Improved Evaluation and Parse Reranking for Combinatory Categorial Grammar

DOMINICK NG
SID: 307135365

Supervisor: Dr James R. Curran
Associate Supervisor: Dr Matthew Honnibal

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Information Technology (Honours)

School of Information Technologies
The University of Sydney
Australia

1 November 2010

THE UNIVERSITY OF
SYDNEY

Student Plagiarism: Compliance Statement

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Dominick Ng

Signature: **Date**:

Abstract

Accurate syntactic parsing of text is crucial for automatically understanding language. This makes parser performance and the metrics used to measure parser performance central to the goals of natural language processing. Unfortunately, the field-standard parser evaluation methods are tied to particular grammar formalisms and testing data, and are uninformative when considering the cross-domain performance of a parser. The wide variety of parsers built on different formalisms makes it difficult to know whether new developments for improving parsing accuracy are relevant across different parsers. For example, supervised reranking of n-best parses has been shown to improve the accuracy of a number of parsers, but the potential benefits of reranking for parsers based on different grammars remains unclear.

This thesis addresses several open questions regarding parser evaluation and performance. We assess a new style of parser evaluation for the C&C Combinatory Categorial Grammar parser, demonstrating that an *extrinsic* task-based evaluation of parsers is meaningful and can be successfully implemented. This improves on traditional evaluation metrics by being formalism-agnostic and requiring no expensive linguistic annotation to supply gold-standard test data. It also provides a fairer assessment of parser performance based on how parsing is used in the field.

We develop a parser reranker for the C&C parser, and provide a comprehensive blueprint for implementing the features used by the system. Our initial experiments showed that state-of-the-art features from an existing reranker substantially reduce the performance of the C&C parser. Following a systematic analysis of errors made by the parser, we develop new reranking features that are better tailored to distinguishing cases of these errors. Our experiments showed that these features produce significant performance improvements over the baseline parser without the use of additional annotated data.

Through this work we have shown that rerankers can produce respectable accuracy improvements for parsers if they are targeted towards the parser in question and the errors it makes. Insights from our analysis of reranking features and parser error will be useful for other reranker implementations, while our work on evaluation will contribute to more representative methods of evaluating parsing performance. The improved parser performance will lead to improvements in downstream systems, providing for more intelligent information search, storage, and management.

Acknowledgements

I am indebted to Matthew Honnibal, for mentoring me through this year and gladly sharing his experience when my own proved insufficient.

This thesis was immeasurably improved by the feedback of Ben Hachey, James Kane, Jonathan Kummerfeld, Tim Dawborn, and Sam Lewin. It was made financially possible through the support of a University of Sydney Merit Scholarship.

Thanks go also to the members of ə-lab for creating a stimulating research environment, where questions never go unanswered and fun is always had.

I am particularly grateful for the support of my incredibly tolerant cohabitants, Sophie Liang, Stephen Merity, and Bin Zhou, who have made this year exceptional for me in so many ways.

In all my endeavours, I have always had the unquestioning blessing of my family, close and near to home. Thank you to Alson, Esther, Patrick, and Samantha for being there for me, and so often bringing yourselves to me, as far as I am away.

Finally, my deepest gratitude and appreciation to James Curran and Tara Murphy, for inspiring teaching, encouraging an appreciation of research, and guiding me ever since the first day of university when I stepped into your class.

For my parents

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

The advent of the information age has dramatically increased the amount of data generated and stored in computers. Search has become a key part of everyday life, and we are growing increasingly reliant on automated processing to help locate the information we want. Unfortunately, most systems possess only a rudimentary understanding of the natural language that much of our information is stored in. Creating systems that can use the structure and semantics of language in an intelligent fashion is a core goal of *Natural Language Processing* (NLP), a field of research within Artificial Intelligence (AI).

A key component of understanding any language is extracting its underlying grammatical structure. Natural language is inherently ambiguous in that most sentences can have multiple interpretations that may be context-sensitive. Many of these ambiguities can be resolved through accurate *syntactic analysis* of sentences, which is performed by *parsers*. Parsers such as the C&C Combinatory Categorial Grammar parser (Clark and Curran, 2007b) will read input sentences and then produce an analysis that could be in several forms, including a *parse tree* or a *list of dependencies between words*.

Parsing is an important component of many systems that process language. In *question answering*, parsing is used to analyse the structure of questions and find answers that match the constraints posed (Echihabi and Marcu, 2003). Automated *machine translation* from one natural language to another uses parsing to extract the underlying semantics from a sentence in one language in order to re-express it in another language (DeNeefe and Knight, 2009). Accurate parsing is crucial for constructing useful systems for these purposes.

Research on English parsing has been dominated by one main corpus: the Penn Treebank Wall Street Journal collection (Marcus et al., 1993). The Treebank is a collection of 1.1 million words of English newswire annotated with syntactic structures that is used by parsers as training data. Parser evaluation has also concentrated on a single metric: the PARSEVAL bracketed F-score calculated over a held-out section of the Treebank (Black et al., 1991). PARSEVAL measures the similarity in structure between the parser's analysis and the gold-standard parse in the held-out set. This combination of training data and evaluation has defined the parsing task for nearly two decades.

The standardization of research around the Penn Treebank and PARSEVAL has driven improvements in parsing techniques. The core parsing algorithm constructs small trees from fragments of the sentence, and combines these together to form a complete analysis. Dynamic programming is necessary to ensure that this process does not take exponential time; this restricts the information that the parser can use to a local context in each subtree. The most accurate parsers use the Penn Treebank to build a statistical model that can be used to predict the structure of unknown text (Collins, 1999).

More recently, methods for *supervised reranking* of the top analyses produced by a parser have been shown to improve accuracy by over 1% on the widely-used Collins and Charniak parsers (Collins, 2000; Charniak and Johnson, 2005). Reranking improves performance because it is not subject to the same constraints as the parser: arbitrary complex information may be used in a reranker, while the parser is restrained to a local context. The reranking model also provides a framework for incorporating information from external sources into parsing, which under the parser's more restricted model would be prohibitively difficult. Reranking theoretically allows parsers to consider more information from existing training data without requiring additional annotated text.

More recent work on reranking using the Charniak system on a different parser (the Berkeley parser; Petrov et al., 2006) has shown less impressive results. Johnson and Ural (2010) recorded a negligible impact on the Berkeley parser's performance using the reranker described in Charniak and Johnson (2005). This raises the question of whether reranking is a viable general technique for improving parser performance, or whether the characteristics of the Collins and Charniak systems allow reranking to improve accuracy. We investigate this by implementing a reranