the alphabet);

- V is a finite, non-empty set of non-terminal symbols (category labels) such that $\Sigma \bigcap V = \emptyset$;
- S $\in$ V is the start symbol;
- P is a finite set of production rules, A $\rightarrow \alpha$, where A $\in$ V and $\alpha \in (V \bigcup \Sigma)^*$;
- D is a function assigning a probability to each member of P. Moreover, $\forall A \in V \sum_{A \rightarrow \alpha \in P} D(\alpha \mid A) = 1$.

PCFGs define a language model in terms of probabilities over strings. The model defines the probability of a parse tree T given a string S, i.e. P(T|S). The most likely tree given a string is the tree that maximises P(T|S). It can be observed that maximising P(T|S) is equivalent to maximising P(T,S) since, given S, P(S) is constant. Furthermore, P(T,S) = P(T)P(S|T), but since yield(T) = S, P(S|T) = 1. Thus P(T,S) = P(T):

$$\arg\max_T P(T \mid S) = \arg\max_T \frac{P(T,S)}{P(S)} = \arg\max_T P(T,S) = \arg\max_T P(T) \tag{4.1}$$

Rule expansions in parse trees are independent. Hence, the probability of a tree T is defined as the product of the probabilities of the token occurrences of the rules expanding each left-hand side (LHS) to its right-hand side (RHS) in the tree:

$$P(T) = \prod_{i=1}^{n} P(RHS_i \mid LHS_i) \tag{4.2}$$

Given a treebank, or corpus of parse-annotated text, it is relatively straightforward to extract a PCFG (Charniak, 1996). The probability associated with each rule is determined by relative frequency, by counting the number of times a rule occurs in a corpus and dividing it by the number of occurrences of all rules expanding the same left hand side.

$$P(LHS \rightarrow RHS_j) = \frac{\#(LHS \rightarrow RHS_j)}{\sum_{i=1}^{n} \#(LHS \rightarrow RHS_i)} \tag{4.3}$$

4.2.3 Parsing with Context-Free Grammars

Tabular, memoisation, or chart-based parsers store intermediate results, avoiding the need to recompute identical analyses in different parts of the search space. Active chart parsers,

in addition, store partially explored analyses. A chart is a set of vertices connected by labelled edges. Edges characterise a completed or partial constituent spanning a group of words. An active edge still has constituents to be found, whereas an inactive edge is completed.

The Cocke-Younger-Kasami (CYK) algorithm (Younger, 1967; Aho and Ullman, 1972), developed in 1965, is a chart-based bottom-up dynamic programming algorithm, with complexity $O(n^3|N|^3)$, where n is the number of words in the input sentence, and |N| is the number of non-terminals in the grammar. It requires the input grammar to be in Chomsky Normal Form (CNF), a restricted type of CFG where all of the rewrite rules must be of the form: A $\rightarrow \alpha$ or A$\rightarrow$B C, where A,B,C are non-terminals and α is a terminal symbol. It is straightforward to convert any CFG into CNF. The chart is filled from left to right and from bottom to top. An extended probabilistic version of the CYK algorithm that parses with PCFGs is outlined in Figure 4.2 (Aho and Ullman, 1972; Collins, 1999). The algorithm has three stages: initialisation, a base case and a recursive case. The initialisation stage initialises the dynamic programming array. The base case adds an entry in the chart for each word and its possible category labels. The recursive case builds up the chart p diagonally from bottom to top as follows: for each substring $w_{i,j}$ we look at all possible ways of breaking it into two parts $w_{i,k}$ and $w_{k+1,j}$. We add A to p[i,j] iff:

- A $\rightarrow$ B C is in our set of grammar rules G,
- B $\in$ p[i,k],
- C $\in$ p[k+1,j].

If we can find B and C satisfying those conditions, the probability of adding A to the chart is p[i,k,B] * p[k+1,j,C] * P(A $\rightarrow$ B C)[1]. For Viterbi parsing, if this probability is higher than any entry for A already there, it overwrites that entry. It is also possible to compute the n most probable parses by storing the n most probable entries at each cell. Figure 4.3 shows the completed chart for the parse of the sentence *The man saw Mary*, given the grammar and probabilities indicated. For example, to assign the category NP to p[1,2], the algorithm looks at cells [1,1] and [2,2]. We add p[1,2,NP] = 0.48, since NP $\rightarrow$ DT N is

[1]Where p[x,y,Z] is the probability that the string spanning positions x to y is analysed as the non-terminal Z and P(Y $\rightarrow \alpha$) is the probability assigned to production Y $\rightarrow \alpha$.

```
# Given:
#   Sentence w1...wn
#   Non-Terminals G
#   p is the dynamic programming array.
#   B is an array of backpointers allowing
#   recovery of the highest probability tree

#initialisation

for all i,j,k
    p[i,j,k] = 0

#base case

for i = 1 ... n
    for k = 1 ... G
        if k -> wi is in grammar
            p[i,i,k] = P(k -> wi)

#recursive case

for s = 2...n
    for i = 1 ... n-s+1
    j = i+s-1
    for m = i ... j-1
        for k = 1 ... G
            for k1 = 1 ... G
                for k2 = 1 ... G

                prob = p[i,m,k1] * p[m+1,j,k2] * P(k -> k1 k2)
                if (prob > p[i,j,k])
                    p[i,j,k] = prob
                    B[i,j,k] = {m,k1,k2}
```

Figure 4.2: Pseudocode for the CYK algorithm.

assigned probability 0.6, [1,1] contains DT with probability 1.0 and [2,2] contains N with probability 0.8.

Parsing with PCFGs allows ranking of solutions. However, PCFGs have a certain bias for smaller, less-hierarchical trees. This is due to the fact that the probability of a parse-tree is calculated by multiplying the probabilities of all rules that contribute to it. A larger, more hierarchical tree will include more rules, therefore more rule probabilities to multiply out, possibly resulting in a smaller probability than a less-preferred tree with less structure. Similarly, PCFGs are biased towards non-terminals with fewer expansions over those with many expansions.

Rule	Prob.	Rule	Prob.
S → NP VP	1.0	N → man	0.8
NP → DT N	0.6	DT → the	1.0
NP → PN	0.4	N → saw	0.2
VP → V NP	1.0	V → saw	1.0
		PN → Mary	1.0

	1 the	2 man	3 saw	4 Mary
1	DT = 1.0	NP =0.48		S = 0.1536
2		N = 0.8		
3			N = 0.2 V = 0.8	VP = 0.32
4				PN = 1.0 NP = 0.4

Figure 4.3: A PCFG and the chart for *The man saw Mary.*

4.3 Improving Probabilistic Context-Free Parsing

Standard probabilistic context-free parsing for natural language is limited because the independence assumptions are too strong. The independence assumptions allow us to calculate the probability of a parse tree by computing the product of the rules it uses. This means that the probability of rewriting a nonterminal X with a production R is independent of the previous sequence of rewrites. In order for context-free parsing to yield optimal results, it must take both lexical information and rule context into account. This can be done in a number of ways outlined in this section. In all cases, the basic idea is to complicate category labels, i.e. to encode some contextual information in a local category label. For example, the parent transformation (Johnson, 1999) allows us to encode rule context information. Similarly, head-lexicalisation allows us to encode lexical context.

4.3.1 Grammar Transformations

The independence assumptions of PCFG parsing are very strong, resulting in them being insensitive to much relevant contextual or lexical information. However, there are various ways in which contextual or lexical information can be "smuggled" into PCFGs, increasing their accuracy while maintaining their computational simplicity over other more complex

parsing methods.

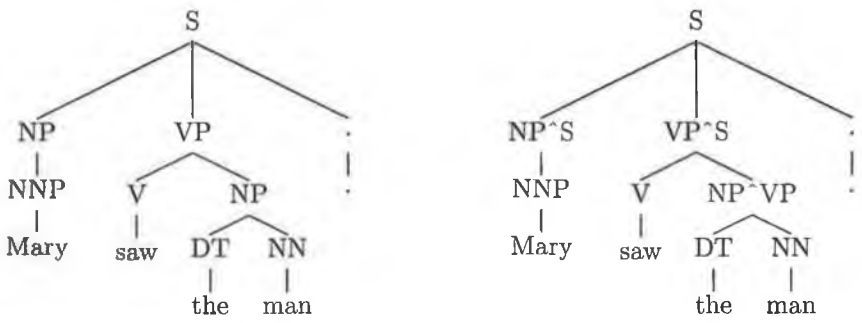


Figure 4.4: Augmenting all non-root, non-preterminal nodes with their parent category label

Johnson (1999) investigates the idea of a "parent-transformation" (credited to Charniak) where each node N of a tree is annotated with its parent category label P to give N^P. The category label NP, for example, becomes NP^S, if it occurs under an S node (Figure 4.4). A training corpus can be transformed automatically in this manner before extracting a parent-annotated grammar. This grammar now has additional contextual information that a basic PCFG does not encode. For example, it is now possible to distinguish between NPs occurring as subjects of a sentence and NPs occurring as objects of a verb: NPs in subject position are daughters of S nodes, NPs in object position are daughters of VP nodes. Subject NPs will be annotated NP^S and object NPs will be annotated NP^VP. Johnson performs experiments on the Penn-II treebank, training on sections 02–21 and testing on section 22. The accuracy of the parser output is measured in terms of labelled precision and recall as defined in equations (2.1) and (2.2) on page 26. The parent transformation achieves labelled precision of 0.8 and labelled recall of 0.792, a significant improvement on the basic PCFG which achieves labelled precision of 0.735 and labelled recall of 0.697. These results show that this transformation is a very simple yet effective method of weakening some of the independence assumptions that cause basic PCFGs to perform poorly.

Klein and Manning (2003) demonstrate that by using simple, linguistically motivated transformations which attenuate false independence assumptions latent in treebank grammars, unlexicalised PCFGs can parse with high accuracy (86.3% f-score). They employ a

number of grammar transformations to achieve this result. The first is Johnson's parent annotation transformation. Next, horizontal history (as opposed to vertical history in the case of the parent transformation) is taken into account by markovising the rules from the head child out following Collins (1999). Another interesting observation is that by taking into account the context in which unary productions occurred (i.e. any non-terminal node with only one child received an extra annotation), accuracy improved by 0.55%. The single most effective way of improving overall results was found to be annotating POS tags with their parent information. This provides key lexical insights into individual word behaviour that was previously unexploited. For example, the distribution of adverbs differs greatly depending on the parent category: *also* and *now* occur under ADVP most often, *not* and *n't* occur most often under VP, *only* and *just* occur most often under NP etc. Annotations were also automatically added for certain determiners, adverbs, prepositions and auxiliary verbs, with further improved overall accuracy. Some functional tags present in the Penn-II treebank (e.g. -TMP) were retained and automatically percolated down to the head of the phrase. Verb phrases were marked as being either finite or non-finite and sentences with empty subjects were given a special annotation. The idea of a base NP as defined in Collins (1999) was also introduced, where all NPs that dominate only pre-terminal symbols were annotated as NP-B. Klein and Manning (2003) show that by performing linguistically motivated transformations on the treebank trees, one can achieve accuracy almost as high as state-of-the-art parsers (Collins, 1999; Charniak, 2000), while staying in the PCFG processing and complexity paradigm.

4.3.2 Lexicalisation

Hindle and Rooth (1993) demonstrate that lexical dependencies are crucial for resolving ambiguities such as PP attachment. However, basic PCFGs do not take lexical information into account. For example, not all verbs can take two NP objects, yet for a simple PCFG, all verbs are equally likely to take two NP objects. One way to overcome this problem is to annotate each phrasal category with its head word (Figure 4.5). However, this immediately leads to sparse data problems, and methods to overcome this have to be devised. Carroll and Rooth (1998) present a system for head-lexicalised PCFG parsing. It

parses using an unlexicalised PCFG and then a modified inside-outside algorithm finds the lexicalised frequencies, simulating lexicalisation of the chart. The context-free framework allows the use of efficient chart parsing techniques while incorporating important lexical dependencies.

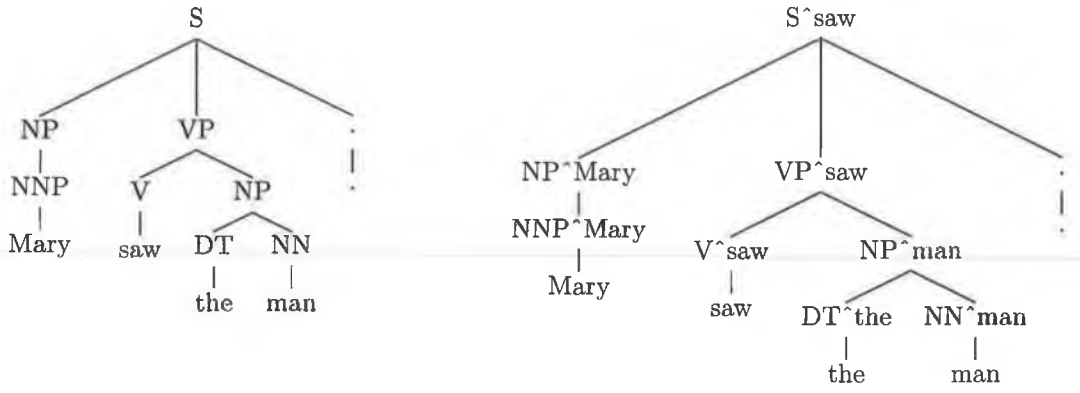


Figure 4.5: A head-lexicalised tree

4.4 Some more complex Approaches to Parsing

There are several other approaches to wide-coverage, probabilistic natural language parsing, some of which achieve better results than simple or current, augmented PCFG-based models. Here I will outline two state-of-the-art parsers: Collins (1999) and Charniak (2000). Both parsers achieve results close to 90% f-score when tested on section 23 of the Penn-II treebank.

Collins (1999) presents three parsing models (1, 2 and 3). Model 1 is a basic history-based model. A history-based model (Black et al., 1992) incorporates a rich context model (where anything that has previously been generated can appear in the conditioning context) and uses decision trees to estimate its parameters. Collins' Model 2 incorporates a distinction between complement and adjunct and Model 3 incorporates traces for wh-movement. Model 1 attempts to overcome the problem of sparse data in lexicalised parsing. This is done by decomposing the generation of the RHS of a rule so that the head constituent is generated first, then the left modifiers are generated, and lastly the right modifiers are generated. A parse-tree is represented as the sequence of decisions

corresponding to a head-centred, top-down derivation of the tree. Table 4.1 outlines the results achieved by each model when evaluated against section 23 of the Penn-II treebank. LP is labelled precision, and LR is labelled recall as defined in equations (2.1) and (2.2) respectively. Model 3 achieves the best results with precision of 88.7 and recall of 88.6 on sentences of length ≤ 40. Model 2 outperforms Model 1 with precision and recall of 88.7 and 88.5 on sentences of length ≤ 40. Each model performs slightly worse overall on sentences of length ≤ 100.

	≤ 40 words		≤ 100 words	
	LP	LR	LP	LR
Model 1	88.2	87.9	87.7	87.5
Model 2	88.7	88.5	88.3	88.1
Model 3	88.7	88.6	88.3	88.0

Table 4.1: Parsing results for Collins' models 1, 2 and 3 against section 23 of the WSJ (Collins, 1999)

The parser presented in Charniak (2000) is a probabilistic generative model which assigns a probability to a parse by a top-down process. A generative model uses the observation that maximising $P(T, S)$ is equivalent to maximising $P(T|S)$ as shown in (4.1). $P(T, S)$ is then estimated by attaching probabilities to a top-down derivation of the tree. A generative model consists of a generative grammar and associated probabilities such that the total probability of the utterances recognised by the grammar sums up to exactly one. Charniak's parser is inspired by a log-linear (or maximum entropy) probability model defined over a set of features. The strength of these models lies in their flexibility and their novel approach to smoothing (Berger et al., 1996). Smoothing is a vital component in any lexicalised parser, since without it, the parser will very quickly run into sparse data problems. Charniak's parser achieves a 13% error reduction over the results in Collins (1997). Table 4.2 presents the results in terms of labelled precision and recall for this parser. It achieves precision and recall of 90.1 on sentences of length ≤ 40, with a slight drop in performance on sentences of length ≤ 100.

I have presented just two instances of alternatives to PCFG parsing using history-based models and probabilistic generative models. Other approaches are documented in the literature including Ratnaparkhi (1999) which implements a maximum entropy model.

	≤ 40 words		≤ 100 words	
	LP	LR	LP	LR
Charniak	90.1	90.1	89.6	89.5

Table 4.2: Parsing results for Charniak's parser against section 23 of the WSJ

Charniak (1997) implements a model which first computes a set of parses and later applies a word-based probability model to choose the most probable parse. Magerman (1995) outlines a statistical decision-tree model which differs from that of Charniak (1997) mainly in the type of probabilities it considers as part of the probability model. A probabilistic LR parser is described by Inui et al. (1997). This model was based on an earlier parser by Briscoe and Carroll (1993) but corrects the probability model. It gains some context-sensitivity by assigning a probability to each LR parsing action according to its left and right context. The Data-Oriented Parsing (DOP) framework combines already-seen tree fragments to build up the most probable parse-tree (Bod and Scha, 2003). The idea behind DOP is that contextual information is explicitly encoded in the tree fragments, addressing a key weakness of basic PCFGs.

4.5 Summary

In this chapter, I have introduced context-free grammars, probabilistic context-free grammars and a simple chart-based algorithm for parsing with them. Parsing with (probabilistic) context-free grammars is efficient, simple and easy to implement. Probabilistic treebank grammars are robust, almost always returning some parse for a given input. PCFGs tend to have certain biases, e.g. in favour of smaller, less hierarchical trees.

The main weakness of PCFGs is their inability to take rule or lexical context into account, since the independence assumptions that define the model are too strong. The simplest method for overcoming some of the weaknesses of PCFGs is to transform the grammar so that it encodes some contextual information. For example, Johnson (1999) augments each node with its parent category label, leading to much improved results over a basic PCFG. Klein and Manning (2003) demonstrate how some very simple linguistically-motivated grammar transformations can lead to a large improvement in the quality of the

parse-trees produced by the parser. Grammar transformations such as these stay within the simple, efficient PCFG parsing paradigm, but with significantly improved results over base-line PCFGs. Lexicalised PCFG parsing tries to overcome the problem of lexical insensitivity of simple PCFG parsing (Hindle and Rooth, 1993; Carroll and Rooth, 1998).

More complex parsing techniques have been developed that do achieve even higher results, most notably those of Collins (1999) and Charniak (2000). They employ history-based and probabilistic generative models. However, these models are complicated to implement, and while source code for them is available, it is not easy to adapt them to different grammars.

In this thesis we will take advantage of the simplicity and efficiency of Viterbi-based PCFG parsing, as well as exploit the improvements that can be gained from simple grammar transformations.

Chapter 5

Treebank-Based PCFG Extraction and Transformation Experiments

5.1 Introduction

Probabilistic context-free phrase-structure grammars (PCFGs) and parsing with such grammars are a core technology used in the present dissertation. In this chapter I present a number of PCFG extraction and transformation experiments. These grammars are used in our PCFG-based LFG approximations presented in Chapter 6. Extracting probabilistic context-free grammars from treebanks is a fairly straightforward task (Charniak, 1996). As discussed in Chapter 4, one of the key weaknesses of PCFGs is their insensitivity to context. However, there are various ways in which PCFGs can take contextual or lexical information into account, without sacrificing any of their computational efficiency or elegance. In this chapter I will present a number of methods of pre-processing and transforming PCFGs and experimental results showing what effect each transformation or pre-processing step has on a baseline grammar. I will then give results of parsing with grammars derived from a number of interacting transformations and pre-processing steps and discuss some of the general patterns we observe.

5.2 Basic Treebank Pre-Processing Prior to Grammar Extraction

PCFGs are often extracted automatically from parse-annotated corpora (treebanks) by reading off and counting rules from treebank trees. Usually, prior to grammar extraction, a number of pre-processing steps are carried out on the treebank. I will first look at the pre-processing originally carried out in Charniak's experiments (Charniak, 1996) and demonstrate what effect each pre-processing step has on a baseline grammar. I will describe different ways in which unary productions can be treated and give experimental results for each.

5.2.1 Original Charniak Pre-Processing Steps

According to Johnson (1999), Charniak (1996) performs the following treebank pre-processing steps in his original treebank-based PCFG parsing experiments:

- Remove all empty nodes and trace elements,
- Delete lexical items,
- Insert a root node,
- Remove all unary productions of the form $X \rightarrow X$,
- Remove all Penn-II functional information (e.g. -SBJ, -TPC labels),
- Replace the POS tags of all auxiliary verbs with AUX and AUXG tags.

We would like to know the contribution of each of the above to the grammars extracted and the parsing results, so we will examine each of them in more detail below.

As a baseline we will take a grammar with minimal pre-processing. To date, simple PCFG-based parsing technology does not support grammars that contain empty productions. The first step, therefore, is to remove them from any grammar induced from the treebank. For the sake of comparison, I will also assume that the WSJ section 23 input to the parser has already been tagged (unless otherwise stated). This ensures that there are no errors introduced at the tagging stage, and the only factors affecting the parsing

	Baseline
# parses	2416
# rules	25452
Labelled Precision	72.32%
Labelled Recall	68.65%
Labelled F-Score	70.44%
Unlabelled Precision	74.57%
Unlabelled Recall	70.79%
Unlabelled F-Score	72.63%
Accuracy	32.74%

Table 5.1: Results of parsing Section 23 with a baseline grammar

results are the parser and the grammars. The baseline grammar has not undergone any pre-processing other than the removal of empty productions and lexical items. It still retains all Penn-II functional labels. All parsing experiments are carried out using the same parser (**BitPar**,Schmid (2004)),[1] trained on sections 02-21 of the WSJ section of the Penn-II treebank and tested on section 23.[2] We evaluate all parse-trees produced by each grammar using the `evalb` software. `evalb` measures labelled precision and labelled recall as defined in equations (2.1) and (2.2) on page 26. Unlabelled precision and recall are calculated in a similar manner, where the correctness of a constituent is measured only by the span of the brackets, ignoring the constituent label.

The results of parsing section 23 of the WSJ with the baseline grammar are presented in Table 5.1. The grammar achieves labelled precision of 72.32%, labelled recall of 68.65% and accuracy (percentage of non-crossing brackets) of 32.74%.

Insert a root node (labelled TOP)

Charniak (1996) added in a root node to the grammar as many of the trees in their version of the treebank did not have a topmost label. Figure 5.1 illustrates this step. Our version of the Penn-II treebank, however, does not have any trees without a topmost label, so the pre-processing we carry out is illustrated in Figure 5.2.

We pre-process the training data from sections 02-21 by adding a TOP label to each tree and parse section 23 with this grammar. The results are shown in Table 5.2. The

[1]There is no explicit function in `BitPar` to allow parsing of tagged input. In order to parse tagged input, we create a dummy lexicon for each sentence with a unique entry for each tag–word pair.

[2]We pre-process section 23 in the same way we pre-process the training corpus to allow us to compare like with like.

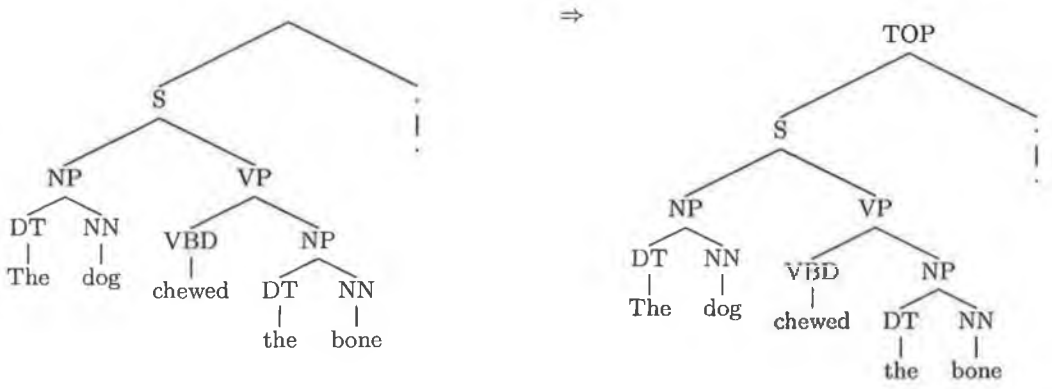


Figure 5.1: Adding a root node label TOP to a tree with no top label

grammar achieves labelled precision of 73.83% (an increase of 1.51%), labelled recall of 70.27% (an increase of 1.62%) and accuracy of 32.74%. Compared to the baseline, the overall improvement in labelled f-score is 1.56%. There is no change in accuracy over the baseline grammar.

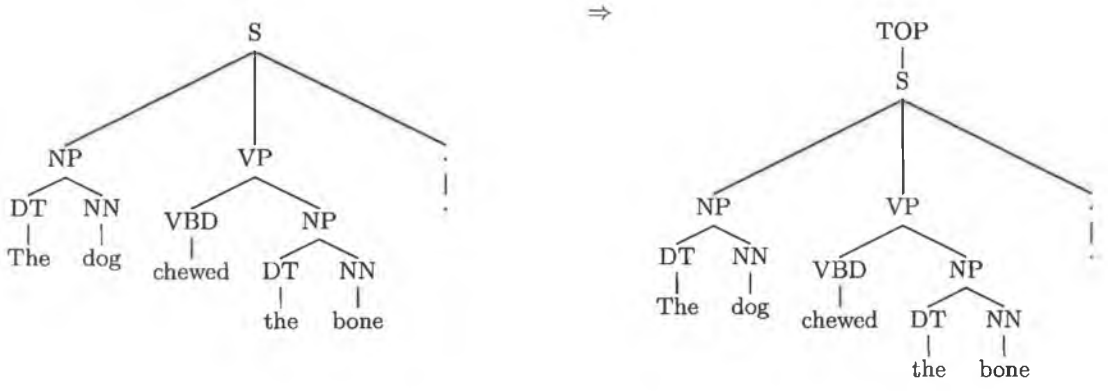


Figure 5.2: Adding a root node label TOP to a tree with a top label

Remove all cyclic unary productions of the form X → X

Cyclic rules of the form X → X result from deleting empty productions. They tend to cause problems for parsing efficiency. If all such rules are removed from the grammar (by collapsing such subtrees in the training corpus), the labelled f-score improves by 0.05% over the baseline. Accuracy increases by 0.04%. The complete results for parsing with a grammar that has no such cyclic unary productions is given in Table 5.3.

	Adding TOP label
# parses	2416
# rules	25489
Labelled Precision	73.83%
Labelled Recall	70.27%
Labelled F-Score	72.00%
Unlabelled Precision	75.96%
Unlabelled Recall	72.30%
Unlabelled F-Score	74.08%
Accuracy	32.74%

Table 5.2: Results of parsing Section 23 with a grammar containing TOP labels

	Removing all X → X
# parses	2416
# rules	25444
Labelled Precision	72.34%
Labelled Recall	68.73%
Labelled F-Score	70.49%
Unlabelled Precision	74.60%
Unlabelled Recall	70.87%
Unlabelled F-Score	72.69%
Accuracy	32.78%

Table 5.3: Results of parsing Section 23 with a grammar that has no cyclic X → X rules

	Deleting Penn-II functional labels
# parses	2416
# rules	14335
Labelled Precision	71.68%
Labelled Recall	66.67%
Labelled F-Score	69.09%
Unlabelled Precision	74.16%
Unlabelled Recall	68.98%
Unlabelled F-Score	71.48%
Accuracy	30.13%

Table 5.4: Results of parsing Section 23 with a grammar without Penn-II functional labels

Remove all Penn-II functional information

The Penn-II treebank contains many functional labels such as -SBJ, -TMP, -LOC. Compared to the baseline grammar, stripping functional labels in the grammar results in fewer rules (14,335 against 25,452). However, the accuracy of parsing with a grammar that discards this functional information decreases by 1.35% over the baseline grammar. Accuracy decreases by 2.61%. The results are shown in Table 5.4. Overall labelled f-score is 69.09%.

Replace the POS tags of all auxiliary verbs with AUX and AUXG tags

The original experiment presented in Charniak (1996) changed the POS tags of all occurrences of the most common auxiliary verbs to AUX and AUXG. Table 5.5 lists the verbs whose POS tags are relabelled to AUX or AUXG. A slight change to this pre-processing step is to only change the POS tags for occurrences of auxiliary verbs where they are followed by another verb. For example, the *have* in *He has two sisters* is not relabelled as an auxiliary, but the *have* in *We have eaten dinner* is. We carry out both changes, with the results as given in Table 5.6. Relabelling all occurrences of auxiliary verbs generates a grammar that produces slightly higher quality trees than the grammar that has only relabelled true auxiliary verbs. This results is perhaps a little surprising since relabelling only true auxiliary verbs seems more linguistically motivated, and therefore, we would expect it to perform slightly better than its counterpart that indiscriminately relabels all occurrences of auxiliary verbs. Compared to the baseline grammar, relabelling all occurrences of auxiliary verbs improves overall labelled f-score by 0.16%, and relabelling true

Verb	New POS tag	Verb	New POS tag
am	AUX	had	AUX
are	AUX	do	AUX
is	AUX	does	AUX
was	AUX	did	AUX
were	AUX	been	AUXG
have	AUX	being	AUXG
has	AUX	having	AUXG

Table 5.5: Auxiliary verbs that receive a new POS tag

	Relabel all occurrences of auxiliary verbs	Only relabel true auxiliary verbs
# parses	2416	2416
# rules	25531	25417
Labelled Precision	72.15%	72.02%
Labelled Recall	69.12%	68.99%
Labelled F-Score	70.60%	70.47%
Unlabelled Precision	74.38%	74.26%
Unlabelled Recall	71.25%	71.14%
Unlabelled F-Score	72.78%	72.67%
Accuracy	32.78%	32.62%

Table 5.6: Results of parsing Section 23 with a grammar that relabels auxiliary verbs

auxiliaries results in an overall improvement over the baseline grammar of 0.03%.

5.2.2 Unary Productions

There are a number of ways of pre-processing grammars with respect to unary productions. Removing all cyclic unary productions of the type X $\rightarrow$ X was outlined above. In the baseline experiment, all unary productions are kept. I will outline two further experiments in this section:

1. Remove all unary productions,

2. Remove all unary productions, but store deleted information.

If we remove all unary productions, it is possible to do simple probabilistic CKY parsing (Younger, 1967; Aho and Ullman, 1972), as the transformation into Chomsky Normal Form (CNF) is trivial. We remove unary productions as follows. For each unary local subtree X dominating Y, X is deleted (as illustrated in Figure 5.3). Alternatively, we could have

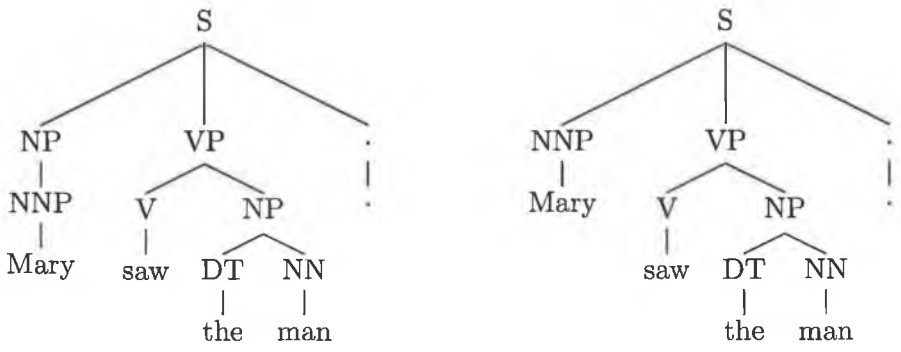


Figure 5.3: Removing unary production NP → NNP

	No unary productions
# parses	2410
# rules	29627
Labelled Precision	73.30%
Labelled Recall	71.89%
Labelled F-Score	72.59%
Unlabelled Precision	75.80%
Unlabelled Recall	74.35%
Unlabelled F-Score	75.07%
Accuracy	36.22%

Table 5.7: Results of parsing Section 23 with a grammar that has no unary productions

deleted Y (the NNP in Figure 5.3). We chose to delete X, the less specific category, so that the more specific category remains. We automatically pre-process the training corpus in this manner and parse section 23. The results of this experiment are given in Table 5.7. Six sentences now fail to get a parse, although labelled f-score improves by 2.15%. A possible contributing factor to this increase is the lower grammar coverage: the six sentences that failed to get a parse are complex, and the baseline grammar, although it successfully assigned a parse to them, possibly did not produce a good parse, therefore lowering overall labelled f-score. In order to test this, we add in a dummy parse (a list of tag–word pairs) for each of the sentences that failed to get a parse. Labelled f-score for the grammar with no unary productions is then 72.53%. This shows that the grammar with no unary productions achieves an overall improvement of 2.09% over the baseline grammar.

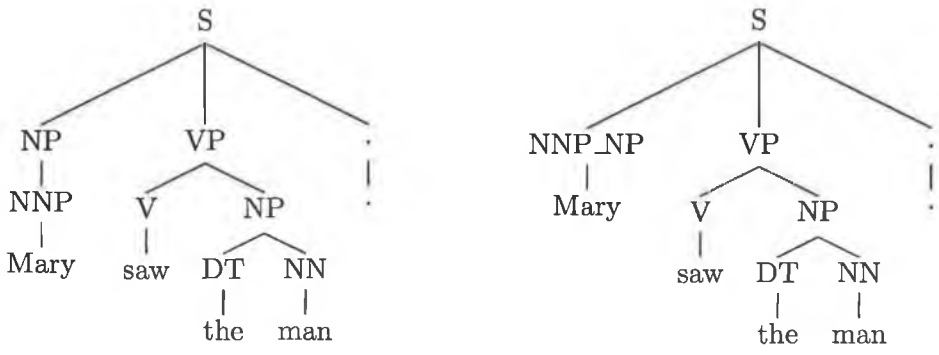


Figure 5.4: Removing unary production NP → NNP

	Unary productions deleted but encoded
# parses	2412
# rules	35979
Labelled Precision	75.50%
Labelled Recall	73.82%
Labelled F-Score	74.65%
Unlabelled Precision	77.65%
Unlabelled Recall	75.93%
Unlabelled F-Score	76.78%
Accuracy	38.31%

Table 5.8: Results of parsing Section 23 with a grammar that has no unary productions, however with new category labels introduced to indicate where the unary productions once were.

Parsing with grammars that retain unary productions yields worse results than parsing with grammars that do not contain unary productions, but valuable information is being discarded when we remove all unary productions. We would like to somehow "remember" where the unary productions occurred. Figure 5.4 illustrates a pre-processing step that does just this. The pre-processed tree has a new category label X_Y, which indicates that there was once a unary production at this point in the tree expanding Y into X. It is straightforward to restore such unary productions in the unary-production-less parse-trees generated by the parser. The results for this experiment are given in Table 5.8. This grammar achieves a labelled f-score of 74.65%, but it fails to produce a parse for 4 sentences which previously received a parse.

Group 1	Group 2
a Add root node **b** No root node	**c** No unary productions **d** No X → X productions **e** Include all unary productions **f** No unary productions, but keep information
Group 3	Group 4
n Keep Penn-II functional labels **o** Remove Penn-II functional labels	**p** No AUX change **q** Change only true auxiliary verbs **r** Change all auxiliary verb labels

Table 5.9: The four groups of pre-processing steps used to test pre-processing interaction. A grammar with one parameter from each group is extracted. This gives 48 grammars.

5.2.3 Combining Pre-Processing Steps

Not all of the grammars presented achieve full coverage. The grammar with no unary productions that has kept the information about unary productions in the category label fails to find parses for 4 sentences, and the grammar with no unary productions and no record of them in category labels is unable to parse 6 of the 2,416 sentences in section 23. If we look at ways in which the pre-processing steps can co-occur, we need to distinguish four distinct groups of pre-processing steps (Table 5.9). This gives 48 possible grammars that can be extracted with these combinations of pre-processing steps. We perform parsing experiments with all grammars. Table 5.10 gives the results for the top 10 grammars according to labelled f-score. None of the grammars in this table achieves complete coverage, and in fact they have all been pre-processed to have no unary productions, but to store the relevant information in category label annotations as in Figure 5.4. Table 5.11 shows the best grammars that achieve full coverage. All grammars in Table 5.11 have a root node label added and the better ones all have Penn-II functional tags. They all have unary productions, though some do not have cyclic unary productions of the form X → X. Table 5.12 gives a summary of the increase or decrease each pre-processing step has over the baseline for labelled and unlabelled f-score, number of parses and the number of rules in each grammar.

Grammar	#Parses	Labelled F-Score (%)	Unlabelled F-Score (%)	Accuracy (%)
afnr	2411	75.46	77.44	44.51
afor	2412	75.43	77.29	42.70
afoq	2412	75.21	77.06	42.21
afnq	2412	75.20	77.16	43.82
afop	2412	75.07	76.92	41.81
bfnr	2412	75.04	77.14	41.59
afnp	2412	75.01	76.98	43.51
bfnq	2411	74.79	76.90	40.92
bfor	2412	74.68	76.84	39.13
bfnp	2412	74.65	76.78	40.79

Table 5.10: Results for the best 10 grammars

Grammar	#Parses	Labelled F-Score (%)	Unlabelled F-Score (%)	Accuracy (%)
adnr	2416	72.18	74.25	35.01
aenr	2416	72.15	74.22	35.01
adnq	2416	72.07	74.15	34.92
adnp	2416	72.05	74.14	35.01
aenq	2416	72.03	74.11	34.88
aenp	2416	72.00	74.08	34.97
adoq	2416	71.09	73.33	32.34
ador	2416	71.06	73.33	32.34
aeoq	2416	71.00	73.25	32.47
aeor	2416	70.94	73.21	32.29

Table 5.11: Results for the best 10 grammars with full coverage

	Increase/Decrease in				
Pre-Processing Step	# Rules	# Parses	Labelled F-Score (%)	Unlabelled F-Score(%)	Accuracy (%)
baseline	25452	2416	70.44	72.63	32.74
a	+37	0	+1.56	+1.45	0
c	+4175	-6	+2.15	+2.44	+3.48
d	-8	0	+0.05	+0.06	+0.04
f	+10527	-4	+4.21	+4.15	+5.57
o	-11117	0	-1.35	-1.15	2.61
q	-35	0	+0.03	+0.04	-0.12
r	+79	0	+0.16	+ 0.15	+0.04

Table 5.12: The increase or decrease each pre-processing step has over a baseline grammar

5.3 Grammar Transformations

The pre-processing steps described in Section 5.2 do not address the main weaknesses of PCFGs in any way. The parent and grandparent transformations, however, do add contextual information to category labels, effectively weakening the independence assumptions of PCFGs. In this section I will describe the parent and grandparent transformations (Johnson, 2002). I will also describe a grammar transformation that automatically adds f-structure information to category labels based on the automatic f-structure annotation algorithm presented in Chapter 2. I will present the results of parsing with parent- and grandparent-transformed grammars and with a grammar that is extracted from an f-structure-annotated version of the Penn-II treebank. Finally I will examine the way in which the transformations and pre-processing steps described interact with one another, presenting the results for the best and worst grammars. The full table of results is provided in Appendix C.

5.3.1 Parent/Grandparent Transformations

The parent transformation is attributed to Charniak, but is first explored in detail in Johnson (1999). This transformation involves augmenting each non-root non-preterminal node in the tree with its parent category label as illustrated in Figure 5.5. This particular transformation effectively weakens many of the independence assumptions inherent in PCFGs. It is now possible, for example, to distinguish between subject NPs (occurring under S nodes) and object NPs (occurring under VP nodes). When evaluating the parser, the parse-trees produced by the parser need to be de-transformed. This involves removing the parent information from any parent-annotated nodes. The results of parsing with a parent-transformed grammar are given in Table 5.13. It achieves an overall labelled f-score of 79.28%, an 8.84% improvement over the baseline grammar. Accuracy increases 11.76% to 44.50%.

Similarly, one can automatically transform the training corpus with a grandparent transformation (illustrated in Figure 5.6). The results of parsing with such a grammar are given in Table 5.14. This grammar also performs significantly better, with an overall improvement of 8.44% on labelled f-score over the baseline grammar.

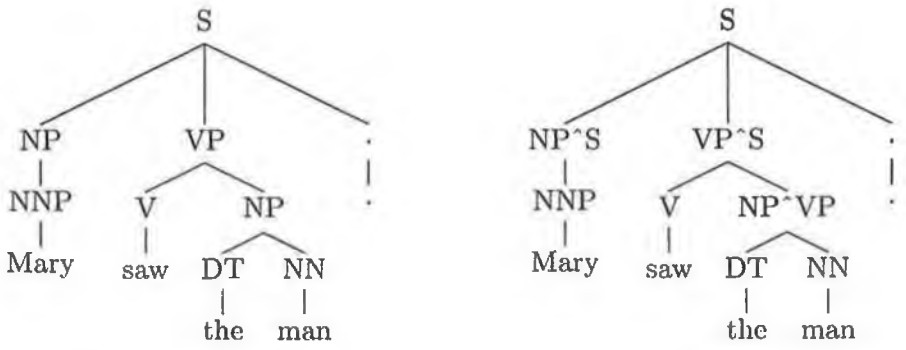


Figure 5.5: Augmenting all non-root, non-preterminal nodes with their parent category label

	Parent Transformation
# parses	2416
# rules	41394
Labelled Precision	79.82%
Labelled Recall	78.74%
Labelled F-Score	79.28%
Unlabelled Precision	81.78%
Unlabelled Recall	80.68%
Unlabelled F-Score	81.22%
Accuracy	44.50%

Table 5.13: Results of parsing Section 23 with a parent-transformed grammar

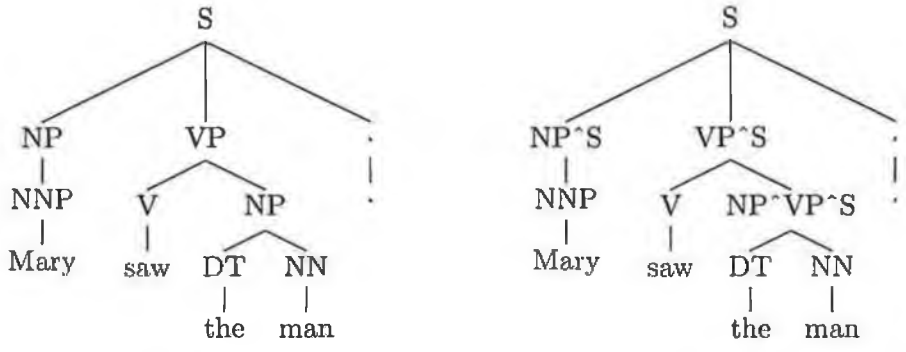


Figure 5.6: Augmenting all non-root, non-preterminal nodes with their grandparent and parent category labels

	Grandparent Transformation
# parses	2416
# rules	67799
Labelled Precision	78.84%
Labelled Recall	78.93%
Labelled F-Score	78.88%
Unlabelled Precision	81.00%
Unlabelled Recall	81.09%
Unlabelled F-Score	81.04%
Accuracy	44.74%

Table 5.14: Results of parsing Section 23 with a grandparent-transformed grammar

	Add F-Structure Information
# parses	2416
# rules	36704
Labelled Precision	76.76%
Labelled Recall	75.39%
Labelled F-Score	76.07%
Unlabelled Precision	78.97%
Unlabelled Recall	77.56%
Unlabelled F-Score	78.26%
Accuracy	41.93%

Table 5.15: Results of parsing Section 23 with an automatically f-structure-annotated grammar

5.3.2 F-Structure-Annotated Rules

The parent and grandparent transformations described in the previous section weaken the independence assumptions of PCFG models of parsing. An alternative method of weakening these independence assumptions is to automatically annotate each node in the tree with f-structure information, using the algorithm outlined in Chapter 2. An annotated PCFG (referred to as A-PCFG) is then extracted where each non-terminal symbol in the grammar has been augmented with LFG f-structure equations, e.g., NP[$\uparrow_{\text{OBJ}}=\downarrow$] $\rightarrow$ DT[$\uparrow_{\text{SPEC}}=\downarrow$] NN[$\uparrow=\downarrow$] . Nodes followed by annotations are treated as a monadic category for grammar extraction and PCFG parsing. To evaluate the parse-trees produced by the parser, all LFG f-equations are deleted after parsing. The results are shown in Table 5.15. This grammar achieves an overall labelled f-score of 76.07, an improvement of 5.63% on the baseline. Accuracy increases 9.19% to 41.93%.

The results presented in Sections 5.3.1 and 5.3.2 show that the parent transformation

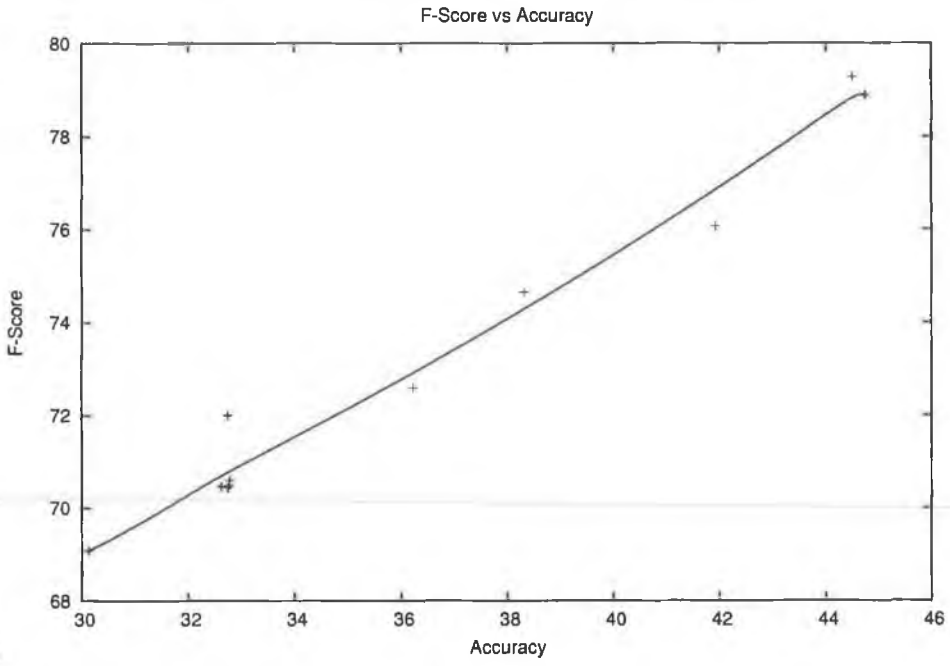


Figure 5.7: The correlation between accuracy and labelled f-score

	Increase in				
Pre-Processing Step	# Rules	# Parses	Labelled F-Score (%)	Unlabelled F-Score(%)	Accuracy (%)
baseline	25452	2416	70.44	72.63	32.74
g	+11252	0	+5.63	+5.63	+9.19
j	+15942	0	+8.84	+8.59	+11.76
k	+42347	0	+8.44	+8.41	+12

Table 5.16: The increase or decrease each pre-processing step has over a baseline grammar

leads to the greatest improvement in parsing results, improving the baseline by 8.84%. The automatically f-structure-annotated grammar also performs significantly better than the baseline with an overall labelled f-score of 76.07. Accuracy correlates closely with labelled f-score as can be seen from the graph in Figure 5.7. Table 5.16 gives a summary of the increase over the baseline that each transformation has for number of rules, parses, labelled and unlabelled f-score and accuracy.

Group 1	Group 2
a Add root node **b** No root node	**c** No unary productions **d** No X → X productions **e** Include all unary productions **f** No unary productions, but keep information
Group 3	Group 4
g Add f-structure annotation **h** No f-structure annotation	**j** Add parent **k** Add grandparent **m** No parent/grandparent transformation
Group 5	Group 6
n Keep Penn-II functional labels **o** Remove Penn-II functional labels	**p** No AUX change **q** Change only true auxiliary verbs **r** Change all auxiliary verb labels

Table 5.17: The six groups of transformations used to test transformation interaction. A grammar with one parameter from each group is extracted. This gives 288 grammars.

5.3.3 Combining Transformations

We want to find out to what extent the grammar transformations and pre-processing steps discussed above interact. To this end, we performed 288 parsing experiments, examining each of the interactions. We grouped the transformation and pre-processing parameters into six different groups (cf. Table 5.17) and each experiment chooses one parameter from each group. [3] Once again we train grammars on sections 02–21 and test the parsing results on section 23. The results for all experiments are given in Appendix C. There were a small number of (extremely large) grammars that caused `BitPar` to produce corrupt data, and therefore we are unable to provide the results for those 21 experiments. Here I will present the results for the best 10 and worst 10 grammars.

Table 5.18 gives the results for the grammars that achieve the highest labelled f-score on the trees produced. Grammar **adgjoq** achieves the highest result, 81.27%. This grammar has a root node inserted, f-structure annotations, parent information on each category label, no X → X cyclic unary productions, no functional labels and only true auxiliary verbs receive the new label AUX. As can be seen from this table, however, none of these grammars achieve full coverage, though they achieve almost full coverage (> 99.8%). Table 5.19 gives the results of the grammars that achieve best highest labelled f-score and that have full coverage. The best grammar with full coverage is **adhjnr** with a labelled f-score

[3] **g** and **h** correspond to integrated and pipeline parsing models respectively, c.f Chapter 6

Grammar	#Parses	Labelled F-Score (%)	Unlabelled F-Score(%)	Accuracy (%)
adgjoq	2413	81.27	83.10	50.18
adgjor	2413	81.25	83.13	50.09
adgjop	2413	81.10	82.95	49.78
aegjor	2414	81.09	83.00	50.11
aegjoq	2414	81.00	82.85	50.07
adgjnr	2412	80.94	82.76	49.26
aegjnr	2413	80.91	82.74	49.33
aegjop	2414	80.81	82.69	49.67
adgjnq	2412	80.74	82.62	49.75
aegjnq	2413	80.66	82.55	49.64

Table 5.18: Results for the best 10 grammars with interacting transformations

Grammar	#Parses	Labelled F-Score(%)	Unlabelled F-Score (%)	Accuracy (%)
adhjnr	2416	80.48	82.30	47.48
aehjnq	2416	80.46	82.30	47.71
adhjnq	2416	80.46	82.31	47.62
aehjnr	2416	80.45	82.27	47.48
adhjnp	2416	80.39	82.21	47.31
aehjnp	2416	80.38	82.21	47.35
aehknr	2416	80.38	82.29	48.11
aehknq	2416	80.34	82.23	47.97
adhknr	2416	80.34	82.25	47.71
adhknq	2416	80.33	82.21	47.66

Table 5.19: Results for the best 10 grammars with interacting transformations and full coverage

of 80.48%. This grammar has a root node inserted, parent information on each node, functional labels, no X $\rightarrow$ X cyclic unary productions, no f-structure annotations and all occurrences of auxiliary verbs receive the new label AUX or AUXG.

Table 5.20 gives the results for the 10 grammars that have the lowest labelled f-score on the trees. The grammar that scores the lowest labelled f-score is **behmop**. This grammar has no root node inserted, has all unary productions, no f-structure annotations, no parent or grandparent information, no functional labels and no changes to the labels on auxiliary verbs. The feature that all grammars in this table have in common is **m**, which means that they have not undergone a parent or grandparent transformation. This indicates that the parent and grandparent transformations lead to higher quality parse trees. They all have full coverage, which might mean that they are being penalised for getting a (bad)

Grammar	#Parses	Labelled F-Score (%)	Unlabelled F-Score (%)	Accuracy (%)
behmop	2416	69.09	71.48	32.34
bdhmop	2416	69.17	71.56	32.20
behmor	2416	69.31	71.70	32.29
behmoq	2416	69.37	71.74	32.47
bdhmor	2416	69.43	71.83	32.34
bdhmoq	2416	69.46	71.83	32.34
behmnp	2416	70.44	72.63	34.97
behmnq	2416	70.47	72.67	34.88
bdhmnp	2416	70.49	72.69	35.01
bdhmnq	2416	70.51	72.71	34.92

Table 5.20: Results for the poorest 10 grammars with interacting transformations

parse for a sentence that another grammar cannot parse at all.

Dummy Parses

A way to measure this is to assign a dummy parse for any sentence that cannot be parsed. The dummy parse is just the list of tag–word pairs linked to a root node. Table 5.21 gives the results for the 10 grammars that have the lowest labelled f-score on trees if a dummy parse is used when the grammar cannot parse a sentence. Similarly, Table 5.22 gives the results for the 10 grammars that have the highest labelled f-score on trees when a dummy parse is inserted if the grammar cannot parse a sentence. These tables show similar patterns, where the better performing grammars all have a parent transformation and have been annotated with f-structure information. The grammars that perform poorest have either undergone no parent or grandparent transformation, or if they have a grandparent transformation, the coverage is poor. Including a dummy parse for unparsed sentences supports better comparison between grammars, since some grammars may yield a high f-score with poor coverage, and may not be as useful as a grammar with slightly lower labelled f-score, but better coverage.

General Trends

Observing the results presented in Appendix C, we can see some general trends in the parsing results. The addition of f-structure information to categories (transformation **g**) gives better results than if this information is omitted. Adding parent or grandparent informa-

Parser	Labelled F-Score (%)	Unlabelled F-Score (%)	Accuracy (%)
afgkor	68.83	71.34	52.86
afgkoq	68.90	71.29	52.52
behmop	69.09	71.48	30.13
bdhmop	69.17	71.56	30.01
behmor	69.31	71.70	30.05
afgkop	69.34	71.73	51.82
behmoq	69.37	71.74	30.26
bdhmor	69.43	71.83	30.13
bdhmoq	69.46	71.83	30.13
behmnp	70.44	72.63	32.74

Table 5.21: Results for the worst 10 grammars with interacting transformations and a dummy parse inserted where no parse is found

Grammar	Labelled F-Score (%)	Unlabelled F-Score (%)	Accuracy (%)
adgjoq	81.22	83.06	47.27
adgjor	81.20	83.09	47.19
adgjop	81.05	82.91	46.85
aegjor	81.05	82.97	47.14
aegjoq	80.96	82.81	47.10
adgjnr	80.88	82.70	46.52
aegjnr	80.84	82.68	46.52
aegjop	80.77	82.65	46.69
adgjnq	80.68	82.56	47.02
aegjnq	80.60	82.50	46.81

Table 5.22: Results for the best 10 grammars with interacting transformations and a dummy parse inserted where no parse is found

tion leads to better parse trees, and combining the parent/grandparent transformations with the addition of f-structure information to category labels, leads to even higher results. Also, the difference in results between the transformations in Group 6 (changing the labels on auxiliary verbs) is very small, suggesting that this transformation has little effect on results. In general, adding in a root node label (transformation **a**) improves the quality of the parse trees produced. Grammars which keep unary productions tend to have better coverage than grammars that delete them. In general, keeping the Penn-II functional tags seems to improve results, though this is to be expected, since these encode a certain amount of contextual information.

5.4 Summary

I have described how the quality of PCFG parsing can be improved by various grammar transformations. I first examined the difference each grammar transformation on its own has over a baseline grammar. The parent transformation (Johnson, 1999) shows the most improvement (8.84%), while the addition of f-structure information to category labels also significantly improves results (by 5.63%). These transformations perform well, since they add contextual information to category labels. Charniak's (1996) pre-processing steps do not affect overall results as much as the parent/grandparent/f-structure annotation transformations, though most pre-processing steps lead to improved parse tree quality. In particular, although labelled f-score increases, coverage decreases for some of the unary production transformations. I carried out a number of experiments to examine the way in which the grammar transformations and pre-processing steps interact with each other. In general, grammars with both a parent category label and f-structure annotations achieve the highest labelled f-score on trees. However, the same grammars without the f-structure annotations achieve slightly better coverage. Changing the label on auxiliary verbs improves results slightly, although there is very little difference between changing the label only on true auxiliaries and changing the label on all occurrences of auxiliary verbs. The grammars that perform worst have not undergone a parent or grandparent transformation, and have no f-structure information added to category labels. The lowest scoring grammar achieves 69.09% labelled f-score on section 23 trees. This grammar has no root label, all

unary productions, no f-structure annotations, no parent or grandparent transformation, no Penn-II functional labels and no changes to the labels on auxiliary verbs. The grammar that performs best achieves a labelled f-score of 81.27% with 99.88% coverage. This grammar has a root node inserted, no unary productions of the form $X \rightarrow X$, f-structure information added to category labels, parent information added to category labels, no Penn-II functional labels and only the labels on true auxiliary verbs have been changed to AUX. The transformations that have the greatest effect on parsing accuracy add contextual information to category labels, thereby weakening the independence assumptions of the PCFG. Further grammar transformations, such as those discussed in Klein and Manning (2003) will be explored in further research.

Chapter 6

Parsing into F-Structures

6.1 Introduction

A large number of researchers have successfully extracted probabilistic grammars from treebank resources. Few, however, have attempted to automatically derive wide-coverage, rich, constraint-based grammars. In Chapter 2, I outlined an automatic f-structure annotation algorithm. We have used this algorithm to develop two PCFG-based parsing architectures (Cahill et al., 2002c) that parse strings into f-structures. In this chapter, I will present these two architectures, and show how they produce proto f-structures. Proto f-structures encode basic, but possibly incomplete, predicate-argument structures where long-distance dependencies (LDDs) are not resolved. Linguistic material is interpreted purely locally where it occurs in the tree. For many linguistic phenomena, however, there is an important difference between the location of the (surface) realisation of linguistic material and the location where this material should be interpreted semantically. Resolution of such LDDs is therefore crucial in the determination of accurate predicate-argument structure. I present and evaluate an algorithm for resolving LDDs at the level of f-structure (Cahill et al., 2004b), based on finite approximations of LFG functional uncertainty equations (Kaplan and Zaenen, 1989; Dalrymple, 2001). I show that the LDD resolution algorithm improves the quality of f-structures, and assess the accuracy of the algorithm. In order to determine reliable and representative quality assessment of the grammars generated by our methodology, we evaluate against three different gold stan-

dards: the DCU 105 (Cahill et al., 2002a), the automatically generated 2,416 f-structures for the original treebank trees in section 23 of the Penn-II treebank, and the PARC 700 Dependency Bank (King et al., 2003). Currently our best grammars achieve an f-score of 81.24% preds-only and 87.04% all grammatical functions against the DCU 105. Against section 23, our best grammar achieves an f-score of 79.38% preds-only and 85.35% all GFs. Against the PARC 700, our best grammar achieves an f-score of 80.33% on the feature set presented in Kaplan et al. (2004).

6.2 Two Parsing Architectures

With the automatic annotation algorithm described in Chapter 2, we can automatically annotate the Penn-II treebank trees with f-structure information in the form of attribute-value structure equations. We have developed two parsing architectures that allow us to parse raw text into f-structures using PCFG-based approximations of probabilistic LFG grammars. Figure 6.1 illustrates the two models, pipeline and integrated.

6.2.1 The Pipeline Model

In the pipeline architecture, we first extract a PCFG from the *unannotated* training corpus (sections 02–21 of the WSJ section of the Penn-II treebank) to parse new text. The most probable tree associated with a string is passed to the automatic f-structure annotation algorithm. The algorithm assigns f-structure equations to the nodes in the tree. We collect these equations and pass them to a constraint solver which generates an f-structure.

6.2.2 The Integrated Model

In the integrated architecture, we extract a PCFG from the f-structure-*annotated* training corpus (A-PCFG). This generates rules such as: NP[↑OBJ=↓] → DT[↑SPEC=↓] NN[↑=↓]. We treat strings consisting of CFG categories followed by one or more f-structure equations as monadic categories for grammar extraction and parsing. We then parse with the annotated grammar and choose the f-structure-annotated tree with the highest probability. We collect the f-structure equations and pass them to a constraint solver to generate an f-structure.

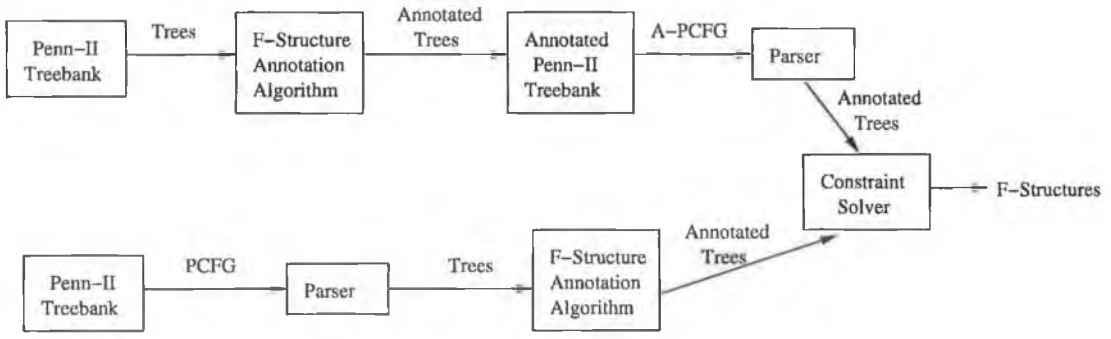


Figure 6.1: Two parsing architectures

6.3 Parsing into Proto F-Structures

Proto F-Structures are basic, but possibly incomplete, predicate argument structures with long-distance dependencies (LDDs) unresolved. Figure 6.2 shows a Penn-II style tree and f-structure for the sentence *U.N. signs treaty the headline said* with co-indexation to indicate the long-distance dependency between the COMP(lement) argument of the verb *say* and the fronted TOPIC(alised) *U.N. signs treaty.* Figure 6.3 shows an incomplete argument structure with an unresolved LDD for the same string. Here, the TOPIC of the sentence is not resolved as the COMP(lement) of *say*, resulting in a proto f-structure. The PCFG technology used in the basic pipeline and integrated architectures described above both parse raw text into trees without traces and empty productions and generate proto f-structures with LDDs unresolved.

6.3.1 Evaluation

We train all grammars on sections 02–21 of the WSJ section of the Penn-II treebank. Following Crouch et al. (2002) and Riezler et al. (2002), we convert f-structures into dependency triple format and use their software to evaluate the quality of the proto f-structures against:

1. The DCU 105 (Cahill et al., 2002a)

2. The full 2416 f-structures *automatically* generated by the f-structure annotation algorithm for the *original* Penn-II trees, in a CCG-style (Hockenmaier, 2003) evaluation experiment.

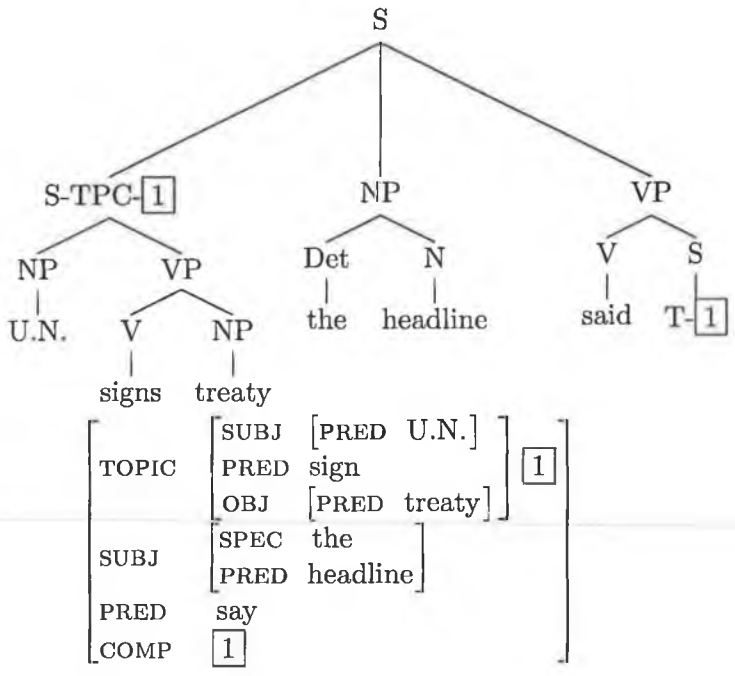


Figure 6.2: Penn-II style tree with LDD trace and corresponding re-entrancy in f-structure

Figure 6.4 shows the conversion of an f-structure into dependency triple format. We evaluate the proto f-structures produced by all grammars described in Chapter 5 against the DCU 105 as well as against the automatically generated f-structures for the full section 23 of the Penn-II treebank. We also measure what percentage of the 2416 sentences receive one covering and connected f-structure (fragmentation). These results are given in Appendices D and E.

Against DCU 105

Table 6.1 gives the results of the 5 grammars that achieve the highest f-score for preds-only proto f-structures in both the pipeline and integrated models. The integrated model performs better than the pipeline model in all cases. The parent-transformed grammars perform better than the other grammars, with 9 out of the 10 grammars in Table 6.1 having a parent-transformation. Also interesting to note is that the inclusion of Penn-II functional labels does not seem to improve results in the integrated model, yet all of the top 5 grammars in the pipeline model have Penn-II functional labels. This is probably because the inclusion of Penn-II functional labels in the integrated model results in more rules and

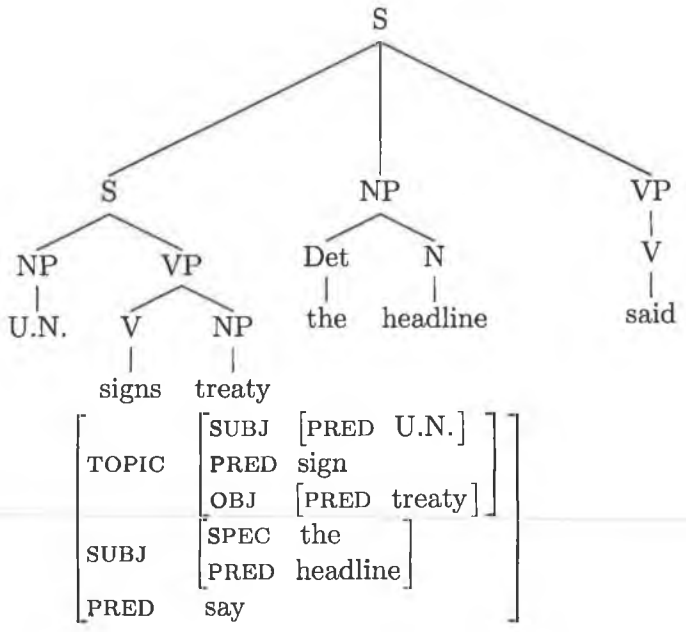


Figure 6.3: PCFG-style parse tree without empty productions and Proto f-structure with unresolved LDD and incomplete argument structure

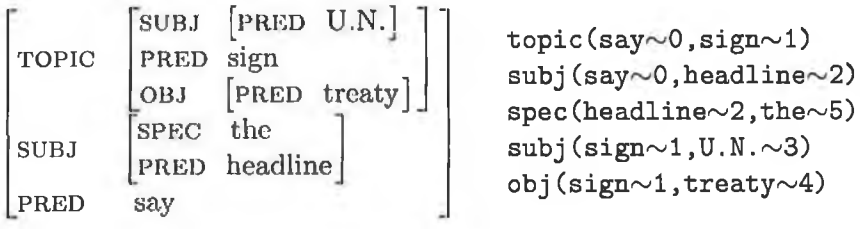


Figure 6.4: Conversion of f-structures into dependency triple format

sometimes the duplication of information (e.g. `NP-SBJ[up-subj=down]`). Also, all of the ten grammars shown have unary productions of some kind as discussed in Section 5.2.2. Removing unary productions degrades parsing performance. Table 6.2 gives a breakdown by function of the preds-only evaluation of grammar **aegjop**.

Against Section 23

We evaluate the proto f-structures produced for all 2416 sentences in section 23 of the Penn-II treebank by the same grammars given in Table 6.1 against the 2416 f-structures *automatically* generated by the f-structure annotation algorithm for the *original* Penn-II trees, in a CCG-style (Hockenmaier, 2003) evaluation experiment. The results are given

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
bfhjnr	74.25	79.81	aegjop	74.80	81.20
afhjnr	74.10	79.74	begjop	74.70	81.21
afhjnp	73.98	81.46	begjoq	74.67	80.14
aehjnr	73.79	79.31	aegjoq	74.41	79.87
bdhjnr	73.79	79.31	bfgmoq	74.36	79.76

Table 6.1: F-score results for the proto f-structures produced by the top 5 grammars in both integrated and pipeline models against the DCU 105

DEP.	PRECISION (%)	RECALL (%)	F-SCORE (%)
focus	1/1 = 100	1/1 = 100	100
topic	12/12 = 100	12/13 = 92	96
det	230/240 = 96	230/269 = 86	90
xcomp	124/144 = 86	124/146 = 85	86
app	14/15 = 93	14/19 = 74	82
coord_form	59/60 = 98	59/85 = 69	81
obj	343/397 = 86	343/461 = 74	80
poss	56/60 = 93	56/81 = 69	79
quant	39/54 = 72	39/52 = 75	74
adjunct	641/804 = 80	641/947 = 68	73
subj	247/271 = 91	247/414 = 60	72
coord	89/120 = 74	89/161 = 55	63
relmod	25/37 = 68	25/50 = 50	57
topicrel	22/26 = 85	22/52 = 42	56
comp	23/30 = 77	23/65 = 35	48
obl	25/51 = 49	25/61 = 41	45
obl_ag	5/11 = 45	5/12 = 42	43
obl2	1/3 = 33	1/2 = 50	40
obj2	0/1 = 0	0/2 = 0	0

Table 6.2: Results for the preds-only evaluation of grammar **aegjop** against the DCU 105 broken down by function

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
bfhjnr	75.67	81.74	aegjop	75.33	82.72
afhjnr	75.81	81.84	begjop	75.31	82.69
afhjnp	75.50	83.20	begjoq	75.39	81.85
aehjnr	75.82	81.91	aegjoq	75.43	81.88
bdhjnr	75.80	81.91	bfgmoq	74.97	81.49

Table 6.3: F-score results for the proto f-structures produced by the grammars in Table 6.1 against the f-structures automatically generated for annotated section 23

DEP.	PRECISION (%)	RECALL (%)	F-SCORE (%)
coord_form	1197/1246 = 96	1197/1461 = 82	88
det	5206/5548 = 94	5206/6327 = 82	88
quant	1054/1134 = 93	1054/1252 = 84	88
poss	1146/1239 = 92	1146/1454 = 79	85
xcomp	2634/3012 = 87	2634/3352 = 79	83
obj	7185/8401 = 86	7185/9538 = 75	80
topic	203/254 = 80	203/269 = 75	78
adjunct	13601/17294 = 79	13601/19918 = 68	73
subj	5420/6214 = 87	5420/9148 = 59	71
coord	1954/2552 = 77	1954/3014 = 65	70
app	262/379 = 69	262/433 = 61	65
topicrel	534/589 = 91	534/1072 = 50	64
focus	5/7 = 71	5/11 = 45	56
relmod	545/872 = 62	545/1084 = 50	56
obl	567/1169 = 49	567/1140 = 50	49
comp	348/646 = 54	348/810 = 43	48
obl_ag	83/211 = 39	83/181 = 46	42
obj2	12/48 = 25	12/44 = 27	26
obl2	14/68 = 21	14/43 = 33	25

Table 6.4: Results for the preds-only evaluation of grammar **aegjop** against section 23 broken down by function

in Table 6.3. In all experiments, results improved over the DCU 105 results by between 1.42% and 2.6% in the pipeline model and between 0.53% and 2.01% in the integrated model. Table 6.4 gives the results broken down by feature for grammar **aegjop**. Overall preds-only score for this grammar improved by 0.53%. Some features improved (e.g. `coord, obj2, topicrel`), while others deteriorated (e.g. `relmod, topic, xcomp`). The results of the grammars that achieve the overall highest preds-only f-score in the integrated and pipeline models are presented in Table 6.5. This table shows that the grammars that achieve the highest scores against the DCU 105 do not score highest against the f-structures generated automatically for section 23. Interestingly, the pipeline model performs better than the integrated model when evaluating against the f-structures generated from the automatically annotated section 23 trees, whereas the integrated model performs better against the DCU 105. The DCU 105 is a small gold standard, and may not be large enough to give us an accurate indication of the performance of each parsing model, since with a small gold standard, there is more risk of overfitting.

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
aehknr	76.13	82.25	aegjor	75.46	81.19
aehknq	76.08	82.93	aegjoq	75.43	81.88
adhknq	76.03	82.91	begjoq	75.39	81.85
adhknr	76.03	82.17	begjor	75.36	81.06
behknq	75.86	82.77	aegjop	75.33	82.72

Table 6.5: F-score results for the proto f-structures produced by the top 5 grammars in both integrated and pipeline models against the f-structures automatically generated for section 23

6.3.2 Fragmentation

We evaluate what percentage of the 2416 sentences parsed produce one covering and connected f-structure (fragmentation). Complete results for all grammars tested are given in Appendix E. Table 6.6 gives the fragmentation results for the 10 grammars in Table 6.1. Almost all sentences receive one covering and connected f-structure in both architectures (over 98.23%). However, knowing what percentage of sentences produce one covering and connected f-structure does not give us a full indication of the coverage of our parsers. We also need to know what percentage of sentences parsed do not produce any f-structure. Grammar A that does not produce as many covering and connected f-structures as Grammar B, may still be producing fragments for the sentences that do not receive a covering and connected f-structure, whereas Grammar B may not be producing any f-structure for these sentences at all. The percentages of sentences receiving no f-structure for the grammars of Table 6.1 are given in Table 6.7. In this instance we see that these grammars either produce one covering and connected f-structure or do not produce any f-structure at all. They do not generate disconnected fragments. This is to be expected, since these grammars are our best ones. If we take a look at the fragmentation results for our poorest performing grammars however, we see that these grammars are in fact producing fragments, though not many. This is perhaps not that surprising, since the fragmentation results for the automatic f-structure annotation algorithm show that over 99.8% of approximately 48,000 original Penn-II trees receive one covering and connected f-structure. In the pipeline model most of the sentences receive one covering and connected f-structure. This is because the f-structure annotation algorithm can look at more context when as-

Pipeline Model		Integrated Model	
Grammar	% Fragmentation	Grammar	% Fragmentation
bfhjnr	98.49	aegjop	99.34
afhjnr	98.23	begjop	99.29
afhjnp	98.45	begjoq	99.21
aehjnr	100.00	aegjoq	99.38
bdhjnr	100.00	bfgmoq	99.25

Table 6.6: Fragmentation results for the top 5 grammars in both integrated and pipeline models against the 2416 sentences in section 23

Pipeline Model		Integrated Model	
Grammar	% 0 F-Structures	Grammar	% 0 F-Structures
bfhjnr	1.51	aegjop	0.66
afhjnr	1.77	begjop	0.71
afhjnp	1.55	begjoq	0.79
aehjnr	0.00	aegjoq	0.62
bdhjnr	0.00	bfgmoq	0.75

Table 6.7: Percentage of sentences producing no f-structure for the top 5 grammars in both integrated and pipeline models against the 2416 sentences in section 23

signing annotations. By contrast, the integrated parsing model is limited to local CFG rules. The annotations on the trees produced by the integrated model are more likely to conflict with each other, since the model cannot take context into account.

6.4 Parsing into Proper F-Structures

For a substantial number of linguistic phenomena such as topicalisation and wh-movement, there is an important difference between the location of the surface realisation of linguis-

Pipeline Model		Integrated Model	
Grammar	% Fragmentation	Grammar	% Fragmentation
behmoq	99.83	afgkor	86.10
aehmoq	99.83	afgkoq	86.23
bdhmoq	99.83	afgkop	87.23
adhmoq	99.83	afgjnq	94.00
behmor	99.88	bfgjnq	93.41

Table 6.8: Fragmentation results for the worst 5 grammars in both integrated and pipeline models against the 2416 sentences in section 23

Pipeline Model		Integrated Model	
Grammar	% 0 F-Structures	Grammar	% 0 F-Structures
behmoq	0.04	afgkor	13.90
aehmoq	0.04	afgkoq	13.77
bdhmoq	0.04	afgkop	12.77
adhmoq	0.04	afgjnq	6.00
behmor	0.04	bfgjnq	6.59

Table 6.9: Percentage of sentences producing no f-structure for the worst 5 grammars in both integrated and pipeline models against the 2416 sentences in section 23

tic material and where this material should be interpreted semantically. The resolution of long-distance dependencies is therefore crucial in the determination of proper predicate-argument structures or deep dependency relations. Theoretically, LDDs can span unbounded amounts of intervening linguistic material as in

$$[\text{U.N. signs treaty}]_1 \text{ the paper claimed } \ldots \text{ a source said } [\ ]_1.$$

In LFG, LDDs are resolved at the f-structure level, obviating the need for empty productions and traces in trees using functional uncertainty (FU) equations (Kaplan and Zaenen, 1989; Dalrymple, 2001). FU equations are regular expressions specifying paths in f-structure between a source (where linguistic material is encountered) and a target (where linguistic material is interpreted semantically). To account for the fronted sentential constituents in Figures 6.2 and 6.3 on pages 83 and 84, an FU equation of the form $\uparrow$ TOPIC $= \uparrow$ COMP* COMP would be required. The equation states that the value of the TOPIC attribute is token identical with the value of the final COMP argument along a path through the immediately enclosing f-structure along zero or more COMP attributes. This FU equation is annotated on the topicalised sentential constituent in the relevant CFG rule as follows

$$\begin{array}{lcccc} \text{S} & \rightarrow & \text{S} & \text{NP} & \text{VP} \\ & & \uparrow\text{TOPIC}=\downarrow & \uparrow\text{SUBJ}=\downarrow & \uparrow=\downarrow \\ & & \uparrow\text{TOPIC}=\uparrow\text{COMP*COMP} & & \end{array}$$

and generates the LDD-resolved proper f-structure in Figure 6.2 for the traceless tree in Figure 6.3, as required.

In addition to FU equations, subcategorisation information is a crucial ingredient in LFG's account of LDDs. As an example, for a topicalised constituent to be resolved as

the argument of a local predicate as specified by an FU equation, the local predicate must (i) subcategorise for the argument in question and (ii) the argument in question must not be already filled. Subcategorisation requirements are provided lexically in terms of semantic forms (subcategorisation lists) and coherence and completeness conditions (all grammatical functions (GFs) specified must be present, and no others may be present) on f-structure representations. Semantic forms specify which GFs a predicate requires locally. For our example in Figures 6.2 and 6.3, the relevant lexical entries are:

$$\begin{array}{lcll} \text{V} & \rightarrow & \text{said} & \uparrow\text{PRED}=\text{say}\langle\uparrow \text{SUBJ}, \uparrow \text{COMP}\rangle \\ \text{V} & \rightarrow & \text{signs} & \uparrow\text{PRED}=\text{sign}\langle\uparrow \text{SUBJ}, \uparrow \text{OBJ}\rangle \end{array}$$

Local completeness requires that all GFs subcategorised for by the local PRED must be present at the local f-structure, while local coherence requires that no other GFs be present (note that adjuncts are non-subcategorisable GFs and thus exempt from these conditions). An f-structure is globally complete and coherent iff all its subsidiary f-structures are locally complete and coherent.

In order to model the LFG account of LDD resolution we require subcategorisation frames (i.e. semantic forms) and LDD resolution paths through f-structure. Traditionally, such resources were hand-coded. Here we show how they can be acquired from f-structure-annotated treebank resources.

6.4.1 Extraction of Semantic Forms

The extraction of semantic forms is described in full detail in O'Donovan et al. (2004). Here we will be brief. LFG distinguishes between governable (arguments) and non-governable (adjuncts) grammatical functions (GFs). If the automatic f-structure annotation algorithm described in Chapter 2 generates high quality f-structures, reliable semantic forms can be extracted (reverse-engineered) (van Genabith et al., 1999): for each f-structure generated, for each level of embedding we determine the local PRED value and collect the governable, (i.e. subcategorisable) grammatical functions present at that level of embedding. For the proper f-structure in Figure 6.2 we obtain the following semantic forms (i.e. lemma followed by subcategorisation frame): `sign([subj,obj])` and `say([subj,comp])`. We extract frames from the full WSJ section of the Penn-II treebank with over 48,000 trees

(without FRAG and X constituents). Unlike many other approaches, our extraction process does not predefine frames, fully reflects LDDs in the source data-structures (cf. Figure 6.2), discriminates between active and passive frames, computes GF, GF:CFG category pair- as well as CFG category-based subcategorisation frames and associates conditional probabilities with frames. Given a lemma l and an argument list s, the probability of s given l is estimated as:

$$\mathcal{P}(s|l) := \frac{count(l,s)}{\sum_{i=1}^{n} count(l,s_i)}$$

Table 6.10 summarises the results. We extract 3586 verb lemmas (root forms) and 10969 unique verbal semantic form types (lemma followed by non-empty argument list). Including prepositions associated with the subcategorised OBLs and particles, this number goes up to 14348. The number of unique frame types (without lemma) is 38 without specific prepositions and particles and 577 with. F-structure annotations allow us to distinguish passive from active frames. Table 6.11 shows the most frequent subcategorisation frames for the lemma `accept`. Passive frames are marked **p**. O'Donovan et al. (2004) carried out a comprehensive evaluation of the automatically acquired verbal semantic forms against the COMLEX resource (Macleod et al., 1994) for the 2992 active verb lemmas that both resources have in common. Here we report on the evaluation of GF-based frames for the full frames with complete prepositional and particle infomation. Relative conditional probability thresholds of 1% and 5% are used to filter the selection of semantic forms (Table 6.12). One reason for the low recall score in Table 6.12 is that the Penn-II treebank is a very specific domain, whereas the COMLEX resource was built from various genres. The other reason is that COMLEX adds a set of 31 default locative prepositional arguments to any verb attested for at least one of them. Doing the same for our experiments increases recall to 40.8% for threshold 1%.

6.4.2 Approximation of Functional Uncertainty Paths

We acquire *finite approximations* of FU-equations by extracting paths between co-indexed material occurring in the automatically generated f-structures from sections 02-21 of the

	Without Prep/Part	With Prep/Part
Lemmas	3586	3586
Sem. Forms	10969	14348
Frame Types	38	577
Active Frame Types	38	548
Passive Frame Types	21	177

Table 6.10: Verb results

Semantic Form	Occurrences	Probability
`accept([obj,subj])`	122	0.813
`accept([subj],p)`	9	0.060
`accept([comp,subj])`	5	0.033
`accept([subj,obl:as],p)`	3	0.020
`accept([obj,subj,obl:as])`	3	0.020
`accept([obj,subj,obl:from])`	3	0.020
`accept([subj])`	2	0.013
`accept([obj,subj,obl:at])`	1	0.007
`accept([obj,subj,obl:for])`	1	0.007
`accept([obj,subj,xcomp])`	1	0.007

Table 6.11: Semantic forms for active and passive occurrences of the verb `accept`

Penn-II treebank. We extract 26 unique TOPIC, 60 TOPICREL and 13 FOCUS path types (with a total of 14,911 token occurrences), each with an associated probability. We distinguish between two types of TOPICREL paths, those that occur in wh-less constructions, and all other types (cf. Table 6.13). Given a path p and an LDD type t (either TOPIC, TOPICREL or FOCUS), the probability of p given t is estimated as:

$$\mathcal{P}(p|t) := \frac{count(t,p)}{\sum_{i=1}^{n} count(t,p_i)}$$

In order to obtain a first measure of how well the approximation models the data, we compute the path types in section 23 not covered by those extracted from sections 02-21: 23/(02-21). There are 3 such path types (Table 6.14), each occuring exactly once. Given that the total number of path tokens in section 23 is 949, the finite approximation extracted from 02-23 covers 99.69% of all LDD paths in the unseen section 23.

	Threshold 1%			Threshold 5%		
	P	**R**	**F-Score**	**P**	**R**	**F-Score**
Exp.	73.7%	22.1%	34.0%	78.0%	18.3%	29.6%

Table 6.12: COMLEX comparison

wh-less TOPICREL	#	*wh*-less TOPICREL	#
subj	5692	adjunct	1314
xcomp:adjunct	610	obj	364
xcomp:obj	291	xcomp:xcomp:adjunct	96
comp:subj	76	xcomp:subj	67

Table 6.13: Most frequent wh-less TOPICREL paths

6.4.3 Long-Distance Dependency Resolution Algorithm

Given a set of semantic forms s with probabilities $\mathcal{P}(s|l)$ (where l is a lemma), a set of paths p with $\mathcal{P}(p|t)$ (where t is either TOPIC, TOPICREL or FOCUS) and an f-structure f, the core of the algorithm to resolve LDDs recursively traverses f. The core of the algorithm is given in Figure 6.5. The algorithm supports multiple, interacting TOPIC, TOPICREL and FOCUS LDDs. We use $\mathcal{P}(s|l) \times \mathcal{P}(p|t)$ to rank a solution, depending on how likely the PRED is to take semantic form s, and how likely the TOPIC, FOCUS or TOPICREL is resolved using path p. The algorithm also supports resolution of LDDs where no overt linguistic material introduces a source TOPICREL function (e.g. in reduced relative clause constructions). We distinguish between passive and active constructions, using the relevant semantic form types when resolving LDDs. The full algorithm also supports re-entrancy with adjuncts as in WHADVP relative clauses (*the place where we met*, etc.). However, in this case there will be no semantic form requesting the presence of the adjunct, so the semantic frame that affects the ranking probability is the frame that is already present in the local f-structure being resolved. The overall architecture of our system is presented in Figure 6.6.

6.4.4 Evaluation of F-Structures

We evaluate the proper f-structures produced by all grammars as described in Chapter 5 against the DCU 105 (Cahill et al., 2002a) and against the automatically generated f-structures for section 23 of the Penn-II treebank. We also evaluate against the PARC

	02–21	23	23 /(02–21)
TOPIC	26	7	2
FOCUS	13	4	0
TOPICREL	60	22	1

Table 6.14: Number of path types extracted

Recursively traverse f to:

find TOPIC|TOPICREL|FOCUS:g pair; retrieve TOPIC|TOPICREL|FOCUS paths; for each path p with $\mathrm{GF}_1 : \ldots : \mathrm{GF}_n : \mathrm{GF}$, traverse f along $\mathrm{GF}_1 : \ldots : \mathrm{GF}_n$ to sub-f-structure h; retrieve local PRED:l;

add GF:g to h iff

* GF is not present at h
* h together with GF is locally complete and coherent with respect to a semantic form s for l

rank resolution by $\mathcal{P}(s|l) \times \mathcal{P}(p|t)$

Figure 6.5: The core of the LDD resolution algorithm

700 Dependency Bank (King et al., 2003) following the experimental setup in Kaplan et al. (2004). There are systematic differences between the PARC 700 f-structures and the f-structures generated in our approach as regards feature-geometry, feature-nomenclature and the treatment of named-entities. In order to evaluate against the PARC 700 we need to map the f-structures produced by our parsers to a format similar to that of the PARC 700 Dependency Bank. This is done as a post-processing stage on the annotated trees produced in both parsing architectures (see Figure 6.7). For the PARC 700 evaluation, in a pre-processing stage before parsing, all named entities are determined and marked as proper nouns (NNP) dominating a single string for parsing (see Figure 6.8). Fractions are also marked as single strings with a CD (cardinal number) POS tag. For full details on

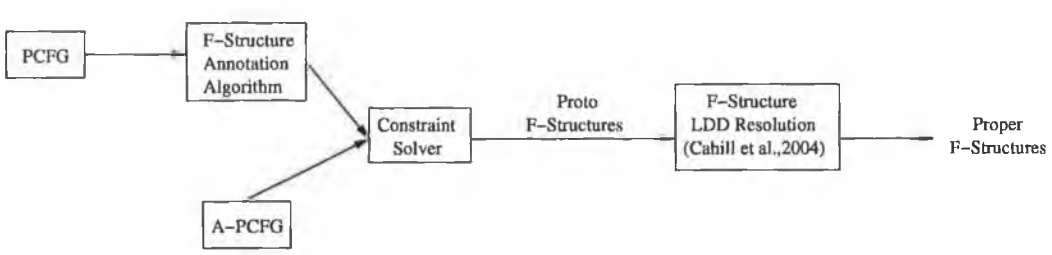


Figure 6.6: The overall architecture of our system

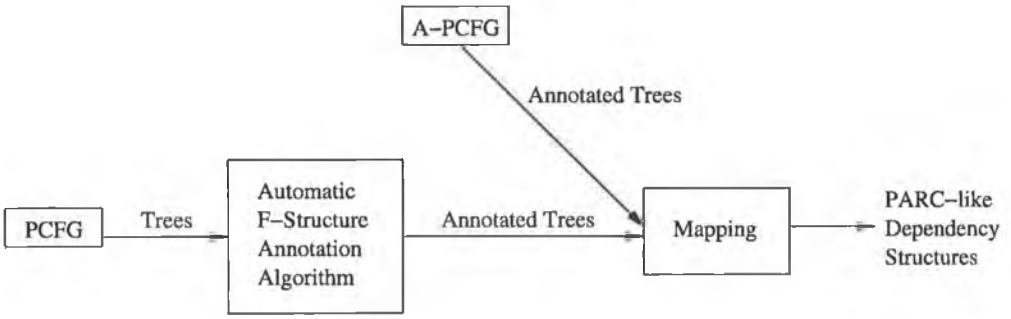


Figure 6.7: Mapping the output of our parsers to a format similar to the PARC 700 Dependency Bank

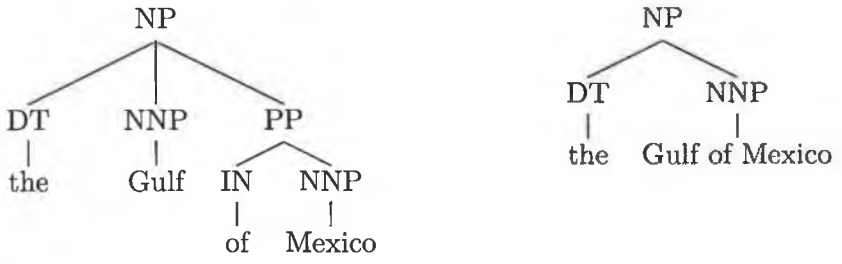


Figure 6.8: Pre-processing of named-entities for PARC 700 evaluation

the mapping, see Burke et al. (2004a).

Against DCU 105

Table 6.15 gives the results of the top 5 grammars in both the integrated and pipeline models against the DCU 105. These are not the same grammars that achieve the highest f-score for proto f-structures against the DCU 105 (Table 6.1). We also evaluate the proper f-structures produced by the grammars in Table 6.1. The results are given in

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
bfhkor	79.88	84.67	aegjop	81.24	87.04
bfhjnr	79.84	84.76	begjoq	81.13	85.99
bfhjop	79.78	87.20	begjop	81.10	87.09
afhjnr	79.65	84.68	aegjoq	80.94	85.70
afhjnp	79.55	86.56	begjor	80.83	84.90

Table 6.15: F-score results for the proper f-structures produced by the top 5 grammars in both integrated and pipeline models against the DCU 105

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
bfhjnr	79.84	84.76	aegjop	81.24	87.04
afhjnr	79.65	84.68	begjop	81.10	87.09
afhjnp	79.55	86.56	begjoq	81.13	85.99
aehjnr	79.40	84.35	aegjoq	80.94	85.70
bdhjnr	79.40	84.35	bfgmoq	80.70	85.55

Table 6.16: F-score results for the proper f-structures produced by the grammars in Table 6.1 against the DCU 105

Table 6.16. Again the integrated model performs better than the pipeline model against the DCU 105. Resolving LDDs leads to improved quality f-structures. Overall results in the pipeline model improve by between 4.94% and 5.08% for all GFs and between 5.55% and 5.61% for preds-only. In the integrated model, overall preds-only results improve by between 6.35% and 6.46% and for all GFs, f-score improves by between 5.79% and 5.88%. There is more improvement in f-score in the integrated model, and in each model, there is more improvement in preds-only evaluation than in all grammatical functions. The best performing grammar **aegjop** achieves 81.24% preds-only and 87.04% all GFs f-score. Table 6.17 gives a breakdown by feature of the results of parsing with grammar **aegjop** against the DCU 105. The results for each individual feature stay the same or improve after LDD resolution.

Against Section 23

Table 6.18 gives the results for the grammars that achieve the highest preds-only f-score against the 2416 f-structures *automatically* generated by the f-structure annotation algorithm for the *original* Penn-II trees, in a CCG-style (Hockenmaier, 2003) evaluation experiment. We also evaluate the proper f-structures produced for section 23 by the same grammars given in Table 6.1 against the automatically generated f-structures for section 23. The results are given in Table 6.19. Resolving LDDs improves results against the automatically annotated section 23 by between 3.23% and 3.58% in the pipeline model and between 3.19% and 3.30% in the integrated model. Table 6.20 gives a breakdown by feature of grammar **aegjop**. The results for all features apart from `app` and `obj2` increase. After LDD resolution, however, there is not the same increase in f-score between the DCU

DEP.	PRECISION (%)	RECALL (%)	F-SCORE (%)
focus	1/1 = 100	1/1 = 100	100
topic	12/12 = 100	12/13 = 92	96
det	253/264 = 96	253/269 = 94	95
xcomp	139/160 = 87	139/146 = 95	91
coord_form	71/72 = 99	71/85 = 84	90
poss	69/73 = 95	69/81 = 85	90
obj	387/445 = 87	387/461 = 84	85
subj	330/361 = 91	330/414 = 80	85
app	14/15 = 93	14/19 = 74	82
adjunct	717/903 = 79	717/947 = 76	78
quant	40/55 = 73	40/52 = 77	75
topicrel	35/42 = 83	35/52 = 67	74
coord	109/143 = 76	109/161 = 68	72
comp	35/43 = 81	35/65 = 54	65
relmod	26/38 = 68	26/50 = 52	59
obl	27/56 = 48	27/61 = 44	46
obl_ag	5/11 = 45	5/12 = 42	43
obl2	1/3 = 33	1/2 = 50	40
obj2	0/1 = 0	0/2 = 0	0

Table 6.17: Results for the preds-only evaluation of grammar **aegjop** against the DCU 105 broken down by function

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
aehknr	79.38	85.35	aegjor	78.73	84.36
aehknq	79.37	86.12	aegjoq	78.73	85.10
afhjnr	79.36	85.24	begjoq	78.69	85.06
adhknq	79.32	86.10	begjor	78.66	84.25
adhknr	79.28	85.28	aegjop	78.62	85.93

Table 6.18: F-score results for the proper f-structures produced by the top 5 grammars in both integrated and pipeline models against the f-structures automatically generated for section 23

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
bfhjnr	79.25	85.16	aegjop	78.62	85.93
afhjnr	79.36	85.24	begjop	78.59	85.88
afhjnp	78.94	86.57	begjoq	78.69	85.06
aehjnr	79.13	85.14	aegjoq	78.73	85.10
bdhjnr	79.10	85.13	bfgmoq	78.47	84.84

Table 6.19: F-score results for the proper f-structures produced by the grammars of Table 6.1 against the f-structures automatically generated for annotated section 23

105 and the 2416 automatically generated f-structures in section 23 as there had been before LDD resolution. The f-scores between the DCU 105 and the 2416 automatically generated f-structures decreases in the integrated model and preds-only in the pipeline model. The only increase in f-score between the DCU 105 and the 2416 automatically generated f-structures is in all grammatical functions in the pipeline model. This suggests the LDD resolution algorithm is not performing as well on the automatically generated f-structures for Section 23 as it is on the DCU 105. The LDD resolution algorithm only makes a distinction between two different types of TOPICREL functions (with and without *wh* elements). However, a number of more fine-grained distinctions are possible, such as prepositional-type TOPICREL clauses such as *in which*. It is also possible to distinguish adverbial focus from non-adverbial focus, a distinction we have not made. Such experiments remain for further work, but we would expect them to result in improved results when evaluating against the automatically generated f-structures for Section 23.

DEP.	PRECISION (%)	RECALL (%)	F-SCORE (%)
coord_form	1288/1348 = 96	1288/1461 = 88	92
det	5594/6014 = 93	5594/6327 = 88	91
quant	1075/1172 = 92	1075/1252 = 86	89
poss	1240/1350 = 92	1240/1455 = 85	88
xcomp	2882/3333 = 86	2882/3352 = 86	86
obj	7739/9173 = 84	7739/9538 = 81	83
subj	6736/7915 = 85	6736/9151 = 74	79
topic	203/254 = 80	203/269 = 75	78
adjunct	14449/18714 = 77	14449/19919 = 73	75
coord	2094/2754 = 76	2094/3014 = 69	73
topicrel	771/1064 = 72	771/1074 = 72	72
comp	556/911 = 61	556/810 = 69	65
app	263/383 = 69	263/433 = 61	64
relmod	575/929 = 62	575/1084 = 53	57
focus	5/7 = 71	5/11 = 45	56
obl	615/1289 = 48	615/1140 = 54	51
obl_ag	83/217 = 38	83/181 = 46	42
obj2	12/52 = 23	12/44 = 27	25
obl2	14/70 = 20	14/43 = 33	25

Table 6.20: Results for the preds-only evaluation of grammar **aegjop** against the f-structures automatically generated for section 23 broken down by function

Pipeline Model		Integrated Model	
Grammar	F-Score (%)	Grammar	F-Score (%)
aehknp	80.33	begjop	78.74
bdhjnp	80.29	aegjop	78.60
behjnp	80.28	aegjnp	78.43
adhknp	80.26	afgmop	78.38
adhjnp	80.24	bfgmop	78.31

Table 6.21: F-score results for the proper f-structures produced by the top 5 grammars in both integrated and pipeline models against the PARC 700

DEP.	PRECISION (%)	RECALL (%)	F-SCORE (%)
number_type	419/435 = 96	419/440 = 95	96
det_form	915/956 = 96	915/964 = 95	95
pron_form	505/548 = 92	505/531 = 95	94
tense	974/1054 = 92	974/1051 = 93	93
num	3709/4004 = 93	3709/4145 = 89	91
coord_form	229/257 = 89	229/252 = 91	90
prog	169/174 = 97	169/203 = 83	90
stmt_type	923/1050 = 88	923/1044 = 88	88
perf	73/82 = 89	73/86 = 85	87
poss	171/199 = 86	171/205 = 83	85
obj	1459/1793 = 81	1459/1866 = 78	80
quant	288/336 = 86	288/381 = 76	80
adegree	997/1224 = 81	997/1290 = 77	79
prt_form	34/41 = 83	34/46 = 74	78
obl_ag	33/41 = 80	33/45 = 73	77
subord_form	52/58 = 90	52/77 = 68	77
pcase	34/41 = 83	34/52 = 65	73
subj	1156/1371 = 84	1156/1779 = 65	73
comp	172/236 = 73	172/257 = 67	70
conj	377/531 = 71	377/552 = 68	70
passlve	140/176 = 80	140/238 = 59	68
xcomp	305/431 = 71	305/478 = 64	67
adjunct	2308/3535 = 65	2308/3568 = 65	65
topicrel	93/184 = 51	93/119 = 78	61
obl	94/271 = 35	94/188 = 50	41
precoord_form	0/0 = 0	0/6 = 0	0
focus	0/0 = 0	0/5 = 0	0
obj_theta	0/6 = 0	0/11 = 0	0

Table 6.22: F-score results for the evaluation of grammar **aehknp** against the PARC 700 broken down by function

Pipeline Model		Integrated Model	
Grammar	F-Score (%)	Grammar	F-Score (%)
bfhjnr	76.36	aegjop	78.60
afhjnr	76.32	begjop	78.74
afhjnp	79.78	begjoq	75.38
aehjnr	76.79	aegjoq	75.17
bdhjnr	76.89	bfgmoq	74.95

Table 6.23: Evaluation of the top 10 grammars of Table 6.1 against the PARC 700

Pipeline Model			Integrated Model		
Grammar	Preds Only (%)	All GFs (%)	Grammar	Preds Only (%)	All GFs (%)
aehknp	78.41	86.22	begjop	81.10	87.09
bdhjnp	79.10	86.30	aegjop	81.24	87.04
behjnp	79.06	86.28	aegjnp	78.95	84.96
adhknp	78.33	86.18	afgmop	80.14	86.22
adhjnp	79.10	86.30	bfgmop	80.24	86.41

Table 6.24: F-score results for the proper f-structures produced by the grammars that perform best against the PARC 700 evaluated against the DCU 105

Against the PARC 700

We evaluate our grammars against the PARC 700 Dependency Bank following the experimental setup in Kaplan et al. (2004) with a set of features that is a proper superset of preds-only, but a proper subset of all grammatical functions (preds-only $\subset$ PARC $\subset$ all GFs). Complete results for all grammars are given in Appendix D. Table 6.21 gives the results of the top five grammars in the pipeline model and the top five in the integrated model according to the evaluation against the PARC 700. Our best grammar currently achieves an f-score of 80.33%. Table 6.22 gives the results broken down by feature of this grammar. The pipeline model performs better than the integrated model, with a difference of 1.59% between the best pipeline model grammar and the best integrated model grammar. Table 6.23 gives the results of the proper f-structures produced by our top 10 grammars of Table 6.1 against the PARC 700. The top grammars according to the DCU 105 do not perform best against the PARC 700. Table 6.24 gives the results of evaluating the grammars that perform best against the PARC 700 against the DCU 105. These results indicate that while the grammars that perform best against the PARC 700 may not

always give the top results against the DCU 105, they still achieve very high results.

Similar patterns emerge as to which transformations and pre-processing steps are most useful for each parsing architecture and gold-standard evaluation experiment. The pipeline architecture performs best with Penn-II functional labels and either a parent or grandparent transformation. The integrated architecture usually performs better with a parent transformation and no Penn-II functional labels. However, it also performs well with no parent transformation, indicating that the f-structure information on category labels alone produces high quality f-structures. The difference in performance between the integrated model and the pipeline model against the PARC 700 is greater than against the DCU 105 or the automatically generated f-structures for section 23. It is unclear why this is the case, and the observation requires further investigation.

6.4.5 Evaluation of LDD Resolution

We measured the effect of LDD resolution in terms of overall improvement in f-score between proto f-structures and proper f-structures against the DCU 105 and against the automatically generated f-structures for Section 23. However, we would also like to know to what extent the LDD resolution considered in isolation is correct. In order to measure how many of the LDD re-entrancies in the gold-standard f-structures are captured correctly by our parsers, we developed evaluation software for f-structure LDD re-entrancies. To determine whether or not an LDD re-entrancy in a proper f-structure determined in one of our parsing architectures is correct, we must determine whether the path between the TOPIC, FOCUS or TOPICREL and its re-entrant element matches the corresponding path in the gold standard. It is straightforward to extract the re-entrancy paths from the dependency relations in triple format using the indices on the arguments. Figure 6.9 shows the paths extracted for two sets of triples. Table 6.25 shows the results of LDD re-entrancy evaluation with a grammar in the integrated model achieving 81.6% correct LDD re-entrancies.[1]

Evaluating against the DCU 105, the accuracy of LDD resolution is higher in the integrated model than in the pipeline model, probably for the reason that here the integrated

[1]There is only one example of FOCUS in our gold standard which is re-entrant with an adjunct. Although our algorithm allows re-entrancy with adjuncts, this path has a low probability, so is therefore not chosen.

```
topic(say~0,have~2)   topic(say~0,have~2)
comp(say~0,have~2)    comp(say~0,be~1)
                      xcomp(be~1,have~2)

Path: ↑topic=↑comp    Path: ↑topic=↑comp:xcomp
```

Figure 6.9: Extracting re-entrancy paths from dependency relation triples

Pipeline Model					Integrated Model				
Grammar	TOPIC	FOCUS	TOPICREL	OVERALL	Grammar	TOPIC	FOCUS	TOPICREL	OVERALL
bfhjnr	88.00%	0.00%	67.33%	70.87%	aegjop	96.00%	0.00%	72.34%	76.03%
afhjnr	88.00%	0.00%	65.98%	69.92%	begjop	96.00%	0.00%	71.58%	75.41%
afhjnp	88.00%	0.00%	68.00%	71.43%	begjoq	96.00%	0.00%	72.92%	76.42%
aehjnr	92.31%	0.00%	64.44%	69.49%	aegjoq	96.00%	0.00%	73.68%	77.05%
bdhjnr	92.31%	0.00%	64.44%	69.49%	bfgmoq	100.00%	0.00%	78.35%	81.60%

Table 6.25: LDD evaluation on the DCU 105 in both integrated and pipeline models

model has an overall higher f-score than the pipeline model. Table 6.26 gives the dependency relation score for the three functions that trigger LDD resolution (TOPIC, FOCUS and TOPICREL) in each grammar. The score for both TOPIC and TOPICREL is better in the integrated model than in the pipeline model. Also, the grammar that has no parent transformation has the best score for TOPIC and TOPICREL and also the best re-entrancy accuracy. Our algorithm attempts to find re-entrancies for all TOPIC, FOCUS and TOPICREL elements, so obviously, the more unresolved correct instances of TOPIC, FOCUS and TOPICREL functions the parser finds prior to LDD resolution, the higher the score for the re-entrancies is likely to be. The score for re-entrancies cannot be higher than the dependency relation score, and Tables 6.25 and 6.26 show that the scores are in fact quite closely related. Table 6.27 shows that, for example, the grammar **bfgmoq** correctly identifies the re-entrancy for all but one of the TOPICRELs it correctly identifies. Grammars that have higher scores for the functions that trigger LDD resolution will have higher overall scores for re-entrancy. The problem then is to try and generate grammars with improved dependency results for these functions. Table 6.28 shows the grammars that achieve the highest overall LDD re-entrancy results. Interestingly, the grammars that perform best in the integrated models do not have any parent transformation, as this information seems to lead to poorer quality TOPIC, FOCUS and TOPICREL determination. Also, the pipeline

Pipeline Model				Integrated Model			
Grammar	TOPIC	FOCUS	TOPICREL	Grammar	TOPIC	FOCUS	TOPICREL
bfhjnr	88%	0%	71%	aegjop	96%	0%	74%
afhjnr	88%	0%	68%	begjop	96%	0%	74%
afhjnp	88%	0%	70%	begjoq	96%	0%	75%
aehjnr	92%	100%	67%	aegjoq	96%	0%	76%
bdhjnr	92%	100%	67%	bfgmoq	100%	100%	80%

Table 6.26: Dependency relation results for TOPIC, FOCUS and TOPICREL against the DCU 105 in both integrated and pipeline models

	TOPIC	FOCUS	TOPICREL
Dependency Relation	13/13	0/0	39/45
Re-entrancy	13/13	0/0	38/45

Table 6.27: The dependency relation precision results and the re-entrancy precision scores for **bfgmoq**

model grammars in the table do not have any Penn-II functional labels – information that generally tends to produce higher quality trees. It is currently difficult to say why exactly these grammars produce higher scores for the TOPIC, FOCUS and TOPICREL functions, yet lower quality f-structures overall than most of our best grammars.

Pipeline Model					Integrated Model				
Grammar	TOPIC	FOCUS	TOPICREL	OVERALL	Grammar	TOPIC	FOCUS	TOPICREL	OVERALL
bfhjop	92.31%	0.00%	80.39%	82.17%	bfgmor	100.00%	0.00%	80.00%	82.81%
afhkor	84.62%	0.00%	78.85%	79.39%	afgmor	100.00%	0.00%	78.79%	82.54%
bfhjoq	88.00%	0.00%	77.55%	79.03%	bfgmoq	100.00%	0.00%	78.35%	81.60%
afhjop	84.62%	0.00%	76.92%	77.86%	afgmoq	100.00%	0.00%	77.08%	81.30%
afhkop	84.62%	0.00%	76.92%	77.86%	bfgmop	100.00%	0.00%	77.55%	80.95%

Table 6.28: The grammars that have the best LDD evaluation results against the DCU 105 in both integrated and pipeline models

6.5 Summary

I presented two PCFG-based parsing architectures for parsing raw text into f-structures, the pipeline model and the integrated model.

I evaluate the proto f-structures (basic, but possibly incomplete, predicate-argument-adjunct structures where LDDs are unresolved) produced by grammars in both models. Against the DCU 105, our best grammar in the pipeline model achieves an f-score of

74.25% preds-only and 79.81% on all grammatical functions. Our best grammar in the integrated architecture achieves an f-score of 74.80% preds-only and 81.20% on all GFs. In a CCG-style experiment against the 2,416 automatically generated f-structures for the original treebank trees in section 23 (Hockenmaier, 2003), the pipeline model achieves higher results than the integrated model (75.67% preds-only and 81.74% all GFs). I evaluate what percentage of f-structures receive a covering and connected f-structure. Generally, for our best performing grammars, only a small number of sentences do not receive any f-structure and almost all sentences receive just one covering and connected f-structure.

I present and evaluate a finite approximation of LDD resolution in automatically constructed, wide-coverage, robust, PCFG-based LFG approximations, effectively turning the "half"(or "shallow")-grammars presented in Section 6.3 and Cahill et al. (2002c) into "full" or "deep" grammars. In our approach, as in mainstream LFG (Dalrymple, 2001), LDDs are resolved in f-structure, not trees. The resolution algorithm requires functional uncertainty (FU) paths and subcategorisation frames. We automatically extract a finite approximation of FU paths and generate subcategorisation frames from the f-structure-annotated Penn-II treebank resource. Currently, the quality of the preds-only f-structures produced by our best grammars after LDD resolution improved results over the same grammars without LDD resolution by between 5.55% and 6.46% for the DCU 105, with the best grammar currently achieving 81.24% preds-only f-score and 87.04% f-score for all GFs. Against the 2,416 automatically generated f-structures for the original Penn-II treebank trees, preds-only f-structures after LDD resolution improved by between 3.28% and 3.58% with the best grammar currently achieving 79.38% preds-only f-score and 85.35% for all GFs. Evaluating against the PARC 700 Dependency Bank following the evaluation described in Kaplan et al. (2004), our best grammar achieves an f-score of 80.33%. Although the improvement in f-structure f-score gives us an indication that our LDD resolution algorithm is performing well, we also need to evaluate the LDD resolution independently. To do this, we measure the quality of the re-entrancies in the triples. Our best grammar achieves an overall score of 82.81% on LDD resolution paths against DCU 105. We show that the quality of the re-entrancies is dependent on the quality of the LDD resolution

triggering functions in the f-structures.

Chapter 7

Comparison of Our Approach with Other Approaches

7.1 Introduction

The work reported in this dissertation provides treebank-based wide-coverage, robust, and – with the addition of LDD resolution – "deep" or "full", PCFG-based LFG *approximations*. Crucially, we do not claim to provide fully adequate statistical models. It is well known (Abney, 1997) that PCFG-type approximations to general constraint-based grammars can yield inconsistent probability models due to loss of probability mass: the parser successfully returns the highest ranked parse tree but the constraint solver cannot resolve the f-structure equations (generated in the pipeline or "hidden" in the integrated model) and the probability mass associated with that tree is lost. Research on adequate probability models for unification grammars is important. Miyao et al. (2003) present a Penn-II treebank-based HPSG with log-linear probability models. They achieve coverage of 50.2% on section 23, as against 98% in our approach. Hockenmaier (2003) provides CCG-based models including LDD resolution. Some of these involve extensive clean-up of the underlying Penn-II treebank resource prior to grammar extraction. In contrast, in our approach we leave the treebank as is and only add (but never correct) annotations. Earlier HPSG work (Tateisi et al., 1998) is based on independently constructed *hand-crafted* XTAG resources. In contrast, we acquire our resources from treebanks and achieve

substantially wider coverage. Collins's (1999) Model 3 is limited to wh-traces in relative clauses (it does not treat topicalisation, focus etc.). Johnson's (2002) work is closest to ours in spirit, adding empty nodes and their antecedents to parser output trees in order to capture LDDs. Riezler et al. (2002) and Kaplan et al. (2004) describe how a carefully hand-crafted LFG is scaled to the full Penn-II treebank with log-linear-based probability models. In this chapter I will compare our work to that of Collins (1999), Johnson (2002), Riezler et al. (2002) and Kaplan et al. (2004) and report on a number of experiments.

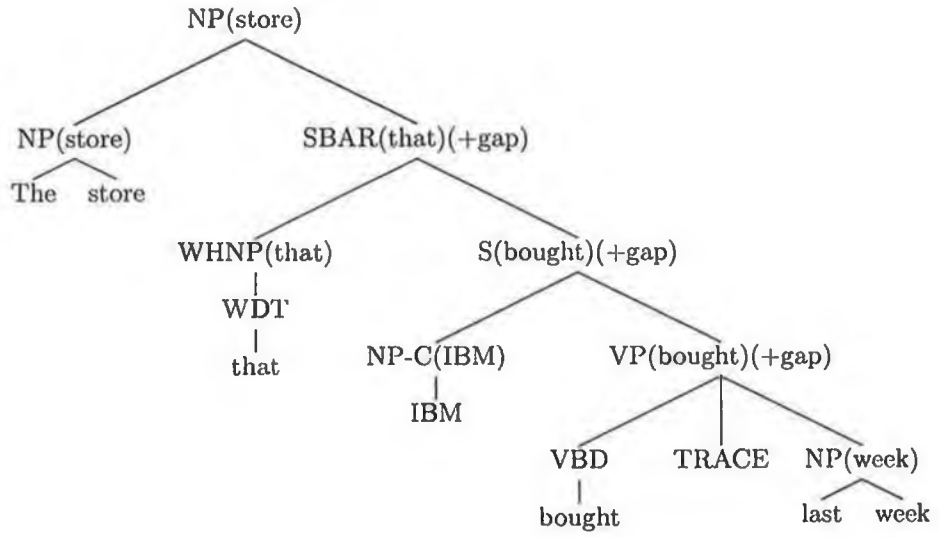
7.2 Other Deep Parsing Approaches

7.2.1 Collins' Model 3

The only type of long-distance dependency dealt with by Collins' Model 3 parser (Collins, 1999) is wh-movement in relative clauses. This is done by identifying a co-indexed gapped element in the parse tree with the WHNP head of the relative clause (SBAR). If the LHS of a rule features a gap annotation, Collins identifies three ways in which this gap may be passed to the RHS. First, the gap can be passed to the head of the phrase. Alternatively, it can be passed on to one of the left or right modifiers of the head. It can also be discharged as a TRACE argument to the left or right of the head. The example in Figure 7.1 (taken from Collins (1999)) shows how a gap is introduced by the WHNP and then passed down through the tree until it can be discharged as the complement of *bought*.

Overall parsing results are improved slightly when wh-movement is incorporated into Collins' parsing model. Model 3 achieves an f-score of 88.65% on sentences of length $\leq$ 40, where Model 2 that does not have any wh-movement achieves an f-score of 88.60%.

It is difficult to provide a satisfactory comparison of Collins' Model 3 and our LDD resolution method, but we have carried out two experiments that compare them at the f-structure level. We use the output of Collins' Model 3 parser and generate f-structures in two versions of our pipeline architecture. As there is no explicit link in Collins' trees between the TRACE node and its antecedent, we add this automatically (assuming that all TRACE nodes are co-indexed with the nearest WHNP node in the tree) as in Figure 7.2. We also re-label the TRACE node to -NONE- so that it resembles the trace informa-



1. NP → NP SBAR(+gap)
2. SBAR(+gap) → WHNP S(+gap)
3. S(+gap) → NP-C VP(+gap)
4. VP(+gap) → VBD TRACE NP

Figure 7.1: A +gap feature is added to non-terminals to describe wh-movement. The gap is passed through the tree until it is discharged as a TRACE complement to the right of *bought*

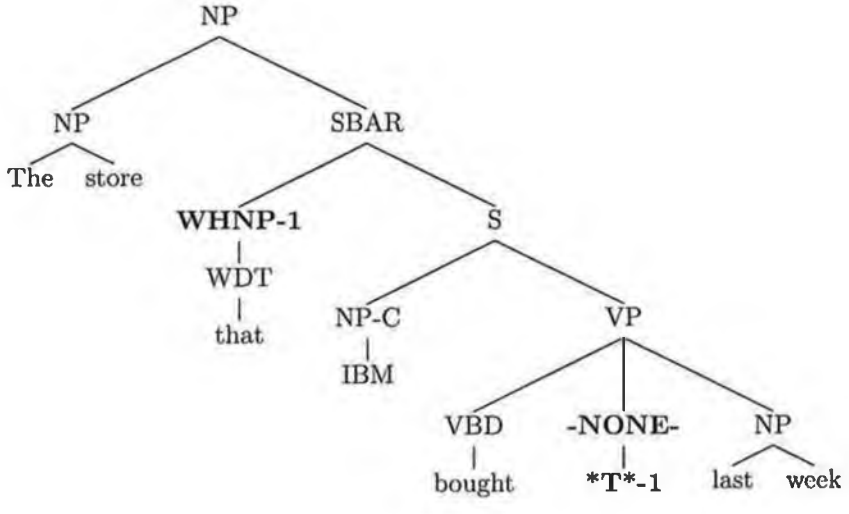


Figure 7.2: Co-indexing and re-labelling the TRACE node of Collins' Model 3 output with the nearest WHNP node (cf. Figure 7.1).

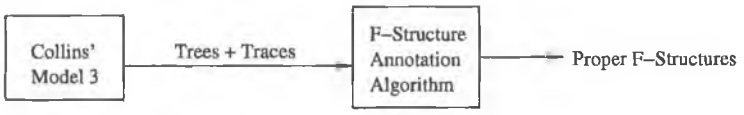


Figure 7.3: Producing proper f-structures from the trees with traces from Collins' Model 3 parser

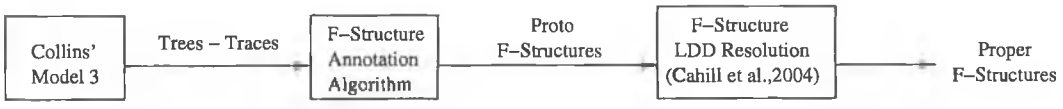


Figure 7.4: Producing proper f-structures from the trees without traces from Collins' Model 3 parser in our pipeline architecture

tion in the Penn-II treebank, to which our automatic f-structure annotation algorithm is sensitive. We carry out two experiments. First, we take the Collins' Model 3 trees with traces and the automatically added co-indexation to the nearest WHNP and apply the automatic f-structure annotation algorithm (cf. Chapter 2) which translates traces and co-indexation in trees into corresponding re-entrancies in f-structure to produce proper f-structures (Figure 7.3). Second, we take the Collins' Model 3 trees without traces or co-indexation and apply our f-structure annotation algorithm followed by LDD resolution (cf. Chapter 6) to produce proper f-structures (this is our standard pipeline processing architecture (Figure 7.4)). The two experiments are designed to discriminate between the effects of Collins' traces in trees and our f-structure-based LDD resolution. We evaluate the f-structures against:

1. The DCU 105 (Cahill et al., 2002a),
2. The full 2416 f-structures *automatically* generated by the f-structure annotation algorithm for the *original* Penn-II trees, in a CCG-style (Hockenmaier, 2003) evaluation experiment,
3. A subset of 560 dependency structures of the PARC 700 Dependency Bank following Kaplan et al. (2004).

The results are presented in Table 7.1. Using the trees with traces and automatically added co-indexation to the nearest WHNP to produce proper f-structures with the f-structure annotation algorithm in the architecture presented in Figure 7.3, we achieve

	DCU 105		Section 23		Parc 700
	Preds-Only	All GFs	Preds-Only	All GFs	
Collins Model 3 - traces (pipeline + LDD resolution)	77.63%	85.30%	79.85%	86.29%	80.21%
Collins Model 3 + traces (f-structure annotation algorithm)	72.91%	80.68%	76.81%	83.49%	80.19%

Table 7.1: Evaluating the effect of Collins' traces in trees and our f-structure-based LDD resolution

a preds-only f-score of 72.91% against DCU 105, 76.81% against the 2416 f-structures *automatically* generated by the f-structure annotation algorithm and 80.19% against the PARC 700. By contrast, using the trees produced by Collins' Model 3 without traces or co-indexation in our standard pipeline architecture with LDD resolution (Figure 7.4, we achieve a preds-only f-score of 77.63% against DCU 105, 79.85% against the 2416 f-structures *automatically* generated by the f-structure annotation algorithm and 80.21% against the PARC 700 following Kaplan et al. (2004). The results show that our LDD resolution on f-structure outperforms Collins' (1999) Model 3.

7.2.2 Johnson 2002

Johnson's (2002) work is closest to ours in spirit. Like our approach he provides a finite approximation of LDDs. Unlike our approach, however, he works with tree fragments in a post-processing approach to add empty nodes and their antecedents to Charniak's (2000) parse trees, while we present an approach to LDD resolution at the level of f-structure. It seems that the f-structure-based approach is more abstract (99 LDD path types against approximately 9,000 tree-fragment types in Johnson (2002)) and fine-grained in its use of lexical information (subcategorisation frames). In contrast to Johnson's approach, our LDD resolution algorithm is not biased. Johnson's approach is biased in favour of deeper patterns because it uses a pre-order traversal to insert empty nodes into the tree. Our algorithm, on the other hand, computes all possible complete resolutions and order-ranks them using LDD path and subcategorisation frame probabilities. It is difficult to provide a satisfactory comparison between the two methods, but we have carried out two experiments that compare them at the f-structure level.

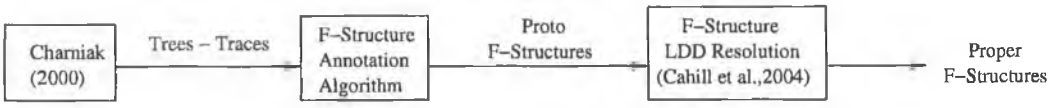


Figure 7.5: Producing proper f-structures from Charniak's (2000) trees in the pipeline architecture

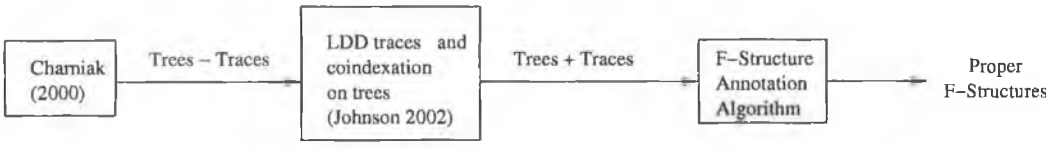


Figure 7.6: Producing proper f-structures from Charniak's (2000) trees with empty nodes and coindexation added by Johnson (2002)

In the first experiment, we take the output of Charniak's parser (Charniak, 2000) and, using the pipeline parsing model (Chapter 6), evaluate (both before and after LDD resolution) against the DCU 105 (Cahill et al., 2002a), the full 2416 f-structures *automatically* generated by the f-structure annotation algorithm for the *original* Penn-II trees in a CCG-style (Hockenmaier, 2003) evaluation experiment, and against a subset of 560 dependency structures of the PARC 700 Dependency Bank following Kaplan et al. (2004)

In the second experiment, using the software described in Johnson (2002), we add empty nodes and coindexed antecedents to the trees generated by Charniak's parser, pass these trees to our automatic annotation algorithm (Chapter 2) which generates proper f-structures and evaluate against the DCU 105, the full 2416 f-structures *automatically* generated by the f-structure annotation algorithm for the *original* Penn-II trees, and the PARC 700 Dependency Bank following Kaplan et al. (2004). Figures 7.5 and 7.6 outline the architectures of both experiments.

In the first experiment, LDD resolution is performed purely at the level of f-structures (as described in Chapter 6), in the second, LDD resolution is triggered purely by co-indexation information on trees generated by Johnson's (2002) software translated into corresponding re-entrancies in f-structure by the f-structure annotation algorithm of Chapter 2. The results are given in Table 7.2. Our method of resolving LDDs at f-structure level results in a preds-only f-score of 80.65% against the DCU 105. Using Johnson's (2002) method of adding empty nodes to the parse-trees results in an f-score of 79.52%

Charniak	-LDD res.	+LDD res.	Johnson (2002)
DCU 105			
All Grammatical Functions	80.86%	86.65%	85.17%
Preds Only	74.43%	80.65%	79.52%
2416 Section 23% Sentences			
All Grammatical Functions	84.89%	88.38%	87.56%
Preds Only	79.34%	83.08%	82.33%
PARC 700			
Subset of GFs following Kaplan et al. (2004)	81.4%	81.79%	81.75

Table 7.2: Comparison at f-structure level of LDD resolution to Johnson (2002) on the DCU 105

against the DCU 105. Against the 2416 f-structures automatically generated, and against the PARC 700, our method of resolving LDDs at f-structure level performs slightly better than using Charniak's trees with empty nodes, traces and coindexation information added by Johnson (2002).

7.2.3 The English ParGram Grammar

The only other wide-coverage LFG grammar for English that we are aware of is the hand-crafted LFG developed at the Palo Alto Research Center in California as part of the ParGram project (Butt et al., 1999, 2002). The aim of the ParGram project is to produce wide-coverage grammars for English, French, German, Norwegian, Japanese, and Urdu which are written collaboratively within the linguistic framework of LFG and with a commonly agreed set of grammatical features.

In Riezler et al. (2002) and Kaplan et al. (2004), the hand-crafted English grammar is scaled to the full Penn-II treebank, using log-linear-based probability models to disambiguate parses. Currently, they achieve 79% coverage (full parse) and 21% fragment/skimmed parses. By the same measure, full parse coverage is around 98% for our automatically acquired PCFG-based LFG approximations. Kaplan et al. (2004) report results for two of their hand-crafted grammars against the PARC 700, LFG core and LFG complete. LFG core is a version of LFG complete that moves most of the optimality marks into the NOGOOD space.[1] LFG core achieves an f-score of 77.6%, while LFG complete achieves an f-score of 79.6% against the PARC 700. For the same experiment, our

[1]For example, in LFG complete, an OT mark disprefers topicalisation but the topicalisation rule will be triggered if no other parse can be built. In LFG core, if the topicalisation OT mark is in the NOGOOD space, the topicalisation rule will never be triggered.

own best automatically-induced grammar (presented in Chapter 6) achieves an f-score of 80.33%. Using Collins' Model 3 in our pipeline architecture (as describe in Section 7.2.1), we achieve 80.21%. Using Charniak's (2000) parser in our pipeline model, we achieve 81.79%, a 2.19% improvement over the best hand-crafted grammar presented in Kaplan et al. (2004).

It is instructive to compare the results established here to another experiment in Kaplan et al. (2004). Kaplan et al. show how the output trees generated by Collins' Model 3 can be processed deterministically into corresponding dependency relations, effectively implementing what was presented as a pipeline architecture in Cahill et al. (2002c). For this deterministic dependency conversion of the Collins trees, Kaplan et al. (2004) report an f-score of 74.6% against the PARC 700. Using our automatic f-structure annotation algorithm, we achieve 80.21% f-score for the Collins trees, an improvement of 5.61%. This and the other results reported here show that our automatic f-structure annotation algorithm can be used to induce high-quality LFG resources and processing architectures, currently outperforming even the best manually created resources.

7.3 Summary

We have developed large-scale, treebank-based, probabilistic LFG approximations with LDD resolution that generate predicate-argument structures or deep dependency relations. We compare our work with three approaches that have similar aims. The first is Collins' Model 3 parser (Collins, 1999) which deals only with wh-movement in relative clauses. To compare our work with this, we use the trees produced by Collins'(1999) parser in two versions of our pipeline processing architecture to produce f-structures from which we generate dependency relations. First we automatically add co-indexation to the nearest WHNP to the Collins' Model 3 trees with traces. Our automatic f-structure annotation algorithm is sensitive to the traces and coindexation. Against the DCU 105, the Collins Model 3 trees with traces and co-indexation achieve a preds-only f-score of 72.91%. Evaluating against the PARC 700, Collins' Model 3 trees with traces and co-indexation achieves an f-score of 80.19%. Second, we use Collins' Model 3 trees without traces or co-indexation and, in our standard pipeline parsing architecture, automatically annotate them with f-

structure information, resolve LDDs and produce proper f-structures. Against the DCU 105, Collins' Model 3 trees without traces or co-indexation achieve a preds-only f-score of 77.63%. Against the PARC 700, the trees without traces or co-indexation achieve an f-score of 80.21%. These results show that our LDD resolution on f-structure outperforms Collins' (1999) Model 3.

Johnson (2002) presents an approach that adds empty nodes and their antecedents in a post-processing stage after parsing in order to capture long-distance dependencies. Again, it is difficult to compare our work directly with his, as the two approaches work on different levels of representation. We carried out two experiments that compare them at the f-structure level. First, we feed the trees generated by Charniak's (2000) parser into our pipeline model with LDD resolution at f-structure level. Second, we take the trees generated by Charniak's parser, apply Johnson's (2002) post-processing to add empty productions and coindexed antecedents to the trees and feed these trees into the automatic f-structure annotation algorithm which translates coindexation in trees into corresponding re-entrancies in f-structure. Against the DCU 105, LDD resolution at the level of f-structure results in an f-score of 80.65% preds-only, while f-structures induced from Johnson's post-processed trees achieve a preds-only f-score of 79.52%. Against the 2416 f-structures automatically generated for the original trees in section 23, the f-structures induced from Johnson's post-processed trees achieve a preds-only f-score of 82.33%, while LDD resolution at f-structure level on Charniak's (2000) original trees results in a preds-only f-score of 83.08%. Against the PARC 700, LDD resolution at f-structure level results in an f-score of 81.79%, while f-structures induced from Johnson's post-processed trees achieve an f-score of 81.75%.

Finally, we compare our dependency structures to those produced by the best hand-crafted grammars of Riezler et al. (2002) and Kaplan et al. (2004). The LFG-complete grammar in Kaplan et al. (2004) achieves an f-score of 79.6% against the PARC 700, where our best grammar (Chapter 6) achieves an f-score of 80.33%. Using Collins' Model 3 in our pipeline architecture (as describe in Section 7.2.1), we achieve 80.21%. Using Charniak's (2000) parser in our pipeline model, we achieve 81.79%, a 2.19% improvement over the best hand-crafted grammar presented in Kaplan et al. (2004). These results show

that the quality of the f-structures and dependency relations produced by our methods, in both pipeline and integrated architectures, is comparable to or better than other existing similar work. Significantly, the results show that our methodology supports the automatic induction of wide-coverage and deep grammatical resources as well as the construction of flexible and modular processing architectures currently outperforming the best deep, hand-crafted, wide-coverage, constraint-based grammar resources. We believe that our methodology constitutes an important alternative to more traditional manual grammar development and processing architectures.

Chapter 8

Migrating Automatic Annotation-Based Grammar Acquisition and Parsing to German and the TIGER Treebank

8.1 Introduction

Developing deep unification grammars is a highly knowledge-intensive task and grammars are typically hand-crafted. Scaling such grammars beyond small fragments to unrestricted, naturally occurring, real text, is time-consuming and expensive, involving, as it does, person years of expert labour. I have outlined a method of inducing wide-coverage, probabilistic LFG grammatical resources for English from an automatically f-structure annotated Penn-II treebank. In this chapter, I will describe how this methodology can be migrated to a different language and treebank resource, namely German and the TIGER treebank (Brants et al., 2002). German is substantially less configurational than English, and the TIGER treebank data structures consist of graphs with crossing edges rather than trees with traces as in Penn-II. In addition, the TIGER treebank features considerably richer functional annotations than those provided in the Penn-II resource. I present an f-structure annotation algorithm for TIGER and outline how LFG grammars for German

can be derived from the f-structure-annotated TIGER resource. I extract PCFG-based LFG approximations and report on a number of parsing experiments. I evaluate both the quality of the automatic f-structure annotation of the TIGER treebank, and the parser output. I then describe a morphological case-simulating grammar transformation and evaluate the results. Part of the work reported here and early results have appeared in Cahill et al. (2003a)

8.2 From TIGER to a German LFG

The TIGER treebank (Brants et al., 2002) is a corpus of approximately 40,000 syntactically annotated German newspaper sentences. The annotation consists of generalised graphs, which may contain crossing and secondary edges. Crossing edges are used to represent long-distance dependencies and secondary edges represent information relating to certain re-entrancies such as shared subject in coordinate constructions. Edges are labelled, so that a TIGER tree encodes both phrase-structural information and dependency relations.

Forst (2003a,b) converts the TIGER graphs directly into f-structures in order to generate a set of reference f-structures to evaluate a hand-crafted wide-coverage German LFG. However, in order to be able to extract an annotated PCFG which can be used to parse text into f-structures, we require trees that have been annotated with f-structure equations, rather than the f-structures themselves.

Since the structure of the TIGER corpus is quite different to that of the Penn-II treebank, the approach taken to annotating the TIGER corpus with f-structure equations differs from the approach for English described earlier. German does not usually rely on configurational information to express functional information, a feature of English that was heavily exploited by us in our previous work. However, the TIGER corpus annotation scheme provides rich functional information by way of labelled edges in the graphs. By exploiting these labels we can annotate the TIGER corpus with f-structure equations.

8.2.1 From TIGER Graphs to Trees

The first stage in annotating the TIGER corpus with f-structure equations (in order to be able to extract PCFG-based LFG approximations) is to convert the TIGER graphs into

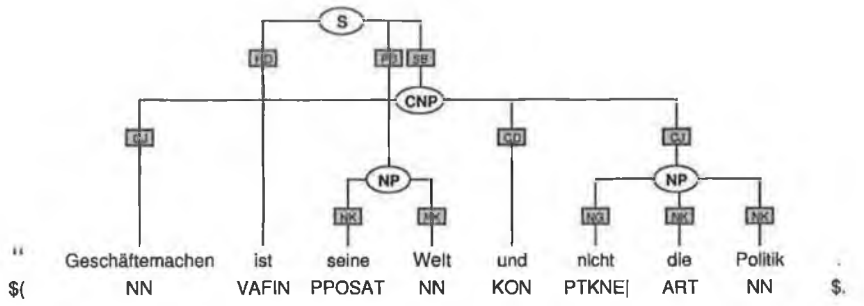
Category	Description
S	Sentence
CNP	Coordinated noun phrase
NP	Noun phrase
POS Tag	**Description**
NN	Common Noun
VAFIN	Finite verb, auxiliary
PPOSAT	Attributive possessive pronoun
KON	Coordinate conjunction
PTKNEG	Negative particle
ART	Definite or indefinite article
Function	**Description**
CJ	Conjunct
NK	Noun kernal element
NG	Negation
CD	Coordinating conjunction
HD	Head
PD	Predicate
SB	Subject

Table 8.1: A glossary for the category, POS tag and functional labels used in the TIGER graph of Figure 8.1

trees similar to those found in the Penn-II treebank.[1] Traces are used to represent the long-distance dependency information captured by crossing edges. Secondary edges have not been incorporated into the f-structure annotation procedure at this stage. Although these edges obviously contain useful information for the generation of f-structures, this information is currently not utilised, since it is unclear how one could encode them in a useful manner in the Penn-II style trees.

Figures 8.1, 8.3 and 8.4 illustrate how traces in tree data structures are used to represent the information encoded by crossing edges in the TIGER graphs. Table 8.1 provides a glossary for the labels used in Figure 8.1. The TIGER graph in Figure 8.1 indicates by means of crossing edges that *Geschäftemachen* and *nicht die Politik* form a discontinuous coordinated constituent, which wraps around the rest of the sentence. This information is represented in terms of traces in the corresponding tree in Figure 8.3. Note that the functional information encoded in the TIGER graphs is preserved in the conversion process in terms of labels (-SB, -HD, etc.) on tree node categories.

[1]Thanks to Michael Schiehlen who provided the code to convert the graphs into corresponding trees with traces.



" Geschäftemachen ist seine Welt und nicht die Politik . "
" Business is his world and not the politics . "

" Business is his world, not politics."

Figure 8.1: TIGER graph #45, containing crossing edges

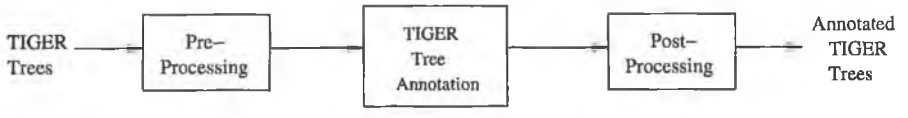


Figure 8.2: The process of annotating a TIGER tree with f-structure information

8.2.2 Annotation of Derived Trees

The annotation of the trees derived from the TIGER graphs is a two-stage process, with a pre- and post-processing phase (see Figure 8.2).

The preprocessing is a simple walk through the tree in order to build a lookup table for the trace nodes. This is required since often the trace occurs before the coindexed node in the tree and the information on the coindexed node realising the displaced material is necessary to assign an f-structure equation to the trace node. Table 8.2 presents the lookup table generated by the tree in Figure 8.3.

The first stage of the TIGER tree annotation attempts to assign an f-structure equation to each node based on the TIGER functional labels present in the tree. We have compiled

Trace	Trace Function	Node Number	Node Label
T1	CD	12	KON
T2	CJ	13	NP

Table 8.2: The lookup table generated in the pre-processing stage for the tree in Figure 8.3

```
(
  ($*LRB* ``)
  (S
    (CNP-SB
      (NN-CJ Geschäftemachen)
      (*T1*-CD -)
      (*T2*-CJ -)
    )
    (VAFIN-HD ist)
    (NP-PD
      (PPOSAT-NK seine)
      (NN-NK Welt)
    )
    (KON-*T1* und)
    (NP-*T2*
      (PTKNEG-NG nicht)
      (ART-NK die)
      (NN-NK Politik)
    )
  )
  ($. .)
)
```

"Geschäftemachen ist seine Welt und nicht die Politik."

Figure 8.3: TIGER graph #45 transformed into a Penn-II style tree with traces and co-indexation

an f-structure equation lookup table which assigns default f-structure equations triggered by each TIGER functional label. For example, the default entry for the SB (subject) label is ↑ SUBJ=↓. Table 8.3 gives the complete set of default annotations.

Default annotation is driven purely by TIGER functional annotations and sometimes overgenerates. It is possible to overwrite the default annotations. For example, the NK label (noun kernel element) alone is often ambiguous, though given some context, it is often straightforward to determine the f-structure equation required, e.g. an ART (article) node

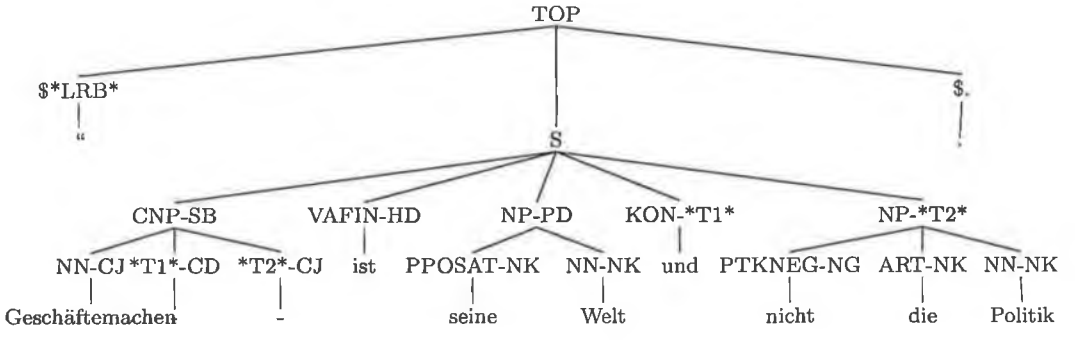


Figure 8.4: A graphical representation of the tree in Figure 8.3

with an NK label can usually be annotated $\uparrow$ SPEC:DET $= \downarrow$ (as in Figure 8.5). The second stage of the TIGER tree annotation involves overwriting default annotations in certain situations. These include:

- Determining the object of pre- and post-positions, labelled AC (adpositional case marker);
- Determining the behaviour of the CP (complementiser) labelled node;[2]
- Determining the head of a coordination phrase with more than one coordinating conjunction.

Figures 8.5 and 8.6 illustrate how the flat TIGER analysis of a German PP can be annotated to give the correct f-structure analysis (with the preposition taking an OBJ(ect) argument) by overwriting the default annotations on the trees. This is needed to encode the preferred hierarchical analysis of PPs.

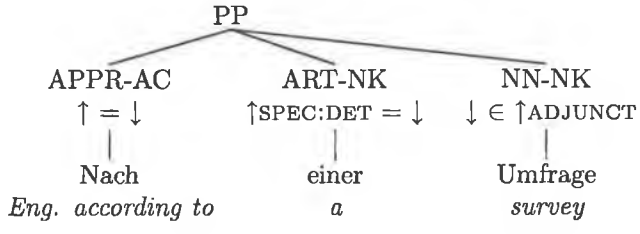


Figure 8.5: A flat analysis of a German PP and its default f-structure annotations

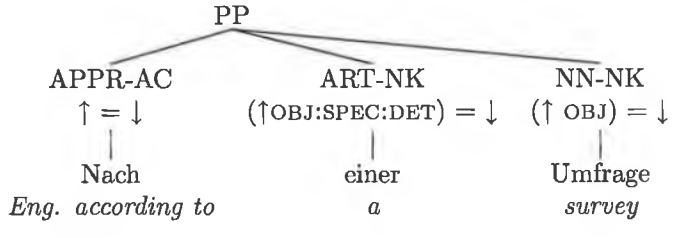


Figure 8.6: A flat analysis of a German PP and its correct f-structure annotations after stage two of the TIGER tree annotation process

[2]In our analysis, true complementisers, e.g. *daß* and *ob*, only contribute a COMP-FORM feature to the f-structure, whereas other conjunctions contribute a semantic form that governs linguistic material.

Functional Label	Description	Default Annotation
AC	Adpositional case marker	↑=↓
ADC	Adjective component	↓∈↑adjunct
AG	Genitive attribute	↑spec:poss=↓
AMS	Measure argument of adjective	↓∈↑adjunct
APP	Apposition	↓∈↑adjunct
AVC	Adverbial phrase component	↓∈↑adjunct
CC	Comparative complement	↑obl_compar=↓
CD	Coordinating conjunction	↑=↓
CJ	Conjunct	↓∈↑conj
CM	Comparative conjunction	↑=↓
CP	Complementiser	↑=↓
CVC	Collocational verb construction	↓∈↑adjunct
DA	Dative	↑obj2=↓
DH	Discourse-level head	↓∈↑adjunct
DM	Discourse marker	↓∈↑adjunct
EP	Expletive *es*	↑topic=↓
GL	Prenominal genitive	↑spec:poss=↓
GR	Postnominal genitive	↓∈↑adj_gen
HD	Head	↑=↓
JU	Junctor	↓∈↑adjunct
MNR	Postnominal modifier	↓∈↑adjunct
MO	Modifier	↓∈↑adjunct
NG	Negation	↓∈↑adjunct
NK	Noun kernel element	↓∈↑adjunct
NMC	Numerical component	↑number:spec=↓
OA	Accusative object	↑obj=↓
OA2	Second accusative object	↑aobj2=↓
OC	Clausal object	↑xcomp=↓
OG	Genitive object	↑obj_gen=↓
OP	Prepositional object	↑obl=↓
PAR	Paranthetical element	↓∈↑adjunct
PD	Predicate	↑xcomp_pred=↓
PG	Pseudo-genitive	↓∈↑adj_gen
PH	Placeholder	↓∈↑adjunct
PM	Morphological particle	↓∈↑adjunct
PNC	Proper noun component	↓∈↑name_mod
RC	Relative clause	↑adj_rel=↓
RE	Repeated element	↑app_clause=↓
RS	Reported speech	↑comp=↓
SB	Subject	↑subj=↓
SBP	Passivised subject	↑obl_ag=↓
SP	Subject or predicate	↑subj=↓
SVP	Separable verb prefix	↑part_form=↓
UC	Unit component	↓∈↑adjunct
VO	Vocative	↓∈↑adjunct

Table 8.3: Default annotations for each functional label in the TIGER treebank

Finally, a post-processing stage explicitly links trace nodes and the reference node. This involves adding equations such as $\uparrow$XCOMP:OBJ $= \downarrow$ to nodes with trace information. For example, in Figure 8.7, the NP-*T2* node receives the annotation $\downarrow \in \uparrow$subj:conj.

```
(TOP
  ($*LRB* ``)
  (S[up=down]
    (CNP-SB[up-subj=down]
      (NN-CJ[down-elem=up:conj]
                    Geschäftemachen)
      (*T1*-CD -)
      (*T2*-CJ -)
    )
    (VAFIN-HD[up=down] ist)
    (NP-PD[up-xcomp_pred=down]
      (PPOSAT-NK[up-spec:poss=down] seine)
      (NN-NK[up=down] Welt)
    )
    (KON-*T1*[up:subj=down] und)
    (NP-*T2*[down-elem=up:subj:conj]
      (PTKNEG-NG[down-elem=up:adjunct] nicht)
      (ART-NK[up-spec:det=down] die)
      (NN-NK[up=down] Politik)
    )
  )
  ($. .)
)
```

Figure 8.7: The tree in Figure 8.3 after automatic annotation

```
subj : conj : 1 : pred : 'Geschäftemachen'
              2 : spec : det : pred : die
                  adjunct : 3 : pred : nicht
                  pred : 'Politik'
       coord_form : und
xcomp_pred : spec : poss : pred : pro
             pred : 'Welt'
pred : ist
```

Figure 8.8: The f-structure produced as a result of automatically annotating the tree in Figure 8.3

Figure 8.3 presents an input tree, Figure 8.7 the automatically annotated tree and Figure 8.8 provides the resulting f-structure.

8.2.3 Evaluation of the Automatic Annotation Algorithm

We first established the coverage of the annotation algorithm on the entire TIGER corpus. Table 8.4 presents the results. 96.86% of the 40,000 sentences receive one covering and connected f-structure. Ideally we would like to generate just one f-structure per sentence. There are, however, a number of sentences (1112) that receive more than one f-structure fragment. This is mainly due to strings such as *Bonn, 7. September*, where in the source TIGER graphs there is no clear relation between the individual constituents of the string and where we do not wish to enforce a relation for the sake of having fewer fragments. We believe that these strings are in fact fragments and should be treated accordingly. There are also a small number of sentences which do not receive any f-structure. This is as a result of feature clashes in the annotated trees, which are caused by inconsistent annotations.

We also evaluate the quality of the annotation against a manually constructed gold-standard of 100 f-structures. In our parsing experiments, we set aside sentences 8000-10000 of the TIGER treebank for testing purposes. We extracted 100 sentences at random from these 2000 sentences, in order to develop our gold standard. The original TIGER trees for these sentences were converted into dependency structures following Forst (2003a) and manually corrected by Martin Forst. We use the triple encoding and evaluation software of Crouch et al. (2002). Table 8.5 shows that currently our automatic f-structure annotation achieves a preds-only f-score of 90.22% against this gold standard, with precision about 7% higher than recall. Table 8.6 shows a more detailed analysis of how well the automatic f-structure annotation algorithm performs on specific functions. Most features achieve very high results, e.g. OBJ is 94%, XCOMP is 95%. The features that we score poorest on are TOPIC and APP-CLAUSE. However, there are not that many occurrences of these feature in the gold standard. We expect all figures to improve as we refine the f-structure annotation algorithm.

# f-str. frags	# sent	percent
0	143	0.3573
1	38765	96.8641
2	1032	2.5787
3	75	0.1874
5	1	0.0025
6	1	0.0025
7	3	0.0075

Table 8.4: Coverage & fragmentation results of German f-structure annotation algorithm

	Preds Only Evaluation
Precision	93.71%
Recall	86.99%
F-Score	90.22%
Complete Match	25

Table 8.5: Evaluation of the f-structures produced by automatically annotating the TIGER trees against 100 gold-standard f-structures

8.3 Parsing Experiments

Using the annotation method described above, we automatically annotate the TIGER corpus with f-structure equations. We then read off a grammar from the annotated treebank, resulting in an *annotated* PCFG (A-PCFG) for German. We use Helmut Schmid's `BitPar` parser (Schmid, 2004) to parse with this grammar, using Viterbi pruning to obtain the most probable parse. We collect the f-structure annotations from the resulting parse tree and use a constraint solver to produce an f-structure for new text. All experiments instantiate the integrated parsing architecture described in Chapter 6.

An annotated grammar (A-PCFG) was extracted from the TIGER corpus excluding the 2000 sentences set aside for testing. Prior to grammar extraction, empty productions were removed from the TIGER trees while TIGER functional labels were kept. The grammar contains 65,758 rules. We also transformed the grammar using a parent transformation (Johnson, 1999) to give us PA-PCFG with 72,127 rules. Using these two grammars, we parsed the 2000 raw, untagged test strings. The results are presented in Table 8.7. We evaluated the quality of the trees produced by the parser and measure how many of the 2000 sentences produce one covering and connected f-structure. Currently 95.47% of the

Dependency	Precision	Recall	F-score
circ-form	2/2 = 100	2/2 = 100	100
dem	7/7 = 100	7/7 = 100	100
obj2	5/5 = 100	5/5 = 100	100
obl	11/11 = 100	11/11 = 100	100
obl-ag	6/6 = 100	6/6 = 100	100
part-form	11/11 = 100	11/11 = 100	100
adj-gen	79/81 = 98	79/82 = 96	97
coord-form	57/58 = 98	57/59 = 97	97
pron-type	16/16 = 100	16/17 = 94	97
det	269/277 = 97	269/283 = 95	96
xcomp	74/77 = 96	74/78 = 95	95
name-mod	29/33 = 88	29/29 = 100	94
obj	315/329 = 96	315/339 = 93	94
xcomp-pred	30/31 = 97	30/35 = 86	91
adjunct	538/584 = 92	538/605 = 89	90
conj	122/145 = 84	122/138 = 88	86
comp-form	11/11 = 100	11/15 = 73	85
comp	14/15 = 93	14/18 = 78	85
adj-rel	12/14 = 86	12/15 = 80	83
number	15/18 = 83	15/18 = 83	83
app	25/26 = 96	25/35 = 71	82
poss	16/18 = 89	16/22 = 73	80
obl-compar	3/5 = 60	3/3 = 100	75
subj	148/151 = 98	148/244 = 61	75
quant	12/14 = 86	12/22 = 55	67
app-clause	5/9 = 56	5/7 = 71	63
topic	0/1 = 0	0/0 = 0	0

Table 8.6: Preds-only evaluation of Automatically Annotating TIGER trees broken down by features

2000 sentences parsed with A-PCFG, and 97.9% of those parsed by PA-PCFG receive one covering and connected f-structure. A more detailed breakdown of the number of fragments per sentence is presented in Table 8.8. We evaluate the quality of the f-structures produced in two ways. First we evaluate against our manually constructed gold standard of 100 f-structures. Second, in a CCG-style experimental setup (Hockenmaier, 2003), we automatically annotate the 2000 held-out original treebank trees with our f-structure annotation algorithm, and evaluate the output of the parser for the 2000 raw, untagged strings against the *automatically* produced f-structures. We currently achieve a labelled f-score of 69.39% on the trees produced by A-PCFG. For the same grammar, the f-structures achieve an f-score of 71% and 74.61% on the 100 gold standard f-structures and the 2000 automatically produced f-structures respectively. Unlike for English, applying the parent transformation (Johnson, 1999) to the German grammar does not improve results: there

	A-PCFG	PA-PCFG
# Rules	65758	72127
# Parses	1993	1990
Lab. F-Score (2000 Trees)	69.39%	68.14%
Unlab. F-Score (2000 Trees)	73.72%	73.16%
Tagging Accuracy (2000 Trees)	95.47%	80.20%
Fragmentation (2000 f-structures)	95.8%	97.9%
F-Score(100 f-structures)	71.00%	70.50%
F-Score(2000 f-structures)	74.61%	74.04%

Table 8.7: Parsing Results

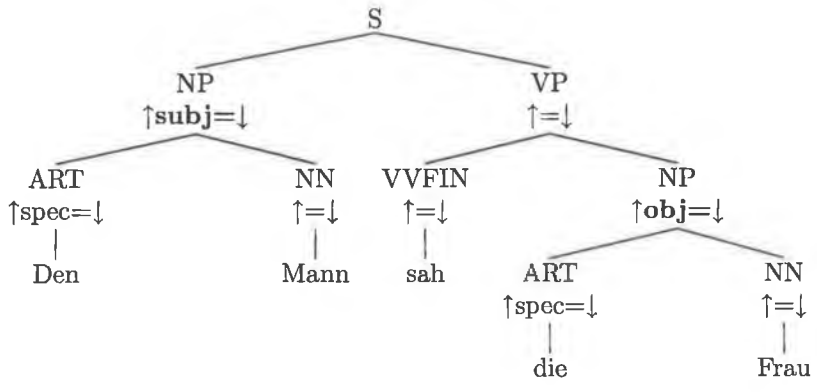
is a decrease of 0.5% in labelled f-score against the 100 gold-standard f-structures, and a decrease of 0.57% against the 2000 f-structures. In fact, the only respect in which PA-PCFG outperforms the A-PCFG is in fragmentation, i.e. the f-structures it produces are less fragmented, with a decrease of 2.1% in fragmentation from A-PCFG to PA-PCFG.

	A-PCFG		PA-PCFG	
# f-structure fragments	# sent	percent	# sent	percent
0	23	1.15	34	1.7
1	1916	95.8	1958	97.9
2	56	2.8	8	0.4
3	5	0.25		

Table 8.8: Coverage & fragmentation results of parsing with the annotated grammar

8.4 Adding Morphological Information

Morphology is extremely important in determining functional information in non-configurational languages such as German. In the acquired grammars outlined above, we do not consider the rich arsenal of morphological information potentially available to us. For example, in (2), the accusative case marking on *den Mann* (the man-acc) indicates that it is the object of the verb *sehen* (to see), although it occurs in a position more usually filled by a subject. Consider the parse tree in Figure 8.9 that might be produced for this sentence by the parser described in Section 8.3. The parser does not recognise that *den Mann* is the object. By adding case information to constituents that encode case morphologically (ARTnom, ARTacc, ...), the parser has an improved chance to rectify the situation and produce the correct tree, with proper grammatical function assignment,



Den Mann sah die Frau
the-acc man-acc saw the-nom woman-nom

It was the man the woman saw

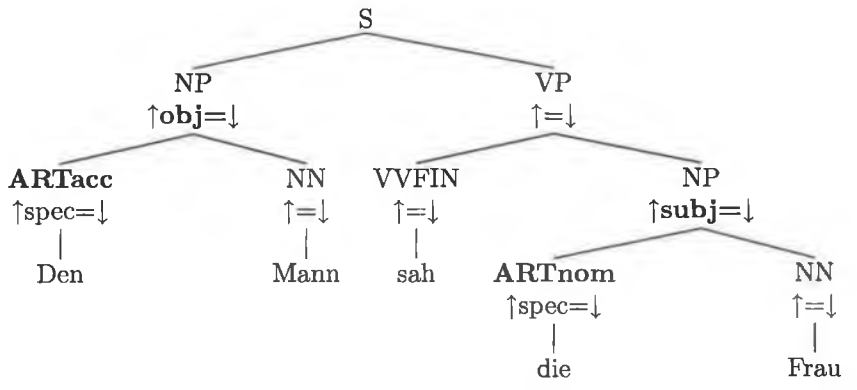
Figure 8.9: An example of where the parser does not recognise functional information encoded in morphological case

as in Figure 8.10.

(2) Den Mann sah die Frau.
The-acc man-acc saw the-nom woman-nom
It was the man the woman saw

8.4.1 Automatically Simulating Morphological Information in TIGER Trees

At the time of submission of the present dissertation, the morphological analysis for TIGER treebank trees was not completed. We hope to run further experiments when the morphologically-annotated version of the treebank becomes available. Until this is the case, however, we have experimented with a method for automatically simulating the effects of morphological case information where this can be ascertained. In order to do this, we simply exploit functional annotation labels in TIGER trees and percolate case information deduced from functional TIGER annotations downwards from the maximal projection in an automatic grammar transformation step. Table 8.9 illustrates the functional labels that trigger morphological case and their associated cases. Whenever one of these labels is encountered, the algorithm projects the case feature associated with it



Den Mann sah die Frau
the-acc man-acc saw the-nom woman-nom

It was the man the woman saw

Figure 8.10: An example of where the parser does recognise functional information encoded in morphology

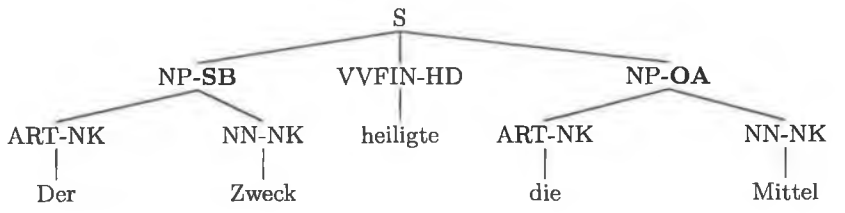
down to the lowest projection and augments certain POS tags with the relevant case information. Figures 8.11 and 8.12 illustrate the effect of the transformation on TIGER trees. Only the POS tags listed in Table 8.10 are annotated with case information.

Functional Label	Description	Associated Case
AG	Genitive Attribute	genitive
DA	Dative	dative
OA	Accusative Object	accusative
OA2	Second Accusative Object	accusative
OG	Genitive Object	genitive
PD	Predicate	nominative
SB	Subject	nominative
SP	Subject or Predicate	nominative

Table 8.9: Table of Functional Tags and their associated morphological cases

8.4.2 Experiments and Results

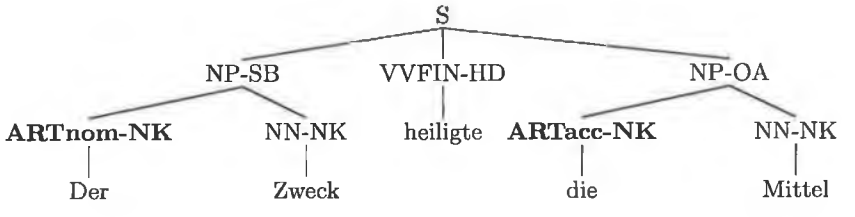
We extracted two grammars, CA-PCFG and CPA-PCFG (with additional parent transformation) after the case transformation. CA-PCFG has 66,110 rules, while CPA-PCFG has 72,441, in comparison with 65,758 and 72,127 for A-PCFG and PA-PCFG. We ran



Der Zweck heiligte die Mittel
the-nom purpose-nom sanctifies the-acc means-acc

The end justifies the means

Figure 8.11: A TIGER tree before case simulating grammar transformation



Der Zweck heiligte die Mittel
the-nom purpose-nom sanctifies the-acc means-acc

The end justifies the means

Figure 8.12: A TIGER tree after case simulating grammar transformation

POS Tag	Description
ADJA	Adjective, attributive
ART	Definite or indefinite article
PDS	Substituting demonstrative pronoun
PDAT	Attributive demonstrative pronoun
PIS	Substituting indefinite pronoun
PIAT	Attributive indefinite pronoun without determiner
PIDAT	Attributive indefinite pronoun with determiner
PPER	Non-reflexive personal pronoun
PPOSS	Substituting possessive pronoun
PPOSAT	Attributive possessive pronoun
PRELS	Substituting relative pronoun
PRF	Reflexive personal pronoun
PWS	Substituting interrogative pronoun
PWAT	Attributive interrogative pronoun

Table 8.10: TIGER POS tags that receive case annotations

the same experiments as outlined in Section 8.3. We measure the quality of the trees produced, what percentage of the held-out 2000 test sentences produce one covering and connected f-structure (fragmentation), the quality of the f-structures generated by the parsers against the manually constructed 100 reference f-structures, and (in a CCG-style experiment (Hockenmaier, 2003)) against the held-out 2000 original treebank trees automatically annotated by our f-structure annotation algorithm. The results are presented in Tables 8.11 and 8.12. Using the CA-PCFG, 1993 sentences receive a parse. We obtain an f-score of 68.88 on the labelled trees. 1915 (95.75%) of the 2000 sentences receive one covering and connected f-structure. 1.15% of the sentences do not receive any f-structure. When we evaluate against our manually constructed gold standard, we achieve an f-score of 70.08. This is an increase of 0.08% on the same grammar without case simulation. Automatically annotating the original 2000 original treebank trees and comparing the parser output against it, produces an f-score of 74.56%. This does not improve on our previous experiments without case simulation. I expect there to be two main reasons responsible for this result: lack of coverage and the fine-grainedness of the grammar transformation.

Currently the addition of morphological case information is not complete enough. There are many instances where we cannot reliably determine case information, as in many prepositional phrases for example.[3] In the cases where we are able to add case information, this is triggered by TIGER functional labels, and these functional labels alone seem to do just as well as the functional labels plus the automatically generated case marking. It remains to be seen whether more fine-grained automatic case simulation can produce better results. Morphological case marking in German is in fact sensitive to grammatical function, number and gender (see Table 8.13), and not just to grammatical function as in our simulation. If this information is available in TIGER trees, a more fine-grained case marking grammar transformation can be attempted. I would expect that reliable addition of case information is a valuable grammar transformation, especially in instances where arguments do not appear in their "expected" positions (e.g. in stressed positions as in Example 2). The experiments show that the simple automatic case simulation presented here does not yield improved parsing results. Of course, when the proper

[3]There are a number of German prepositions that can take either accusative or dative case, and from the context in the TIGER trees, we cannot determine which case it is.

morphological analysis for the entire TIGER treebank is available, it will be possible to carry out more extensive experiments. One experiment would examine to what extent the morphology alone (ignoring functional labels) can be used to automatically annotate the TIGER treebank with f-structure information in order to carry out further parsing experiments.

	CA-PCFG	CPA-PCFG
# Rules	66110	72441
# Parses	1993	1979
Lab. F-Score (2000 Trees)	68.88%	67.23%
Unlab. F-Score (2000 Trees)	73.22%	72.31%
Fragmenation (2000 f-structures)	95.75%	97.3%
Tagging Accuracy (2000 Trees)	95.05%	79.85%
F-Score(100 f-structures)	71.08%	69.25%
F-Score(2000 f-structures)	74.56%	73.59%

Table 8.11: Parsing Results with Case Simulation

	CA-PCFG		CPA-PCFG	
# f-str. frags	# sent	percent	# sent	percent
0	23	1.15	43	2.15
1	1915	95.75	1946	97.3
2	57	2.85	11	0.55
3	5	0.25		

Table 8.12: Coverage & fragmentation results of parsing with the annotated grammar including Morphological Information

	Masc.	Fem.	Neut.	
NOM	der Mann	die Frau	das Kind	
ACC	den Mann	die Frau	das Kind	SG
GEN	des Mannes	der Frau	des Kindes	
DAT	dem Mann	der Frau	dem Kind	
NOM	die Männer	die Frauen	die Kinder	
ACC	die Männer	die Frauen	die Kinder	PL
GEN	der Männer	der Frauen	der Kinder	
DAT	den Männern	den Frauen	den Kindern	

Table 8.13: Grammatical function, number and gender influence case marking in German as this table illustrates

8.5 Summary

In this chapter, I showed how the methodology for developing large-scale, PCFG-based English LFG grammar approximations induced from treebanks can be adapted to German. The method constitutes a novel approach to deep multilingual unification grammar acquisition based on treebank resources and automatic f-structure annotation algorithms and can offer substantially reduced grammar development cost where a treebank is available. Depending on the size of the treebank, the method can deliver robust and wide-coverage unification grammars.

The method, originally developed and tested on English and the Penn-II treebank, has been adapted to German and the TIGER treebank resource. We parsed 2000 sentences with the LFG approximation extracted automatically from the f-structure annotated TIGER treebank. We evaluated parser output against manually constructed gold-standard f-structures for 100 sentences and, against the full 2000 f-structures generated by the automatic f-structure annotation algorithm for the original held-out 2000 treebank trees in a CCG-style experiment. The basic A-PCFG achieves an f-score of 71% against the 100 gold standard f-structures and 74.6% against the 2000. We automatically add some morphological case information in a case simulating grammar transformation experiment that improves f-score against the manually annotated gold standard by 0.08%. F-score on the 2000 automatically annotated original treebank trees decreases slightly by 0.05%. I believe this is because the grammar transformation is too coarse-grained, since in many cases we are not able to determine case information from the functional labels alone. I expect more sophisticated grammar transformations to produce improved results and hope to explore this in future research. Once the proper morphological analysis of the treebank becomes available, we will carry out further experiments and comparisons with grammar transformations. The grammars reported on here achieve more than 95.75% coverage on unseen TIGER treebank data after a total of just over three person months of development effort. By contrast, a hand-crafted, fine-grained, wide-coverage German LFG grammar currently achieves approximately 70% coverage (measured as full spanning parse) after several person years of development effort (Forst, 2003a).

Chapter 9

Conclusions

Manual development of wide-coverage, deep, constraint-based grammatical resources that scale to unrestricted text is knowledge-intensive, time-consuming and expensive. This thesis is part of a larger project to automate wide-coverage, deep, constraint-based grammar development using treebank resources and automatic f-structure annotation algorithms to address the knowledge-acquisition bottleneck in constraint-based grammar development.

The work presented in this thesis has:

- contributed to the development, implementation and evaluation of an automatic f-structure annotation algorithm for the Penn-II treebank (Cahill et al., 2002a; Burke et al., 2004a,b).
- provided the corpus inspection, search and visualisation tools (TTS) (Cahill and van Genabith, 2002) as well as the f-structure annotation algorithm application and visualisation tools (FSAT). TTS has been essential in establishing the linguistic basis which seeds the f-structure annotation algorithm. FSAT has been essential in monitoring and validating the application of the automatic f-structure annotation algorithm to the treebank.
- investigated the interaction between treebank pre-processing and transformation steps on PCFG grammars and parsing performance.
- developed and evaluated two PCFG-based parsing architectures (pipeline and integrated) for parsing raw text into proto f-structures (with LDDs unresolved) using

the f-structure annotation algorithm (Cahill et al., 2002c).

- developed and evaluated an LDD resolution method on f-structure based on finite approximations of functional uncertainty equations and lexical resources (subcategorisation frames) automatically acquired from the f-structure annotated Penn-II treebank resource (Cahill et al., 2004b).
- compared the results obtained against other state-of-the-art approaches (Collins, 1999; Johnson, 2002; Riezler et al., 2002; Kaplan et al., 2004).
- integrated external state-of-the-art parsing technology (Collins, 1999; Charniak, 2000) into the pipeline processing model
- applied and evaluated the automatic wide-coverage, constraint-based grammar extraction and parsing methods to German and the TIGER treebank resource (Cahill et al., 2003a).

The main, perhaps surprising, result is that our treebank-based, automatic, deep, constraint-based grammar acquisition method together with simple (strictly speaking, mathematically inadequate) but flexible PCFG-based processing architectures outperforms the best hand-crafted resources and sophisticated processing techniques. Currently, our best result is 81.79% f-score against the PARC 700, while Kaplan et al. (2004) report a score of 79.6%. Below I summarise these and the other main results obtained.

PCFG-based techniques are a core part of this dissertation. We explore a number of grammar transformations that can be applied to treebanks before the induction of PCFGs and examine how they interact. This indicates what grammar transformations and preprocessing steps might be useful prior to grammar extraction. Our experiments show that grammars that have been parent-transformed, or that contain LFG f-structure equations perform well, with an increase of between 5.6% and 8.8% over a baseline grammar. Grammars with both parent transformation and f-structure information perform even better, with the best grammar achieving 81.27% labelled f-score on trees against section 23 of the Penn-II treebank.

We develop two architectures (pipeline and integrated) that parse unseen text into proto f-structures. Proto f-structure are basic, but possibly incomplete, predicate-

argument structures with long-distance dependencies (LDDs) unresolved. We evaluate the proto f-structures, and against the DCU 105, our best grammar in the pipeline model achieves an f-score of 74.25% preds-only and 79.81% on all grammatical functions (GFs). Our best grammar in the integrated architecture achieves an f-score of 74.8% preds-only and 81.2% on all GFs. In a CCG-style experiment (Hockenmaier, 2003), against the 2,416 automatically generated f-structures for the original treebank trees in section 23, the pipeline model achieves higher results than the integrated model (75.67% preds-only and 81.74% all GFs). We evaluate what percentage of f-structures receive a covering and connected f-structure. Generally, for our best performing grammars, 99.8% receive one covering and connected f-structure and less than 0.2% of the sentences do not receive any or more than one f-structure fragments.

The resolution of LDDs is crucial in the mapping of text to information (represented in terms of predicate-argument or dependency structure). Most PCFG-based parsing does not deal with LDDs. We present a method of resolving LDDs at f-structure level using finite approximations of functional uncertainty equations and automatically acquired subcategorisation frames (O'Donovan et al., 2004), both extracted from the f-structure annotated Penn-II treebank. This allows our parsers to produce deep linguistic representations. Currently, the quality of the preds-only f-structures produced by our best grammars after LDD resolution improves results over the same grammars without LDD resolution by between 5.55% and 6.46% against the DCU 105. Our best grammar achieves 81.24% preds-only f-score and 87.04% f-score for all GFs. Evaluating against the PARC 700 Dependency Bank following the evaluation described in Kaplan et al. (2004), our best grammar achieves an f-score of 80.33%. The improvement in f-structure f-score after LDD resolution indicates that our LDD resolution algorithm is performing well. However, we also need to evaluate the LDD resolution algorithm independently. We evaluate the quality of the re-entrancies in dependency structures and show that our best grammar achieves an overall f-score of 82.81% on LDD resolution paths against the DCU 105.

It is interesting to compare the research presented in this thesis to other approaches. We compare against three bodies of work: Collins (1999), Johnson (2002) and Riezler et al. (2002); Kaplan et al. (2004). It is difficult to perform a satisfactory comparison

between our work and Collins (1999) or Johnson (2002), as they add empty nodes to parse trees to encode long-distance dependencies, while we resolve LDDs at the level of f-structure. We carry out a number of experiments at the level of f-structure in our pipeline parsing architecture. Our results show that our LDD resolution on f-structure outperforms Collins' (1999) Model 3 and Johnson's (2002) algorithm for recovering empty nodes and their antecedents. In order to compare our work with the hand-crafted wide-coverage LFG grammars of Riezler et al. (2002) and Kaplan et al. (2004), we map our f-structures to a format similar to the PARC 700 and evaluate against a subset of the PARC 700 with a reduced feature set. Our best automatically induced grammar achieves an f-score of 80.33%, against 79.6% for the best hand-crafted grammar presented in Kaplan et al. (2004). Using the output of Charniak's parser in our pipeline parsing architecture, we achieve an f-score of 81.79% against the PARC 700, an improvement of 2.19% over the best results presented in Kaplan et al. (2004). These results show that our treebank and automatic f-structure annotation algorithm-based grammar acquisition and PCFG-based parsing architectures can be used to induce high-quality LFG resources, currently outperforming even the best manually created resources and sophisticated (log-linear-based) processing architectures.

Our methodology can be ported to other languages and treebanks, and we carry out experiments on German and the TIGER treebank. With just three months of development effort, we created a German LFG approximation that can parse unseen newspaper text into f-structures. Our parser achieves an f-score of 71% against a gold standard of 100 randomly selected sentences and achieves coverage (measured in terms of covering and connected f-structures) of over 95%.

Nothing in the methodology presented here precludes its application to other languages, corpora or formalisms. Suitable annotation schemes could also be applied to automatically derive HPSG typed feature structures (Miyao et al., 2003), dependency structures or logical forms (Cahill et al., 2003b). We believe that our approach can provide an attractive, wide-coverage, multilingual constraint-based grammar acquisition paradigm, complementing and, in certain cases, replacing, more traditional, manual grammar development.

9.1 Future Work

In both of our parsing architectures, quality of the parsing output correlates with the quality of the f-structures generated. We are experimenting with a number of methods of improving parser output. This feeds into our parsing architectures in two ways. First, we can improve the quality of our own grammars using standard PCFG parsing technology (Schmid, 2004), and second, we can incorporate external, state-of-the-art parsers (Collins, 1999; Charniak, 2000) into our pipeline architecture to produce high quality f-structures. We aim to improve the quality of our grammars by investigating further grammar transformations. Klein and Manning (2003) present a number of linguistically motivated grammar transformations that significantly improve parsing results. We hope to incorporate at least some of them into future work. Our research into the case-simulating grammar transformation for German proved inconclusive. In ongoing work we hope to refine the automatic case-simulation transformation, and when the proper morphological information for the TIGER treebank becomes available, we will perform case-sensitive grammar transformation and extraction experiments.

It is well known that PCFG-based approximations of unification grammars do not provide an adequate probability model (Abney, 1997), as sometimes probability mass is lost when the parser produces a most probable parse, but this parse cannot generate an f-structure. We do not claim to provide an adequate probability model; rather we have taken an engineering approach to the problem, exploiting simple but effective approximations. In future work, we hope to explore more complex parsing models, in particular log-linear-based disambiguation models (Riezler et al., 2002; Miyao et al., 2003). While we have shown that our grammar induction and PCFG-based techniques perform well (in fact better than the current best hand-crafted grammars and sophisticated processing architectures), the probability model is, strictly speaking, inconsistent, and we expect an improved probability model to yield improved results. Another area of future research is to improve parsing by using subcategorisation frames. O'Donovan et al. (2004) show that large-scale, high-quality subcategorisation frames can be automatically extracted from the f-structure-annotated Penn-II treebank. While we have used these frames in our LDD resolution algorithm, they have not been incorporated into our PCFG parsing model. Carroll

et al. (1998) show that subcategorisation probabilities can significantly improve parse accuracy, and we hope to also improve parse accuracy using our automatically acquired subcategorisation frames.

Our long-distance dependency resolution algorithm is perhaps too coarse-grained. We only distinguish between two types of TOPICRELS, but more fine-grained distinctions are possible, e.g. determiner-case TOPICREL as in *customers* ***whose*** *addresses have changed.* We hope to refine our LDD resolution algorithm to incorporate more fine-grained path distinctions. We have not yet investigated long-distance dependency resolution in our German LFG approximations, another area we hope to explore in the future.

This thesis has shown that large-scale high-quality probabilistic LFG approximations can be automatically acquired from treebanks. However, the f-structures that our grammars generate do not quite feature the rich fine-grained feature set of hand-crafted grammars. This poses an interesting area of future research: can the automatically acquired treebank-based LFG approximations be used to bootstrap manual grammar writing? This would be particularly interesting for languages where a treebank is available but no hand-crafted grammar already exists, such as Chinese, Spanish or Arabic. Languages such as English and German that already have large hand-crafted LFGs could use the treebank-based LFG approximations as a fall back to improve coverage, one of the main weaknesses of hand-crafted grammars.

To date, grammars automatically induced from treebanks have not been used in probabilistic generation. The main reason for this is that standard PCFGs do not associate strings with meaning representations. This thesis has shown how our PCFG-based LFG approximations automatically acquired from treebanks can associate strings with meaning representations in the form of f-structures approximating to basic predicate-argument structures. The "refinement" grammars of Kaplan and Wedekind (2000) bear some similarity to our automatically generated treebank-based PCFG LFG approximations. We hope to combine our work on automatically annotated treebanks and extracted PCFG-based LFG approximations with the theoretical work on LFG generation of Kaplan and Wedekind (2000) to investigate a treebank-unification-grammar-based probabilistic generation framework.

Bibliography

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

Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation and Compiling: Volumn 1: Parsing.* Prentice-Hall, Englewood Cliffs, NJ.

Berger, A. L., Della Pietra, S. A., and Della Pietra, V. J. (1996). A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, **22**(1):39–71.

Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., and Roukos, S. (1992). Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings, DARPA Speech and Natural Language Workshop*, pages 134–139, Harriman, NY.

Bod, R. and Kaplan., R. M. (1998). A probabilistic corpus-driven model for lexical-functional analysis. In *Proceedings of COLING/ACL98: Joint Meeting of the 36th Annual Meeting of the ACL and the 17th International Conference on Computational Linguistics*, pages 145–151, Montréal, Canada.

Bod, R. and Scha, R. (2003). A DOP Model for Phrase-Structure Trees. In Bod, R., Scha, R., and Sima'an, K., editors, *Data-Oriented Parsing*, pages 13–24, Stanford, CA. CLSI Publications.

Brants, T., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER Treebank. In Hinrichs, E. and Simov, K., editors, *Proceedings of the first Workshop on Treebanks and Linguistic Theories (TLT'02)*, pages 24–41, Sozopol, Bulgaria.

Bresnan, J. (2001). *Lexical-Functional Syntax.* Blackwell, Oxford.

Briscoe, T. and Carroll, J. (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, **19**(1):25–59.

Burke, M. (forthcoming). *Automatic Annotation of the Penn-II Treebank with F-Structure Information.* PhD thesis, School of Computing, Dublin City University, Dublin, Ireland.

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J., and Way, A. (2004a). The Evaluation of an Automatic Annotation Algorithm against the PARC 700 Dependency Bank. In *Proceedings of the Ninth International Conference on LFG*, Christchurch, New Zealand (to appear).

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J., and Way, A. (2004b). Treebank-Based Acquisistion of Wide-Coverage, Probabilistic LFG Resources: Project Overview, Results and Evaluation. In *The First International Joint Conference on Natural Language Processing (IJCNLP-04), Workshop "Beyond shallow analyses - Formalisms and statistical modeling for deep analyses"*, Hainan Island, China.

Butt, M., Dyvik, H., King, T. H., Masuichi, H., and Rohrer, C. (2002). The Parallel Grammar Project. In *Proceedings of COLING 2002, Workshop on Grammar Engineering and Evaluation*, pages 1–7, Taipei, Taiwan.

Butt, M., King, T. H., Niño, M. E., and Segond, F. (1999). *A Grammar Writer's Cookbook.* CSLI Publications, Stanford, CA.

Cahill, A., Burke, M., McCarthy, M., O'Donovan, R., van Genabith, J., and Way, A. (2004a). Evaluating Automatic F-Structure Annotation for the Penn-II Treebank. In Hinrichs, E. and Simov, K., editors, *Journal of Language and Computation; Special Issue on "Treebanks and Linguistic Theories"*. Kluwer Academic Press (to appear).

Cahill, A., Burke, M., O'Donovan, R., van Genabith, J., and Way, A. (2004b). Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG Approximations. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 320–327, Barcelona, Spain.

Cahill, A., Forst, M., McCarthy, M., O'Donovan, R., Rohrer, C., van Genabith, J., and Way, A. (2003a). Treebank-Based Multilingual Unification-Grammar Development. In

Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Development, at the 15th European Summer School in Logic Language and Information, pages 17–24, Vienna, Austria.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2002a). Automatic Annotation of the Penn Treebank with LFG F-Structure Information. In *Proceedings of the LREC Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data*, pages 8–15, Las Palmas, Canary Islands, Spain.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2002b). Evaluating Automatic F-Structure Annotation for the Penn-II Treebank. In Hinrichs, E. and Simov, K., editors, *Proceedings of the Treebanks and Linguistic Theories (TLT'02) Workshop*, pages 42–60, Sozopol, Bulgaria.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2002c). Parsing with PCFGs and Automatic F-Structure Annotation. In Butt, M. and King, T. H., editors, *Proceedings of the Seventh International Conference on LFG*, pages 76–95, Stanford, CA. CSLI Publications.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2003b). Quasi-Logical Forms for the Penn Treebank. In Bunt, H., van der Sluis, I., and Morante, R., editors, *Proceedings of the Fifth International Workshop on Computational Semantics, IWCS-05*, pages 55–71, Tilburg, The Netherlands.

Cahill, A. and van Genabith, J. (2002). TTS – A Treebank Tool Suite. In Rodriguez, M. and Arnajo, C. S., editors, *LREC 2002, The Third International Conference on Language Resources and Evaluation*, volume **V**, pages 1712–1717, Las Palmas de Grand Canaria, Spain.

Calder, J. (2000). Thistle and Interarbora. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 992–996, Saarbruecken, Germany.

Carroll, G. and Rooth, M. (1998). Valence induction with a head-lexicalized PCFG. In

Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP 3), pages 36–45, Granada, Spain.

Carroll, J., Minnen, G., and Briscoe, E. (1998). Can subcategorisation probabilities help a statistical parser? In *Proceedings of the 6th ACL/SIGDAT Workshop on Very Large Corpora*, pages 118–126, Montréal, Canada.

Charniak, E. (1996). Tree-Bank Grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1031–1036, Menlo Park, CA.

Charniak, E. (1997). Statistical Parsing with a Context-Free Grammar and Word Statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Providence, RI.

Charniak, E. (2000). A maximum entropy inspired parser. In *Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2000)*, pages 132–139, Seattle, WA.

Collins, M. (1997). Three Generative, Lexicalized Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, Philadelphia, PA.

Crouch, R., Kaplan, R., King, T. H., and Riezler, S. (2002). A comparison of evaluation metrics for a broad coverage parser. In *Proceedings of the LREC Workshop: Beyond PARSEVAL – Towards Improved Evaluation Measures for Parsing Systems*, pages 67–74, Las Palmas, Canary Islands, Spain.

Dalrymple, M. (2001). *Lexical-Functional Grammar.* San Diego, CA; London. Academic Press.

Egedi, C. D. D., Hockey, B. A., Srinivas, B., and Zaidel, M. (1994). Xtag system - a wide coverage grammar for english. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING 94)*, pages 922–928, Kyoto, Japan.

Forst, M. (2003a). Treebank Conversion - Creating an f-structure bank from the TIGER Corpus. In Butt, M. and King, T. H., editors, *Proceedings of the Eighth International Conference on LFG*, pages 205–216. CSLI Publications, Stanford, CA.

Forst, M. (2003b). Treebank Conversion - establishing a test suite for a broad-coverage LFG from the TIGER treebank. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora (LINC'03)*, pages 25–32, Budapest, Hungary.

Frank, A. (2000). Automatic F-Structure Annotation of Treebank Trees. In Butt, M. and King, T. H., editors, *Proceedings of the Fifth International Conference on LFG*, pages 140–160, Stanford, CA. CSLI Publications.

Frank, A., Sadler, L., van Genabith, J., and Way, A. (2003). From Treebank Resources to LFG F-Structures. In Abeille, A., editor, *Treebanks: Building and Using Syntactically Annotated Corpora*, pages 367–389. Kluwer Academic Publishers, Dordrecht/Boston/London,The Netherlands.

Gazdar and Mellish (1989). *Natural Language Processing in PROLOG: An Introduction to Computational Linguistics.* Addison-Wesley Publishing Co., Wokingham, England.

Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford, UK.

Hindle, D. and Rooth, M. (1993). Structural Ambiguity and Lexical Relations. *Computational Linguistics*, **19**(1):103–120.

Hockenmaier, J. (2003). Parsing with Generative models of Predicate-Argument Structure. In *Proceedings of the 41st Annual Conference of the Association for Computational Linguistics*, pages 359–366, Sapporo, Japan.

Hockenmaier, J. and Steedman, M. (2002). Generative Models for Statistical Parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 335–342, Philadelphia, PA.

Inui, K., Sornlertlamvanich, V., Tanaka, H., and Tokunaga, T. (1997). A new formalization

of probabilistic GLR parsing. In *Proceedings of the 5th International Workshop on Parsing Technologies*, pages 123–134, Boston, MA.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar.* CSLI Lecture Notes, Chicago University Press.

Johnson, M. (1999). PCFG models of linguistic tree representations. *Computational Linguistics*, **24**(4):613–632.

Johnson, M. (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 136–143, Philadelphia, PA.

Kaplan, R. (1995). The Formal Architecture of Lexical-Functional Grammar. In Dalrymple, M., editor, *Formal Issues in Lexical-Functional Grammar*, pages 7–28, Stanford, CA. CSLI Publications.

Kaplan, R. and Bresnan, J. (1982). Lexical Functional Grammar, a Formal System for Grammatical Representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Kaplan, R., Riezler, S., King, T. H., Maxwell, J. T., Vasserman, A., and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference and the 4th Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, pages 97–104, Boston, MA.

Kaplan, R. and Wedekind, J. (2000). LFG Generation Produces Context-Free Languages. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 425–431, Saarbruecken, Germany.

Kaplan, R. and Zaenen, A. (1989). Long-distance Dependencies, Constituent Structure and Functional Uncertainty. In Baltin, M. and Kroch, A., editors, *Alternative Conceptions of Phrase Structure*, pages 17–42, Chicago. Chicago University Press. Reprinted in Dalrymple et al. (editors), *Formal Issues in Lexical-Functional Grammar.* CSLI Publications, 1995.

Kaplan, R. M., Netter, K., Wedekind, J., and Zaenen, A. (1989). Translation by structural correspondences. In *Proceedings of the 4th Meeting of the European Chapter of the Association for Computational Linguistics*, pages 272–281, University of Manchester, UK.

Kay, M. (1985). Parsing in Functional-Unification Grammar. In Dowty, D. R., Karttunen, L., and Zwicky, A. M., editors, *Natural Language Parsing*, pages 251–278. Cambridge University Press, Cambridge, UK.

Kay, M., Gawron, J. M., and Norvig, P. (1994). *Verbmobil. A Translation System for Face-to-Face Dialog*, volume **33**. CSLI Lecture Notes. Chicago University Press.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M., and Kaplan, R. (2003). The PARC700 dependency bank. In *Proceedings of the EACL03: 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, pages 1–8, Budapest, Hungary.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan.

Lappin, S., Golan, I., and Rimon, M. (1989). Computing Grammatical Functions from Configurational Parse Trees. Technical Report 88.268, IBM Israel, Haifa, Israel.

Macleod, C., Meyers, A., and Grishman, R. (1994). The COMLEX Syntax Project: The First Year. In *Proceedings of the ARPA Workshop on Human Language Technology*, pages 669–703, Princeton, NJ.

Magerman, D. (1994). *Natural Language Parsing as Statistical Pattern Recognition.* PhD thesis, Department of Computer Science, Stanford University, CA.

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

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument

Structure. In *Proceedings of the ARPA Workshop on Human Language Technology*, pages 110–115, Princton, NJ.

McCarthy, M. (2003). Design and Evaluation of the Linguistic Basis of an Automatic F-Structure Annotation Algorithm for the Penn-II Treebank. Master's thesis, School of Computing, Dublin City University, Dublin, Ireland.

Miyao, Y., Ninomiya, T., and Tsujii, J. (2003). Probabilistic modeling of argument structures including non-local dependencies. In *Proceedings of the Conference on Recent Advances in Natural Language Processing (RANLP)*, pages 285–291, Borovets, Bulgaria.

O'Donovan, R., Burke, M., Cahill, A., van Genabith, J., and Way, A. (2004). Large-Scale Induction and Evaluation of Lexical Resources from the Penn-II Treebank. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 368–375, Barcelona, Spain.

Pito, R. (1993). Documentation for `tgrep`. LDC, University of Pennsylvania, PA.

Pollard, C. and Sag, I. (1994). *Head-driven Phrase Structure Grammar*. CSLI Publications, Stanford, CA.

Ratnaparkhi, A. (1999). Learning to Parse Natural Language with Maximum Entropy Models. *Machine Learning*, **34**(1-3):151–176.

Riezler, S., King, T., Kaplan, R., Crouch, R., Maxwell, J. T., and Johnson, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*, pages 271–278, Philadelphia, PA.

Sadler, L., van Genabith, J., and Way, A. (2000). Automatic F-Structure Annotation from the AP Treebank. In Butt, M. and King, T. H., editors, *Proceedings of the Fifth International Conference on LFG*, pages 226–243, Stanford, CA. CSLI Publications.

Schmid, H. (2004). Efficient Parsing of Highly Ambiguous Context-Free Grammars with Bit Vectors. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2004)*, pages 162–168, Geneva, Switzerland.

Shieber, S. M. (1984). The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics*, pages 362–366, Stanford University, CA.

Tateisi, Y., Torisawa, K., Miyao, Y., and Tsujii, J. (1998). Translating XTAG english grammar to HPSG. In *Proceedings of Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 172–175, Philadelphia, PA.

van Genabith, J. and Crouch, R. (1996). Direct and Underspecified Interpretations of LFG f-Structures. In *16th International Conference on Computational Linguistics (COLING 96)*, pages 262–267, Copenhagen, Denmark.

van Genabith, J., Way, A., and Sadler, L. (1999). Data-driven Compilation of LFG Semantic Forms. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora (LINC-99)*, pages 69–76, Bergen, Norway.

Xia, F. (1999). Extracting Tree Adjoining Grammars from Bracketed Corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99)*, pages 398–403, Beijing, China.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information Control*, **10**(2):189–208.

Appendix A

Non-punctuation tags in the Penn-II Treebank

Tag Label	Tag Description
CC	Coordinating conjunction e.g. and,but,or...
CD	Cardinal Number
DT	Determiner
EX	Existential there
FW	Foreign Word
IN	Preposision or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List Item Marker
MD	Modal e.g. can, could, might, may...
NN	Noun, singular or mass
NNP	Proper Noun, singular
NNPS	Proper Noun, plural
NNS	Noun, plural
PDT	Predeterminer e.g. all, both ... when they precede an article
POS	Possessive Ending e.g. Nouns ending in 's

Tag Label	Tag Description
PRP	Personal Pronoun e.g. I, me, you, he...
PRP$	Possessive Pronoun e.g. my, your, mine, yours...
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection e.g. uh, well, yes, my...
VB	Verb, base form – subsumes imperatives, infinitives and subjunctives
VBD	Verb, past tense –includes the conditional form of the verb to be
VBG	Verb, gerund or persent participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner e.g. which, and that when it is used as a relative pronoun
WP	Wh-pronoun e.g. what, who, whom...
WP$	Possessive wh-pronoun e.g. whose
WRB	Wh-adverb e.g. how, where why

Appendix B

DCU 105 Gold Standard Sentences

1. The investment community , for one , has been anticipating a speedy resolution .
2. " The market has taken two views: that the labor situation will get settled in the short term and that things look very rosy for Boeing in the long term , " said Howard Rubel , an analyst at Cyrus J. Lawrence Inc .
3. " I would n't expect an immediate resolution to anything . "
4. In separate developments Talks have broken off between Machinists representatives at Lockheed Corp. and the Calabasas , Calif. , aerospace company .
5. United Auto Workers Local 1069 , which represents 3,000 workers at Boeing 's helicopter unit in Delaware County , Pa. , said it agreed to extend its contract on a day-by-day basis , with a 10-day notification to cancel , while it continues bargaining .
6. The planes , long range versions of the medium-haul twin-jet , will be delivered with Pratt & Whitney PW4060 engines .
7. Martinair Holland is based in Amsterdam .
8. The projects are big .

9. " Our long suit is our proven ability to operate " power plants , he said .

10. " This is a real thrust on our utility side , " he said , adding that Canadian Utilities is also mulling projects in underdeveloped countries , though he would be specific .

11. Mr. Stram said Enron is considering building gas-fired power plants in the U.K. capable of producing about 500 megawatts of power at a cost of about $ 300 million to $ 400 million .

12. PSE Inc. said it expects to report third earnings of $ 1.3 million to $ 1.7 million , or 14 cents to 18 cents a share .

13. The company said the improvement is related to additional cogeneration facilities that have been put into operation .

14. CONCORDE trans-Atlantic flights are $ 2,400 to Paris and $ 3,200 to London .

15. Diamond Shamrock Offshore 's stock rose 12.5 cents Friday to close at $ 8.25 in New York Stock Exchange composite trading .

16. Kaufman & Broad Home Corp. said it formed a $ 53.4 million limited partnership subsidiary to buy land in California suitable for residential development .

17. The land to be purchased by the joint venture has n't yet received zoning and other approvals required for development , and part of Kaufman & Broad 's job will be to obtain such approvals .

18. Typically , developers option property , and then once they get the administrative approvals , they buy it , " said Mr. Karatz , adding that he believes the joint venture is the first of its kind .

19. And if rain does n't fall soon across many of the Great Plains ' wheat-growing areas , yields in the crop now being planted could be reduced , further squeezing supplies .

20. That would be the lowest level since the early 1970s .

21. The government estimates that the new plan will boost production next year by about 66 million bushels .

22. On the Chicago Board of Trade Friday , wheat for December delivery settled at $ 4.0675 a bushel , unchanged .

23. In July , the CBOT ordered Ferruzzi Finanziaria S.p . A. to liquidate futures positions equal to about 23 million bushels of soybeans .

24. Unseasonably hot , dry weather across large portions of the Great Plains and in wheat-growing areas in Washington and Oregon is threatening to reduce the yield from this season 's winter wheat crop , said Conrad Leslie , a futures analyst and head of Leslie Analytical in Chicago .

25. That figure climbs to about 47 % in wheat-growing portions of Kansas , he said .

26. Looking ahead to other commodity markets this week

27. Late Thursday , after the close of trading , the market received what would normally have been a bullish U.S. Department of Agriculture estimate of the 1989-90 Florida orange crop .

28. It settled with a loss of 4.95 cents at $ 1.3210 a pound .

29. New York futures prices have dropped significantly from more than $ 2 a pound at midyear .

30. Barring a cold snap or other crop problems in the growing areas , downward pressure on prices is likely to continue into January , when harvesting and processing of oranges in Florida reach their peak , the analyst said .

31. On the New York Mercantile Exchange , West Texas Intermediate crude for November delivery finished at $ 20.89 a barrel , up 42 cents on the day .

32. There has been little news to account for such buoyancy in the oil markets .

33. Many traders foresee a tightening of near-term supplies , particularly of high-quality crudes such as those produced in the North Sea and in Nigeria .

34. If a hostile predator emerges for Saatchi & Saatchi Co. , co-founders Charles and Maurice Saatchi will lead a management buy-out attempt , an official close to the company said .

35. Last week , Saatchi 's largest shareholder , Southeastern Asset Management , said it had been approached by one or more third parties interested in a possible restructuring .

36. And Carl Spielvogel , chief executive officer of Saatchi 's big Backer Spielvogel Bates advertising unit , said he had offered to lead a management buy-out of the company , but was rebuffed by Charles Saatchi .

37. The executive said any buy-out would be led by the current board , whose chairman is Maurice Saatchi and whose strategic guiding force is believed to be Charles Saatchi .

38. The executive denied speculation that Saatchi was bringing in the new chief executive officer only to clean up the company financially so that the brothers could lead a buy-back .

39. Asked about the speculation that Mr. Louis-Dreyfus has been hired to pave the way for a buy-out by the brothers , the executive replied , That is n't the reason Dreyfus has been brought in .

40. It has n't had any impact on us , nor do we expect it to , " said a spokeswoman for Miller Brewing Co. , a major client of Backer Spielvogel .

41. Executives at Backer Spielvogel client Avis Inc. , as well as at Saatchi client Philips Lighting Co. , also said they saw no effect .

42. NEW ACCOUNT

43. ACCOUNT REVIEW

44. As expected , Young & Rubicam Inc. along with two senior executives and a former employee , pleaded not guilty in federal court in New Haven , Conn. , to conspiracy and racketeering charges .

45. KOREAN AGENCY

46. Samsung already owns Korea First Advertising Co. , that country 's largest agency .

47. Revenue soared to $ 117 million from $ 81.5 million .

48. Operationally , Maxtor benefited from robust sales of products that store data for high-end personal computers and computer workstations .

49. He retired as senior vice president , finance and administration , and chief financial officer of the company Oct. 1 .

50. Southmark Corp. said that it filed part of its 10-K report with the Securities and Exchange Commission , but that the filing does n't include its audited financial statements and related information .

51. Southmark said it plans to amend its 10K to provide financial results as soon as its audit is completed .

52. Alan Seelenfreund , 52 years old , was named chairman of this processor of prescription claims , succeeding Thomas W. Field Jr. , 55 , who resigned last month .

53. Messrs. Malson and Seelenfreund are directors of McKesson , which has an 86 % stake in PCS .

54. MedChem Products Inc. said a U.S. District Court in Boston ruled that a challenge by MedChem to the validity of a U.S. patent held by Pharmacia Inc. was without merit . "

55. The patent is related to hyaluronic acid , a rooster-comb extract used in eye surgery .

56. MedChem said the court 's ruling was issued as part of a first-phase trial " in the patent-infringement proceedings and concerns only one of its defenses in the case .

57. MedChem said that the court scheduled a conference for next Monday to set a date for proceedings on Pharmacia 's motion for a preliminary injunction .

58. Newspaper publishers are reporting mixed third-quarter results , aided by favorable newsprint prices and hampered by flat or declining advertising linage , especially in the Northeast .

59. Many papers throughout the country are also faced with a slowdown in classified-ad spending , a booming category for newspapers in recent years .

60. Improved paper prices will help offset weakness in linage , but the retailers ' problems have affected the amount of ad linage they usually run , " said Edward J. Atorino , industry analyst for Salomon Brothers Inc .

61. For instance , Gannett Co. posted an 11 % gain in net income , as total ad pages dropped at USA Today , but advertising revenue rose because of a higher circulation rate base and increased rates .

62. Gannett 's 83 daily and 35 non-daily newspapers reported a 3 % increase in advertising and circulation revenue .

63. At Dow Jones & Co. , third-quarter net income fell 9.9 % from the year-earlier period .

64. Revenue gained 5.3 % to $ 404.1 million from $ 383.8 million .

65. Ad linage at the Journal fell 6.1 % in the third quarter .

66. William O. Taylor , the parent 's chairman and chief executive officer , said earnings continued to be hurt by softness in ad volume at the Boston newspaper .

67. After a supply crunch caused prices to rise 14 % since 1986 to $ 650 a metric ton , analysts are encouraged , because they do n't expect a price increase for the rest of this year .

68. Times Co. 's regional daily newspapers are holding up well , but there is little sign that things will improve in the New York market , " said Alan Kassan , an analyst with Shearson Lehman Hutton .

69. According to analysts , profits were also helped by successful cost-cutting measures at Newsweek .

70. However , analysts point to positive advertising spending at several of its major daily newspapers , such as the Miami Herald and San Jose Mercury News .

71. GM is under intense pressure to close factories that became unprofitable as the giant auto maker 's U.S. market share skidded during the past decade .

72. Now , GM appears to be stepping up the pace of its factory consolidation to get in shape for the 1990s .

73. Against that backdrop , UAW Vice President Stephen P. Yokich , who recently became head of the union 's GM department , issued a statement Friday blasting GM 's flagrant insensitivity " toward union members .

74. That means two plants one in Scarborough , Ontario , and the other in Lordstown , Ohio probably will be shut down after the end of 1991 .

75. But Canadian auto workers may benefit from a separate GM move that affects three U.S. car plants and one in Quebec .

76. That announcement left union officials in Van Nuys and Oklahoma City uncertain about their futures .

77. He said he believes GM has plans to keep building A-body cars into the mid-1990s .

78. Union officials have taken a beating politically as a result .

79. The French company said the government gave it 30 days in which to submit information to further support its takeover plan .

80. Alan Nymark , executive vice president of Investment Canada , which oversees foreign takeovers , said it marked the first time in its four-year history that the agency has made an adverse net-benefit decision about the acquisition of a publicly traded company .

81. Mr. Andre issued the ruling based on a recommendation by Investment Canada .

82. This has become a very politicized deal , concerning Canada 's only large , world-class bio-research or pharmaceutical company , " Mr. Mehta said .

83. The university is considering a settlement proposal made by Connaught .

84. Officials for the two concerns , which are bidding C$ 30 a share for Connaught , could n't be reached for comment .

85. Weatherford said market conditions led to the cancellation of the planned exchange .

86. Weatherford currently has approximately 11.1 million common shares outstanding .

87. Earnings for most of the nation 's major pharmaceutical makers are believed to have moved ahead briskly in the third quarter , as companies with newer , big-selling prescription drugs fared especially well .

88. Less robust earnings at Pfizer Inc. and Upjohn Co. were attributed to those companies ' older products , many of which face stiffening competition from generic drugs and other medicines .

89. In New York , the company declined comment .

90. Sales of both drugs have been hurt by new state laws restricting the prescriptions of certain tranquilizing medicines and adverse publicity about the excessive use of the drugs .

91. Revenue is expected to be up modestly " from the $ 26.5 million reported a year ago .

92. Sharp-witted and funny but never mean , she 's a memorialist a bit like Truman Capote , if he 'd been Jewish and female and less bitchy .

93. Rosie died young and Lily has remembered her as a romantic figure , who did n't interfere much with her child 's education on the streets .

94. She analyzed families by their sleeping arrangements .

95. Maybe Lily became so obsessed with where people slept and how because her own arrangements kept shifting .

96. They came by their strangeness honestly .

97. For the most part , though , there 's much pleasure in her saucy , poignant probe into the mysteries of the Babylonian Bronx .

98. For his sixth novel , Mr. Friedman tried to resuscitate the protagonist of his 1972 work , About Harry Towns . "

99. Harry has avoided all that by living in a Long Island suburb with his wife , who 's so addicted to soap operas and mystery novels she barely seems to notice when her husband disappears for drug-seeking forays into Manhattan .

100. In 1984 the EPA notified Gulf Resources , which was a part-owner of the smelter , that it was potentially liable for sharing cleanup costs at the site under the federal Superfund program .

101. The company said that as part of its agreement with the EPA , it made certain voluntary undertakings with respect to intercorporate transactions entered into after the reorganization . "

102. Under the agreement , Gulf must give the U.S. government 45 days ' advance written notice before issuing any dividends on common stock .

103. Assuming that the market does n't head into a bottomless free fall , some executives think Friday 's action could prove a harbinger of good news as a sign that the leveraged buy-out and takeover frenzy of recent years may be abating .

104. Where 's the guy who can say Enough is enough ' " ?

105. There has n't been any fundamental change in the economy , " added John Smale , whose Procter & Gamble Co. took an $ 8.75 slide to close at $ 120.75 .

Appendix C

Parsing Results for Section 23 Trees

Group 1	Group 2
a Add root node **b** No root node	**c** No unary productions **d** No X → X productions **e** Include all unary productions **f** No unary productions, but keep information
Group 3	Group 4
g Add f-structure annotation **h** No f-structure annotation	**j** Add parent **k** Add grandparent **m** No parent/grandparent transformation
Group 5	Group 6
n Keep Penn-II functional labels **o** Remove Penn-II functional labels	**p** No AUX change **q** Change only true auxiliary verbs **r** Change all auxiliary verb labels

The six groups of transformations used to test transformation interaction. A grammar with one feature from each group is extracted. This gives 288 grammars.

There were a small number of grammars that caused `BitPar` to produce corrupt data, and therefore we are unable to provide the results for those 21 experiments here.

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
acgjnp	82729	2305	78.75	80.79	45.90
acgjnq	82615	2298	78.75	80.73	45.91
acgjnr	83014	2296	78.64	80.63	45.56
acgjop	65405	2334	78.65	80.70	46.14

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
acgjoq	65290	2328	78.66	80.67	46.09
acgjor	65887	2328	78.76	80.73	46.05
acgknp	148261				
acgknq	148028				
acgknr	148488				
acgkop	116037	2134	77.01	79.05	45.17
acgkoq	115821	2113	76.96	78.97	45.34
acgkor	116597	2112	77.09	79.20	45.60
acgmnp	44055	2399	79.51	81.59	47.44
acgmnq	43999	2399	79.56	81.61	47.52
acgmnr	44267	2399	79.69	81.67	47.39
acgmop	35591	2399	79.56	81.56	46.52
acgmoq	35541	2399	79.78	81.79	46.85
acgmor	35896	2399	79.84	81.82	46.85
achjnp	48630	2391	78.72	80.82	45.38
achjnq	48656	2390	78.75	80.82	45.40
achjnr	48912	2387	78.91	80.93	45.08
achjop	29419	2396	78.89	80.97	44.91
achjoq	29448	2396	78.96	81.03	44.74
achjor	29797	2394	78.93	80.99	44.40
achknp	77410	2351	77.90	80.06	45.13
achknq	77414	2347	77.97	80.09	44.65
achknr	77757	2347	77.60	79.69	44.65
achkop	45677	2378	78.26	80.37	43.48
achkoq	45694	2378	78.22	80.35	43.44
achkor	46206	2377	78.48	80.57	43.42
achmnp	29613	2410	75.95	78.49	40.54
achmnq	29641	2410	76.04	78.56	40.54
achmnr	29792	2409	76.01	78.49	40.85
achmop	18968	2410	75.21	77.70	38.76
achmoq	19000	2410	75.31	77.82	39.09
achmor	19204	2410	75.46	77.90	39.71

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
adgjnp	68232	2412	80.58	82.47	46.48
adgjnq	68130	2412	80.74	82.62	46.93
adgjnr	68361	2412	80.94	82.76	46.43
adgjop	50527	2413	81.10	82.95	46.79
adgjoq	50459	2413	81.27	83.10	47.20
adgjor	50835	2413	81.25	83.13	47.12
adgknp	124442	2386	79.00	81.07	44.68
adgknq	124245	2385	79.11	81.17	44.78
adgknr	124536	2383	79.14	81.26	44.82
adgkop	91395	2398	79.54	81.48	45.08
adgkoq	91259	2396	79.57	81.50	45.28
adgkor	91784	2397	79.60	81.58	45.26
adgmnp	36816	2416	77.32	79.38	41.89
adgmnq	36756	2416	77.42	79.48	41.93
adgmnr	36940	2416	77.48	79.52	42.14
adgmop	27545	2416	77.17	79.30	41.51
adgmoq	27502	2416	77.26	79.38	41.76
adgmor	27753	2416	77.33	79.45	41.89
adhjnp	40820	2416	80.39	82.21	44.54
adhjnq	40770	2416	80.46	82.31	44.87
adhjnr	40933	2416	80.48	82.30	44.74
adhjop	21638	2416	79.29	81.30	42.67
adhjoq	21602	2416	79.41	81.40	42.88
adhjor	21811	2416	79.33	81.28	42.88
adhknp	66460	2416	80.23	82.14	44.74
adhknq	66366	2416	80.33	82.21	44.66
adhknr	66561	2416	80.34	82.25	44.62
adhkop	34345	2416	80.04	81.85	43.00
adhkoq	34296	2416	80.20	82.01	43.50
adhkor	34606	2416	80.13	81.96	43.38
adhmnp	25480	2416	72.05	74.14	32.78
adhmnq	25445	2416	72.07	74.15	32.66

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
adhmnr	25559	2416	72.18	74.25	32.78
adhmop	14327	2416	70.82	73.08	30.01
adhmoq	14303	2416	71.09	73.33	30.13
adhmor	14456	2416	71.06	73.33	30.13
aegjnp	68046	2413	80.52	82.42	46.33
aegjnq	67943	2413	80.66	82.55	46.75
aegjnr	68172	2413	80.91	82.74	46.46
aegjop	50072	2414	80.81	82.69	46.64
aegjoq	50003	2414	81.00	82.85	47.06
aegjor	50375	2414	81.09	83.00	47.10
aegknp	124106	2388	79.07	81.12	44.89
aegknq	123907	2387	79.21	81.24	44.95
aegknr	124198	2387	79.27	81.37	45.25
aegkop	90621	2397	79.69	81.60	45.14
aegkoq	90483	2398	79.62	81.53	45.45
aegkor	91008	2397	79.62	81.57	45.35
aegmnp	36741	2416	77.32	79.39	41.93
aegmnq	36682	2415	77.43	79.51	42.03
aegmnr	36865	2416	77.48	79.52	42.14
aegmop	27332	2414	77.11	79.24	41.67
aegmoq	27291	2414	77.18	79.31	41.92
aegmor	27542	2416	77.25	79.39	41.97
aehjnp	40843	2416	80.38	82.21	44.58
aehjnq	40793	2416	80.46	82.30	44.95
aehjnr	40957	2416	80.45	82.27	44.74
aehjop	21656	2416	79.18	81.19	42.67
aehjoq	21620	2416	79.31	81.30	42.88
aehjor	21829	2416	79.25	81.20	42.84
aehknp	66530	2416	80.23	82.14	44.95
aehknq	66436	2416	80.34	82.23	44.91
aehknr	66631	2416	80.38	82.29	44.99
aehkop	34401	2416	80.02	81.82	43.17

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
aehkoq	34352	2416	80.18	82.00	43.67
aehkor	34662	2416	80.10	81.92	43.63
aehmnp	25489	2416	72.00	74.08	32.74
aehmnq	25454	2416	72.03	74.11	32.62
aehmnr	25568	2416	72.15	74.22	32.78
aehmop	14335	2416	70.74	73.00	30.13
aehmoq	14311	2416	71.00	73.25	30.26
aehmor	14464	2416	70.94	73.21	30.05
afgjnp	91229	2294	77.18	79.03	45.29
afgjnq	91014	2285	77.27	79.09	45.25
afgjnr	91331	2285	77.16	79.00	45.43
afgjop	70502	2329	77.17	79.00	45.86
afgjoq	70308	2322	77.14	78.92	45.82
afgjor	70915	2321	77.35	79.13	46.14
afgknp	161852				
afgknq	161553				
afgknr	161934				
afgkop	124505	2115	75.35	77.19	44.96
afgkoq	124225	2092	75.45	77.24	45.17
afgkor	125010	2086	75.48	77.38	45.40
afgmnp	48620	2399	77.95	79.76	47.60
afgmnq	48499	2399	78.05	79.88	47.73
afgmnr	48744	2399	78.13	79.88	47.64
afgmop	37993	2399	77.86	79.61	46.85
afgmoq	37903	2399	78.13	79.88	47.06
afgmor	38270	2399	78.23	79.97	47.19
afhjnp	62062	2381	77.45	79.16	45.02
afhjnq	61945	2378	77.50	79.21	44.95
afhjnr	62183	2375	77.61	79.30	45.01
afhjop	36459	2393	77.55	79.28	45.22
afhjoq	36371	2392	77.59	79.32	45.11
afhjor	36812	2391	77.77	79.44	45.42

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
afhknp	104841	2316	76.35	78.12	44.56
afhknq	104676	2307	76.48	78.23	44.43
afhknr	104995	2310	76.29	78.05	44.55
afhkop	59467	2373	76.94	78.70	44.46
afhkoq	59344	2367	77.00	78.73	44.40
afhkor	59946	2371	77.05	78.79	44.16
afhmnp	35971	2412	75.01	76.98	41.00
afhmnq	35904	2412	75.20	77.16	41.29
afhmnr	36063	2411	75.46	77.44	41.97
afhmop	22324	2412	75.07	76.92	39.22
afhmoq	22272	2412	75.21	77.06	39.59
afhmor	22534	2412	75.43	77.29	40.09
bcgjnp	84171	2297	78.37	80.67	45.93
bcgjnq	84041	2289	78.31	80.56	45.87
bcgjnr	84435	2288	78.15	80.40	45.41
bcgjop	65868	2328	78.35	80.64	46.22
bcgjoq	65752	2321	78.41	80.64	46.14
bcgjor	66346	2323	78.47	80.68	46.02
bcgknp	151956				
bcgknq	151698				
bcgknr	152142				
bcgkop	117489	2119	76.55	78.90	44.93
bcgkoq	117267	2091	76.53	78.84	45.05
bcgkor	118033	2089	76.69	79.08	45.43
bcgmnp	44084	2405	78.78	81.11	46.65
bcgmnq	44028	2405	78.80	81.11	46.78
bcgmnr	44296	2405	78.93	81.18	46.57
bcgmop	35485	2405	78.55	80.85	45.90
bcgmoq	35435	2405	78.79	81.09	46.32
bcgmor	35790	2405	78.99	81.25	46.32
bchjnp	47691	2393	78.73	81.07	46.13
bchjnq	47716	2393	78.77	81.10	46.09

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
bchjnr	47956	2391	78.86	81.10	45.55
bchjop	28021	2399	78.65	80.88	44.56
bchjoq	28045	2399	78.69	80.93	44.56
bchjor	28355	2399	78.72	80.90	44.23
bchknp	77197	2352	77.44	79.91	44.94
bchknq	77201	2351	77.47	79.88	44.75
bchknr	77499	2348	77.35	79.77	44.63
bchkop	44453	2377	78.32	80.67	43.88
bchkoq	44464	2375	78.29	80.65	43.92
bchkor	44943	2374	78.49	80.83	44.23
bchmnp	29627	2410	72.59	75.07	36.22
bchmnq	29655	2410	72.57	75.05	36.14
bchmnr	29806	2410	72.58	75.05	36.51
bchmop	18850	2410	71.33	73.94	33.82
bchmoq	18883	2410	71.46	74.07	34.15
bchmor	19087	2410	71.75	74.30	34.81
bdgjnp	70395	2409	79.52	81.48	46.82
bdgjnq	70271	2409	79.62	81.58	47.32
bdgjnr	70487	2409	79.67	81.59	46.74
bdgjop	51882	2411	79.92	81.87	46.45
bdgjoq	51811	2411	80.08	82.03	46.87
bdgjor	52191	2411	80.09	82.08	46.87
bdgknp	128652				
bdgknq	128427				
bdgknr	128742				
bdgkop	93687	2394	78.21	80.31	44.90
bdgkoq	93536	2392	78.09	80.18	45.03
bdgkor	94088	2393	78.20	80.35	44.88
bdgmnp	36780	2416	76.07	78.25	41.89
bdgmnq	36720	2416	76.17	78.35	41.93
bdgmnr	36904	2416	76.24	78.39	42.14
bdgmop	27508	2416	75.92	78.16	41.51

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
bdgmoq	27465	2416	76.01	78.24	41.76
bdgmor	27716	2416	76.08	78.32	41.89
bdhjnp	41371	2416	79.28	81.23	44.45
bdhjnq	41311	2416	79.37	81.34	44.74
bdhjnr	41475	2416	79.36	81.30	44.62
bdhjop	21601	2416	78.15	80.27	42.67
bdhjoq	21565	2416	78.27	80.37	42.88
bdhjor	21774	2416	78.19	80.25	42.88
bdhknp	67729	2416	78.89	81.05	44.54
bdhknq	67619	2416	79.03	81.16	44.54
bdhknr	67812	2416	79.10	81.28	44.54
bdhkop	34308	2416	78.95	80.85	43.00
bdhkoq	34259	2416	79.11	81.02	43.50
bdhkor	34569	2416	79.04	80.97	43.38
bdhmnp	25444	2416	70.49	72.69	32.78
bdhmnq	25409	2416	70.51	72.71	32.66
bdhmnr	25523	2416	70.63	72.81	32.78
bdhmop	14290	2416	69.17	71.56	30.01
bdhmoq	14266	2416	69.46	71.83	30.13
bdhmor	14419	2416	69.43	71.83	30.13
begjnp	70207	2409	79.52	81.48	46.62
begjnq	70082	2409	79.63	81.59	47.16
begjnr	70297	2409	79.67	81.61	46.82
begjop	51410	2411	79.83	81.82	46.66
begjoq	51338	2409	79.98	81.94	47.07
begjor	51717	2411	80.02	82.01	47.12
begknp	128307				
begknq	128082				
begknr	128398				
begkop	92887	2394	78.39	80.45	44.82
begkoq	92736	2394	78.21	80.26	44.90
begkor	93288	2394	78.31	80.38	44.95

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
begmnp	36704	2416	76.07	78.26	41.93
begmnq	36645	2416	76.18	78.37	42.01
begmnr	36828	2416	76.24	78.40	42.14
begmop	27295	2416	75.85	78.10	41.68
begmoq	27254	2416	75.92	78.16	41.93
begmor	27505	2416	75.99	78.25	41.97
behjnp	41394	2416	79.28	81.22	44.50
behjnq	41334	2416	79.36	81.33	44.78
behjnr	41499	2416	79.34	81.27	44.58
behjop	21619	2416	78.04	80.15	42.67
behjoq	21583	2416	78.17	80.27	42.88
behjor	21792	2416	78.10	80.16	42.84
behknp	67799	2416	78.88	81.04	44.74
behknq	67689	2416	79.04	81.16	44.66
behknr	67882	2416	79.14	81.30	44.78
behkop	34364	2416	78.92	80.83	43.17
behkoq	34315	2416	79.10	81.02	43.67
behkor	34625	2416	79.01	80.93	43.63
behmnp	25452	2416	70.44	72.63	32.74
behmnq	25417	2416	70.47	72.67	32.62
behmnr	25531	2416	70.60	72.78	32.78
behmop	14298	2416	69.09	71.48	30.13
behmoq	14274	2416	69.37	71.74	30.26
behmor	14427	2416	69.31	71.70	30.05
bfgjnp	92641	2286	78.93	81.01	45.14
bfgjnq	92408	2277	78.97	81.02	45.10
bfgjnr	92728	2276	78.89	80.96	45.25
bfgjop	70937	2323	79.06	81.11	45.93
bfgjoq	70740	2317	79.01	81.01	45.88
bfgjor	71347	2317	79.23	81.24	46.22
bfgknp	165490				
bfgknq	165165				

Grammar	#Rules	#Parses	Labelled F-Score	Unlabelled F-Score	Accuracy
bfgknr	165540				
bfgkop	125894				
bfgkoq	125605				
bfgkor	126385				
bfgmnp	48651	2407	79.35	81.44	46.78
bfgmnq	48530	2407	79.52	81.61	46.95
bfgmnr	48775	2407	79.62	81.64	46.82
bfgmop	37887	2407	79.25	81.28	46.12
bfgmoq	37797	2407	79.50	81.55	46.45
bfgmor	38164	2407	79.62	81.63	46.61
bfhjnp	61153	2383	79.53	81.50	45.32
bfhjnq	61039	2383	79.62	81.58	45.20
bfhjnr	61303	2380	79.80	81.78	45.38
bfhjop	34979	2395	79.57	81.45	45.18
bfhjoq	34902	2395	79.68	81.58	45.39
bfhjor	35297	2394	79.77	81.64	45.36
bfhknp	104871	2321	78.04	80.17	44.03
bfhknq	104697	2314	78.14	80.20	43.86
bfhknr	104998	2314	78.16	80.19	44.38
bfhkop	58087	2372	78.99	81.01	44.01
bfhkoq	57969	2366	78.93	80.94	44.00
bfhkor	58542	2370	79.10	81.09	44.09
bfhmnp	35979	2412	74.65	76.78	38.31
bfhmnq	35912	2411	74.79	76.90	38.45
bfhmnr	36072	2412	75.04	77.14	39.10
bfhmop	22198	2412	74.43	76.59	36.40
bfhmoq	22147	2412	74.52	76.67	36.65
bfhmor	22409	2412	74.68	76.84	36.65

Appendix D

Parsing Results for F-Structures against the DCU 105 and PARC 700

Group 1	Group 2
a Add root node **b** No root node	**c** No unary productions **d** No X → X productions **e** Include all unary productions **f** No unary productions, but keep information
Group 3	Group 4
g Add f-structure annotation **h** No f-structure annotation	**j** Add parent **k** Add grandparent **m** No parent/grandparent transformation
Group 5	Group 6
n Keep Penn-II functional labels **o** Remove Penn-II functional labels	**p** No AUX change **q** Change only true auxiliary verbs **r** Change all auxiliary verb labels

The six groups of transformations used to test transformation interaction. A grammar with one feature from each group is extracted. This gives 288 grammars.

There were a small number of grammars that caused `BitPar` to produce corrupt data, and therefore we are unable to provide the results for those 21 experiments here.

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
acgjnp	72.10	78.58	78.26	84.15	74.56
acgjnq	70.91	76.44	76.16	81.18	71.78
acgjnr	71.93	77.59	76.49	81.42	71.95
acgjop	72.59	78.75	79.39	84.95	75.37
acgjoq	71.87	77.94	77.66	83.17	72.34
acgjor	72.37	78.74	77.52	83.11	72.43
acgknp					
acgknq					
acgknr					
acgkop	74.00	77.37	53.62	67.05	71.59
acgkoq	72.74	76.05	53.35	66.69	68.43
acgkor	71.71	74.90	52.53	66.06	68.78
acgmnp	73.78	79.78	80.57	86.17	77.37
acgmnq	73.85	79.84	79.56	85.10	74.39
acgmnr	73.95	79.96	78.97	84.33	74.47
acgmop	73.52	79.82	80.36	86.18	77.53
acgmoq	73.95	80.27	79.58	85.36	74.45
acgmor	73.87	80.03	78.91	84.37	74.24
achjnp	69.12	74.45	77.14	82.33	73.94
achjnq	69.11	74.48	75.89	81.10	71.34
achjnr	69.33	74.73	75.28	80.31	71.21
achjop	67.83	73.93	76.40	82.24	73.81
achjoq	67.68	73.64	75.20	80.89	71.25
achjor	67.39	73.38	74.41	80.00	70.89
achknp	67.37	73.19	75.72	81.19	72.73
achknq	67.35	73.05	74.68	80.03	70.25
achknr	67.38	73.29	73.97	79.31	69.98
achkop	67.38	73.83	75.99	81.81	73.07
achkoq	67.39	73.67	74.70	80.38	70.67
achkor	67.42	73.73	74.10	79.55	70.29
achmnp	66.15	71.61	75.38	80.88	72.48

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
achmnq	65.13	70.66	73.48	78.98	69.96
achmnr	64.88	70.53	72.61	78.10	69.71
achmop	64.94	69.43	74.93	79.32	72.28
achmoq	64.94	69.58	73.72	78.23	69.68
achmor	65.38	69.97	73.31	77.62	69.28
adgjnp	71.24	76.71	77.81	82.87	77.81
adgjnq	71.60	77.24	77.04	82.12	74.45
adgjnr	72.65	78.53	77.29	82.29	74.38
adgjop	72.87	78.47	79.42	84.51	77.84
adgjoq	72.58	78.26	78.19	83.26	74.44
adgjor	72.35	78.13	77.35	82.31	74.20
adgknp	71.69	75.22	79.17	82.36	77.54
adgknq	71.38	74.79	77.92	81.09	74.74
adgknr	71.44	75.65	77.17	80.86	74.43
adgkop	72.83	77.91	80.22	84.88	77.61
adgkoq	72.63	77.58	79.08	83.66	74.55
adgkor	71.92	76.93	77.49	82.03	74.12
adgmnp	72.36	78.17	79.66	84.99	77.27
adgmnq	72.27	78.17	78.35	83.83	74.22
adgmnr	72.36	78.48	77.81	83.25	74.25
adgmop	71.88	77.76	79.25	84.58	77.12
adgmoq	72.17	78.35	78.39	83.95	74.16
adgmor	72.48	78.57	77.80	83.18	73.87
adhjnp	73.68	79.10	81.17	86.30	80.24
adhjnq	73.76	79.13	80.09	85.13	77.02
adhjnr	73.79	79.40	79.31	84.35	76.79
adhjop	70.91	76.79	79.89	85.65	79.25
adhjoq	71.32	77.22	79.12	84.80	76.17
adhjor	71.16	77.27	78.36	84.04	75.81
adhknp	72.79	78.33	80.94	86.18	80.26
adhknq	73.12	78.60	80.20	85.29	77.03

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
adhknr	73.55	79.19	79.72	84.78	76.69
adhkop	72.27	77.97	80.50	85.70	79.71
adhkoq	72.37	77.95	79.58	84.67	76.46
adhkor	72.24	77.75	78.90	83.59	76.04
adhmnp	70.30	74.17	79.80	83.77	77.86
adhmnq	69.36	74.07	78.02	82.69	74.54
adhmnr	69.47	74.08	77.22	81.45	74.27
adhmop	66.63	69.89	77.57	80.87	76.19
adhmoq	65.84	69.21	75.90	79.39	73.32
adhmor	66.31	68.98	75.63	78.12	73.04
aegjnp	72.56	78.95	79.09	84.96	78.43
aegjnq	72.92	79.48	78.31	84.19	75.05
aegjnr	74.04	80.80	78.61	84.40	74.87
aegjop	74.80	81.24	81.20	87.04	78.60
aegjoq	74.41	80.94	79.87	85.70	75.17
aegjor	74.21	80.83	79.07	84.80	74.99
aegknp	71.95	75.46	79.53	82.69	77.71
aegknq	71.65	75.03	78.25	81.39	74.88
aegknr	71.70	75.92	77.51	81.18	74.66
aegkop	73.18	78.26	80.47	85.13	77.91
aegkoq	73.15	78.08	79.53	84.09	74.89
aegkor	72.93	77.95	78.56	83.09	74.38
aegmnp	72.39	78.20	79.67	85.00	77.58
aegmnq	72.30	78.19	78.36	83.84	74.55
aegmnr	72.38	78.51	77.82	83.26	74.40
aegmop	71.92	77.83	79.27	84.61	77.80
aegmoq	72.21	78.42	78.41	83.98	74.78
aegmor	72.48	78.60	77.80	83.20	74.32
aehjnp	73.64	79.06	81.16	86.28	80.24
aehjnq	73.72	79.09	80.07	85.11	77.01
aehjnr	73.79	79.40	79.31	84.35	76.79

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
aehjop	70.91	76.79	79.89	85.65	79.28
aehjoq	71.32	77.22	79.12	84.80	76.20
aehjor	71.16	77.27	78.36	84.04	75.84
aehknp	72.88	78.41	80.98	86.22	80.33
aehknq	73.21	78.68	80.24	85.33	77.07
aehknr	73.76	79.45	79.79	84.92	76.73
aehkop	72.20	77.87	80.46	85.65	79.77
aehkoq	72.29	77.85	79.54	84.62	76.54
aehkor	72.24	77.75	78.90	83.59	76.07
aehmnp	70.30	74.17	79.80	83.77	77.86
aehmnq	69.36	74.07	78.02	82.69	74.54
aehmnr	69.47	74.08	77.22	81.45	74.27
aehmop	66.60	69.84	77.56	80.84	76.19
aehmoq	65.81	69.16	75.89	79.36	73.32
aehmor	66.20	68.87	75.57	78.07	72.99
afgjnp	70.02	76.59	76.87	82.78	75.14
afgjnq	69.33	74.99	75.26	80.35	72.06
afgjnr	70.10	75.88	75.34	80.32	71.83
afgjop	71.09	77.48	78.49	84.20	75.84
afgjoq	70.34	76.50	76.80	82.36	72.69
afgjor	70.71	77.17	76.31	81.94	72.39
afgknp					
afgknq					
afgknr					
afgkop	73.91	77.18	60.64	66.16	71.91
afgkoq	72.71	75.89	60.57	65.88	68.74
afgkor	70.82	73.89	60.03	64.46	68.36
afgmnp	73.17	79.39	79.99	85.71	78.30
afgmnq	73.32	79.51	79.05	84.68	74.84
afgmnr	73.43	79.49	78.49	83.88	74.73
afgmop	73.81	80.14	80.37	86.22	78.38

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
afgmoq	74.25	80.60	79.57	85.36	74.98
afgmor	74.00	80.16	78.84	84.28	74.59
afhjnp	73.98	79.55	81.46	86.56	79.78
afhjnq	73.38	78.84	79.83	84.92	76.53
afhjnr	74.10	79.65	79.74	84.68	76.32
afhjop	72.31	78.67	80.68	86.41	79.60
afhjoq	72.13	78.32	79.54	85.26	76.31
afhjor	71.90	78.08	78.67	84.24	76.04
afhknp	70.30	76.12	77.96	83.28	78.47
afhknq	70.30	75.98	76.79	81.95	75.08
afhknr	71.28	77.24	76.89	81.95	75.04
afhkop	71.31	77.78	79.40	85.35	78.80
afhkoq	70.99	77.37	77.91	83.82	75.59
afhkor	71.53	78.28	77.74	83.63	75.43
afhmnp	71.10	76.78	79.59	85.13	78.73
afhmnq	71.18	76.87	78.57	84.06	75.70
afhmnr	71.45	77.13	78.18	83.47	75.38
afhmop	69.87	74.93	78.99	83.80	78.23
afhmoq	69.97	75.15	77.98	82.99	75.38
afhmor	70.40	76.51	77.44	83.12	74.93
bcgjnp	72.41	78.84	78.78	84.52	73.98
bcgjnq	70.90	76.96	76.12	81.71	71.20
bcgjnr	71.80	77.98	76.39	81.90	71.48
bcgjop	72.55	78.72	79.37	84.93	74.98
bcgjoq	71.87	77.94	77.66	83.17	71.91
bcgjor	72.33	78.71	77.50	83.10	72.05
bcgknp					
bcgknq					
bcgknr					
bcgkop	73.51	76.75	46.98	66.57	70.76
bcgkoq	72.28	75.49	46.70	66.22	67.71

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
bcgkor	71.27	74.37	45.86	65.58	68.05
bcgmnp	73.78	79.78	80.57	86.17	77.36
bcgmnq	73.85	79.84	79.56	85.10	74.36
bcgmnr	73.91	79.92	78.95	84.31	74.44
bcgmop	73.84	80.14	80.54	86.38	77.47
bcgmoq	74.16	80.47	79.70	85.48	74.42
bcgmor	73.89	80.05	78.93	84.40	74.26
bchjnp	69.89	75.41	77.60	82.88	73.86
bchjnq	70.03	75.73	76.44	81.82	71.30
bchjnr	69.85	75.43	75.51	80.64	71.16
bchjop	68.00	74.48	76.60	82.72	73.78
bchjoq	68.01	74.50	75.38	81.46	71.14
bchjor	67.84	74.34	74.62	80.58	70.87
bchknp	66.80	72.72	75.42	80.93	72.79
bchknq	67.64	73.38	75.17	80.54	70.42
bchknr	67.88	73.75	74.61	79.95	70.07
bchkop	67.97	74.60	76.28	82.20	72.85
bchkoq	68.30	74.74	75.70	81.45	70.50
bchkor	68.57	75.30	75.21	80.98	70.11
bchmnp	66.31	71.46	75.68	81.01	72.22
bchmnq	65.66	70.99	74.15	79.59	69.75
bchmnr	65.87	71.34	73.49	78.87	69.56
bchmop	64.33	68.27	74.62	78.58	71.84
bchmoq	64.24	68.33	73.35	77.42	69.19
bchmor	64.58	68.78	72.94	76.90	68.94
bdgjnp	71.09	76.48	77.79	82.88	77.62
bdgjnq	71.69	77.24	77.20	82.29	74.23
bdgjnr	72.11	77.88	76.96	81.98	74.20
bdgjop	72.88	78.43	79.48	84.61	77.77
bdgjoq	72.84	78.46	78.46	83.56	74.45
bdgjor	72.37	78.13	77.40	82.41	74.02

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
bdgknp					
bdgknq					
bdgknr					
bdgkop	72.10	77.04	79.23	83.78	77.35
bdgkoq	71.90	76.55	78.12	82.45	74.33
bdgkor	70.95	75.75	76.52	80.87	74.20
bdgmnp	72.36	78.17	79.66	84.99	77.32
bdgmnq	72.27	78.17	78.35	83.83	74.28
bdgmnr	72.36	78.48	77.81	83.25	74.30
bdgmop	71.88	77.76	79.25	84.58	77.15
bdgmoq	72.17	78.35	78.39	83.95	74.19
bdgmor	72.48	78.57	77.80	83.18	73.90
bdhjnp	73.68	79.10	81.17	86.30	80.29
bdhjnq	73.76	79.13	80.09	85.13	77.04
bdhjnr	73.79	79.40	79.31	84.35	76.89
bdhjop	70.91	76.79	79.89	85.65	79.30
bdhjoq	71.32	77.22	79.12	84.80	76.23
bdhjor	71.16	77.27	78.36	84.04	75.87
bdhknp	72.79	78.33	80.94	86.18	80.15
bdhknq	73.12	78.60	80.20	85.29	76.96
bdhknr	73.55	79.19	79.72	84.78	76.62
bdhkop	72.27	77.97	80.50	85.70	79.77
bdhkoq	72.37	77.95	79.58	84.67	76.52
bdhkor	72.24	77.75	78.90	83.59	76.10
bdhmnp	70.30	74.17	79.80	83.77	77.89
bdhmnq	69.36	74.07	78.02	82.69	74.58
bdhmnr	69.47	74.08	77.22	81.45	74.30
bdhmop	66.63	69.89	77.57	80.87	76.21
bdhmoq	65.84	69.21	75.90	79.39	73.34
bdhmor	66.31	68.98	75.63	78.12	73.06
begjnp	72.47	78.75	79.11	84.99	78.22

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
begjnq	73.07	79.50	78.51	84.38	74.84
begjnr	73.48	80.11	78.27	84.07	74.68
begjop	74.70	81.10	81.21	87.09	78.74
begjoq	74.67	81.13	80.14	85.99	75.38
begjor	74.23	80.83	79.12	84.90	75.05
begknp					
begknq					
begknr					
begkop	72.26	77.05	79.50	83.91	77.79
begkoq	72.04	76.66	78.39	82.69	74.75
begkor	71.84	76.57	77.47	81.76	74.54
begmnp	72.39	78.20	79.67	85.00	77.63
begmnq	72.30	78.19	78.36	83.84	74.61
begmnr	72.38	78.51	77.82	83.26	74.46
begmop	71.92	77.83	79.27	84.61	77.83
begmoq	72.21	78.42	78.41	83.98	74.81
begmor	72.48	78.60	77.80	83.20	74.35
behjnp	73.64	79.06	81.16	86.28	80.28
behjnq	73.53	78.79	80.09	85.14	77.03
behjnr	73.71	79.33	79.27	84.31	76.88
behjop	70.91	76.79	79.89	85.65	79.34
behjoq	71.32	77.22	79.12	84.80	76.26
behjor	71.16	77.27	78.36	84.04	75.90
behknp	72.88	78.41	80.98	86.22	80.21
behknq	73.21	78.68	80.24	85.33	76.99
behknr	73.76	79.45	79.79	84.92	76.69
behkop	72.20	77.87	80.46	85.65	79.83
behkoq	72.29	77.85	79.54	84.62	76.60
behkor	72.24	77.75	78.90	83.59	76.13
behmnp	70.30	74.17	79.80	83.77	77.89
behmnq	69.36	74.07	78.02	82.69	74.58

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
behmnr	69.47	74.08	77.22	81.45	74.30
behmop	66.60	69.84	77.56	80.84	76.21
behmoq	65.81	69.16	75.89	79.36	73.34
behmor	66.20	68.87	75.57	78.07	73.01
bfgjnp	70.33	76.74	77.23	82.98	74.53
bfgjnq	69.71	75.89	75.62	81.28	71.42
bfgjnr	70.28	76.59	75.60	81.17	71.27
bfgjop	71.05	77.45	78.47	84.18	75.44
bfgjoq	70.34	76.50	76.80	82.36	72.31
bfgjor	70.67	77.13	76.29	81.92	71.98
bfgknp					
bfgknq					
bfgknr					
bfgkop					
bfgkoq					
bfgkor					
bfgmnp	73.17	79.37	80.12	85.83	78.25
bfgmnq	73.34	79.50	79.17	84.80	74.81
bfgmnr	73.44	79.49	78.58	83.97	74.72
bfgmop	73.92	80.24	80.56	86.41	78.31
bfgmoq	74.36	80.70	79.76	85.55	74.95
bfgmor	74.17	80.32	79.03	84.47	74.64
bfhjnp	73.53	78.91	81.28	86.19	79.75
bfhjnq	73.48	78.88	80.18	85.17	76.55
bfhjnr	74.25	79.84	79.81	84.76	76.36
bfhjop	72.97	79.78	81.00	87.20	79.58
bfhjoq	72.32	78.85	79.39	85.35	76.33
bfhjor	72.53	79.01	79.01	84.79	75.92
bfhknp	70.32	76.11	78.02	83.36	78.34
bfhknq	70.85	76.55	77.57	82.77	75.17
bfhknr	71.56	77.22	77.51	82.46	75.02

	DCU 105				PARC 700
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	
bfhkop	72.55	79.00	80.11	85.76	78.73
bfhkoq	72.61	78.66	79.27	84.70	75.49
bfhkor	73.26	79.88	79.14	84.67	75.32
bfhmnp	71.01	76.33	79.82	85.02	78.51
bfhmnq	71.33	77.06	78.94	84.55	75.47
bfhmnr	71.49	77.22	78.21	83.64	75.10
bfhmop	69.57	74.10	79.08	83.45	78.11
bfhmoq	69.67	74.26	78.07	82.54	75.23
bfhmor	69.99	74.94	77.41	82.00	74.81

Appendix E

Parsing Results for F-Structures against the 2,416 F-Structures automatically generated for Section 23

Group 1	Group 2
a Add root node **b** No root node	**c** No unary productions **d** No X → X productions **e** Include all unary productions **f** No unary productions, but keep information
Group 3	Group 4
g Add f-structure annotation **h** No f-structure annotation	**j** Add parent **k** Add grandparent **m** No parent/grandparent transformation
Group 5	Group 6
n Keep Penn-II functional labels **o** Remove Penn-II functional labels	**p** No AUX change **q** Change only true auxiliary verbs **r** Change all auxiliary verb labels

The six groups of transformations used to test transformation interaction. A grammar with one feature from each group is extracted. This gives 288 grammars.

There were a small number of grammars that caused `BitPar` to produce corrupt data, and therefore we are unable to provide the results for those 21 experiments here.

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
acgjnp	72.81	76.07	80.36	83.49	94.84
acgjnq	72.84	76.04	79.46	82.54	94.60
acgjnr	72.96	76.16	78.93	81.97	94.47
acgjop	73.54	76.84	81.09	84.26	96.14
acgjoq	73.44	76.72	80.11	83.26	95.92
acgjor	73.70	77.11	79.64	82.86	95.88
acgknp					
acgknq					
acgknr					
acgkop	75.61	78.27	0.72	68.03	87.82
acgkoq	74.27	76.98	0.72	67.64	86.94
acgkor	73.81	76.43	0.69	67.86	86.98
acgmnp	74.93	78.40	82.28	85.64	98.92
acgmnq	75.01	78.47	81.40	84.73	98.92
acgmnr	75.00	78.42	80.67	83.95	98.79
acgmop	75.05	78.68	82.37	85.85	98.87
acgmoq	75.07	78.70	81.42	84.89	98.87
acgmor	75.02	78.61	80.70	84.08	98.83
achjnp	71.16	74.68	79.49	82.87	95.73
achjnq	71.01	74.48	78.41	81.76	95.69
achjnr	71.04	74.52	77.68	80.98	95.56
achjop	70.49	73.97	79.22	82.51	95.74
achjoq	70.35	73.76	78.18	81.42	95.74
achjor	70.32	73.71	77.40	80.58	95.57
achknp	69.69	72.62	78.20	81.02	94.47
achknq	69.60	72.55	77.21	80.04	94.38
achknr	69.48	72.39	76.40	79.15	94.29
achkop	69.19	72.52	78.22	81.38	95.29
achkoq	69.03	72.32	77.16	80.28	95.25
achkor	69.06	72.33	76.44	79.49	95.20
achmnp	68.81	71.85	77.97	81.09	96.27

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
achmnq	68.65	71.65	76.93	80.01	96.27
achmnr	68.60	71.59	76.17	79.17	96.14
achmop	67.69	70.62	77.50	80.45	96.43
achmoq	67.56	70.45	76.45	79.36	96.47
achmor	67.56	70.50	75.71	78.60	96.47
adgjnp	74.45	77.59	81.87	84.95	98.13
adgjnq	74.56	77.73	81.05	84.15	98.09
adgjnr	74.79	78.06	80.55	83.70	98.26
adgjop	74.66	77.99	82.00	85.23	97.97
adgjoq	74.73	78.07	81.12	84.36	98.01
adgjor	74.80	78.08	80.49	83.63	97.97
adgknp	73.17	75.97	81.09	83.86	97.28
adgknq	73.28	76.14	80.29	83.11	97.19
adgknr	73.30	76.21	79.60	82.44	97.31
adgkop	73.26	76.26	81.14	84.09	97.62
adgkoq	73.39	76.42	80.38	83.36	97.50
adgkor	73.34	76.37	79.57	82.54	97.62
adgmnp	72.95	75.77	80.89	83.67	98.39
adgmnq	73.00	75.85	80.00	82.83	98.34
adgmnr	73.35	76.31	79.53	82.42	98.43
adgmop	72.65	75.53	80.49	83.36	97.72
adgmoq	72.78	75.69	79.69	82.59	97.76
adgmor	72.91	75.81	79.10	81.95	97.68
adhjnp	75.63	78.83	83.37	86.54	100.00
adhjnq	75.83	79.05	82.67	85.86	100.00
adhjnr	75.82	79.13	81.91	85.14	100.00
adhjop	74.00	77.05	82.50	85.53	99.83
adhjoq	74.14	77.20	81.70	84.73	99.83
adhjor	74.13	77.17	80.96	83.92	99.83
adhknp	75.69	79.01	83.48	86.70	99.96
adhknq	76.03	79.32	82.91	86.10	99.96

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
adhknr	76.03	79.28	82.17	85.28	99.96
adhkop	74.58	77.86	82.91	86.11	99.92
adhkoq	74.88	78.15	82.24	85.42	99.92
adhkor	74.73	77.88	81.39	84.41	99.92
adhmnp	72.50	74.66	81.99	84.23	100.00
adhmnq	72.01	74.19	80.60	82.87	100.00
adhmnr	72.16	74.38	79.97	82.24	100.00
adhmop	70.29	71.84	80.75	82.32	99.88
adhmoq	70.04	71.77	79.50	81.31	99.83
adhmor	70.08	71.84	78.82	80.56	99.88
aegjnp	75.08	78.26	82.52	85.62	99.17
aegjnq	75.15	78.37	81.65	84.79	99.17
aegjnr	75.30	78.59	81.05	84.21	99.09
aegjop	75.33	78.62	82.72	85.93	99.34
aegjoq	75.43	78.73	81.88	85.10	99.38
aegjor	75.46	78.73	81.19	84.36	99.30
aegknp	73.46	76.24	81.36	84.11	97.82
aegknq	73.55	76.39	80.56	83.36	97.70
aegknr	73.60	76.49	79.88	82.69	97.86
aegkop	73.98	77.06	81.85	84.88	98.66
aegkoq	74.11	77.22	81.12	84.17	98.67
aegkor	73.93	77.01	80.17	83.18	98.62
aegmnp	73.45	76.24	81.39	84.15	99.46
aegmnq	73.47	76.29	80.49	83.29	99.42
aegmnr	73.80	76.72	80.03	82.88	99.50
aegmop	73.46	76.33	81.38	84.25	99.54
aegmoq	73.53	76.41	80.50	83.37	99.54
aegmor	73.66	76.56	79.95	82.80	99.63
aehjnp	75.64	78.83	83.37	86.54	100.00
aehjnq	75.84	79.06	82.67	85.86	100.00
aehjnr	75.82	79.13	81.91	85.14	100.00

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
aehjop	74.07	77.12	82.56	85.58	99.83
aehjoq	74.20	77.27	81.75	84.78	99.83
aehjor	74.18	77.24	80.99	83.96	99.83
aehknp	75.73	79.06	83.50	86.72	99.96
aehknq	76.08	79.37	82.93	86.12	99.96
aehknr	76.13	79.38	82.25	85.35	99.96
aehkop	74.68	77.95	82.96	86.15	99.92
aehkoq	74.99	78.24	82.30	85.46	99.92
aehkor	74.84	77.99	81.47	84.48	99.92
aehmnp	72.49	74.65	81.99	84.23	100.00
aehmnq	72.00	74.18	80.59	82.87	100.00
aehmnr	72.16	74.38	79.97	82.24	100.00
aehmop	70.29	71.84	80.75	82.32	99.88
aehmoq	70.03	71.77	79.50	81.31	99.83
aehmor	70.04	71.80	78.80	80.54	99.88
afgjnp	72.46	75.73	80.03	83.15	94.29
afgjnq	72.39	75.64	79.03	82.14	94.00
afgjnr	72.60	75.80	78.58	81.61	94.00
afgjop	73.32	76.61	80.92	84.06	95.92
afgjoq	73.28	76.52	79.94	83.05	95.74
afgjor	73.57	76.90	79.50	82.66	95.56
afgknp					
afgknq					
afgknr					
afgkop	75.03	77.64	0.70	67.52	87.23
afgkoq	73.64	76.28	0.70	67.09	86.23
afgkor	73.08	75.69	0.68	67.17	86.10
afgmnp	74.83	78.34	82.20	85.58	98.75
afgmnq	74.89	78.40	81.33	84.69	98.79
afgmnr	75.00	78.49	80.72	84.03	98.79
afgmop	74.94	78.53	82.33	85.78	98.96

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
afgmoq	74.98	78.56	81.40	84.82	98.92
afgmor	75.01	78.56	80.71	84.06	98.83
afhjnp	75.50	78.94	83.20	86.57	98.45
afhjnq	75.56	79.00	82.36	85.72	98.32
afhjnr	75.81	79.36	81.84	85.24	98.23
afhjop	74.73	78.17	82.90	86.20	98.96
afhjoq	74.85	78.29	82.08	85.38	98.91
afhjor	74.99	78.41	81.50	84.73	98.79
afhknp	73.52	76.70	81.38	84.46	95.68
afhknq	73.39	76.60	80.36	83.45	95.28
afhknr	73.56	76.77	79.89	82.90	95.41
afhkop	73.43	76.79	81.80	85.08	98.10
afhkoq	73.51	76.86	80.94	84.20	97.85
afhkor	73.62	76.99	80.37	83.61	97.98
afhmnp	73.63	76.79	82.21	85.42	99.71
afhmnq	73.86	77.07	81.48	84.73	99.71
afhmnr	74.07	77.28	80.96	84.14	99.71
afhmop	72.59	75.83	81.69	84.91	99.71
afhmoq	72.90	76.12	81.09	84.31	99.75
afhmor	72.90	76.19	80.34	83.53	99.79
bcgjnp	72.44	75.75	80.00	83.18	94.34
bcgjnq	72.44	75.72	79.06	82.22	94.06
bcgjnr	72.61	75.87	78.56	81.67	93.97
bcgjop	73.42	76.77	81.01	84.24	95.88
bcgjoq	73.34	76.67	80.03	83.25	95.65
bcgjor	73.58	77.00	79.56	82.80	95.70
bcgknp					
bcgknq					
bcgknr					
bcgkop	75.20	77.73	0.72	67.63	87.26
bcgkoq	73.72	76.25	0.72	67.12	86.13

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
bcgkor	73.24	75.73	0.69	67.32	86.07
bcgmnp	74.90	78.35	82.34	85.68	99.21
bcgmnq	74.96	78.40	81.45	84.76	99.21
bcgmnr	74.93	78.34	80.70	83.96	99.13
bcgmop	75.02	78.58	82.44	85.85	99.17
bcgmoq	75.05	78.61	81.50	84.91	99.17
bcgmor	75.04	78.58	80.82	84.16	99.17
bchjnp	71.18	74.74	79.47	82.85	95.82
bchjnq	71.13	74.63	78.51	81.84	95.82
bchjnr	71.10	74.64	77.74	81.07	95.69
bchjop	70.66	74.14	79.29	82.59	95.79
bchjoq	70.47	73.94	78.19	81.46	95.79
bchjor	70.44	73.96	77.44	80.72	95.71
bchknp	69.67	72.69	78.20	81.08	94.13
bchknq	69.63	72.62	77.25	80.12	94.13
bchknr	69.74	72.75	76.66	79.49	94.08
bchkop	69.92	73.37	78.71	81.98	95.08
bchkoq	69.74	73.17	77.62	80.86	94.99
bchkor	69.78	73.23	76.93	80.13	94.95
bchmnp	68.64	71.47	77.83	80.76	96.31
bchmnq	68.42	71.22	76.73	79.61	96.31
bchmnr	68.61	71.48	76.19	79.08	96.31
bchmop	67.41	70.21	77.38	80.20	96.60
bchmoq	67.31	70.10	76.32	79.11	96.60
bchmor	67.40	70.24	75.58	78.35	96.64
bdgjnp	74.23	77.39	81.56	84.67	97.26
bdgjnq	74.30	77.50	80.71	83.85	97.14
bdgjnr	74.38	77.67	80.09	83.26	97.34
bdgjop	74.52	77.79	81.86	85.04	97.88
bdgjoq	74.60	77.90	81.02	84.22	97.84
bdgjor	74.64	77.98	80.34	83.54	98.01

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
bdgknp					
bdgknq					
bdgknr					
bdgkop	73.00	75.95	80.87	83.77	97.28
bdgkoq	73.03	75.99	80.06	82.97	97.28
bdgkor	73.16	76.15	79.35	82.26	97.24
bdgmnp	72.95	75.77	80.89	83.67	98.39
bdgmnq	73.00	75.85	80.00	82.83	98.34
bdgmnr	73.35	76.31	79.53	82.42	98.43
bdgmop	72.65	75.53	80.49	83.36	97.72
bdgmoq	72.78	75.69	79.69	82.59	97.76
bdgmor	72.91	75.81	79.10	81.95	97.68
bdhjnp	75.61	78.82	83.36	86.53	100.00
bdhjnq	75.82	79.03	82.65	85.84	100.00
bdhjnr	75.80	79.10	81.91	85.13	100.00
bdhjop	74.00	77.05	82.51	85.53	99.83
bdhjoq	74.14	77.20	81.70	84.73	99.83
bdhjor	74.13	77.17	80.96	83.92	99.83
bdhknp	75.54	78.86	83.41	86.62	99.25
bdhknq	75.83	79.13	82.77	85.96	99.25
bdhknr	75.80	79.09	81.98	85.13	99.34
bdhkop	74.58	77.86	82.91	86.11	99.92
bdhkoq	74.88	78.15	82.24	85.42	99.92
bdhkor	74.73	77.88	81.39	84.41	99.92
bdhmnp	72.50	74.66	81.99	84.23	100.00
bdhmnq	72.01	74.19	80.60	82.87	100.00
bdhmnr	72.16	74.38	79.97	82.24	100.00
bdhmop	70.29	71.84	80.75	82.32	99.88
bdhmoq	70.04	71.77	79.50	81.31	99.83
bdhmor	70.08	71.84	78.82	80.56	99.88
begjnp	74.77	77.98	82.11	85.26	98.22

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
begjnq	74.86	78.11	81.27	84.45	98.09
begjnr	74.82	78.17	80.52	83.73	98.09
begjop	75.31	78.59	82.69	85.88	99.29
begjoq	75.39	78.69	81.85	85.06	99.21
begjor	75.36	78.66	81.06	84.25	99.25
begknp					
begknq					
begknr					
begkop	73.76	76.82	81.62	84.61	98.45
begkoq	73.88	76.92	80.88	83.88	98.54
begkor	73.83	76.86	80.04	83.01	98.37
begmnp	73.45	76.24	81.39	84.15	99.46
begmnq	73.49	76.31	80.52	83.32	99.46
begmnr	73.80	76.72	80.03	82.88	99.50
begmop	73.49	76.37	81.42	84.29	99.63
begmoq	73.57	76.47	80.56	83.46	99.63
begmor	73.66	76.56	79.95	82.80	99.63
behjnp	75.63	78.83	83.37	86.54	100.00
behjnq	75.47	78.65	82.67	85.83	100.00
behjnr	75.80	79.11	81.91	85.13	100.00
behjop	74.07	77.12	82.56	85.58	99.83
behjoq	74.20	77.27	81.75	84.78	99.83
behjor	74.18	77.24	80.99	83.96	99.83
behknp	75.57	78.89	83.42	86.63	99.25
behknq	75.86	79.17	82.77	85.97	99.25
behknr	75.86	79.14	82.00	85.15	99.34
behkop	74.68	77.95	82.96	86.15	99.92
behkoq	74.99	78.24	82.30	85.46	99.92
behkor	74.84	77.99	81.47	84.48	99.92
behmnp	72.49	74.65	81.99	84.22	100.00
behmnq	72.00	74.18	80.59	82.87	100.00

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
behmnr	72.16	74.38	79.97	82.24	100.00
behmop	70.29	71.84	80.75	82.32	99.88
behmoq	70.03	71.77	79.50	81.31	99.83
behmor	70.04	71.80	78.80	80.54	99.88
bfgjnp	72.10	75.41	79.66	82.84	93.70
bfgjnq	72.00	75.33	78.64	81.84	93.41
bfgjnr	72.17	75.45	78.16	81.28	93.41
bfgjop	73.20	76.58	80.82	84.05	95.74
bfgjoq	73.19	76.50	79.88	83.07	95.51
bfgjor	73.45	76.85	79.41	82.66	95.38
bfgknp					
bfgknq					
bfgknr					
bfgkop					
bfgkoq					
bfgkor					
bfgmnp	74.75	78.21	82.19	85.55	99.09
bfgmnq	74.81	78.28	81.33	84.66	99.13
bfgmnr	74.92	78.36	80.73	84.01	99.17
bfgmop	74.89	78.40	82.36	85.74	99.21
bfgmoq	74.97	78.47	81.49	84.84	99.25
bfgmor	75.03	78.52	80.82	84.14	99.21
bfhjnp	75.34	78.80	83.03	86.40	98.62
bfhjnq	75.43	78.88	82.24	85.61	98.62
bfhjnr	75.67	79.25	81.74	85.16	98.49
bfhjop	74.70	78.29	82.84	86.30	99.04
bfhjoq	74.85	78.43	82.06	85.49	99.04
bfhjor	75.01	78.67	81.52	84.98	98.96
bfhknp	73.43	76.59	81.34	84.35	95.91
bfhknq	73.46	76.58	80.49	83.47	95.59
bfhknr	73.66	76.85	79.96	82.94	95.59

	Automatically Annotated Section 23				Section 23
Grammar	Preds Only -LDD Res.	Preds Only +LDD Res.	All GFs -LDD Res.	All GFs +LDD Res.	Frag.
bfhkop	73.77	77.34	81.93	85.36	98.06
bfhkoq	73.93	77.44	81.24	84.60	97.80
bfhkor	74.09	77.65	80.67	84.03	97.97
bfhmnp	73.52	76.46	82.24	85.26	99.75
bfhmnq	73.73	76.74	81.47	84.54	99.75
bfhmnr	74.00	77.07	80.99	84.06	99.79
bfhmop	72.26	75.30	81.53	84.57	99.75
bfhmoq	72.56	75.58	80.91	83.96	99.75
bfhmor	72.55	75.64	80.16	83.19	99.79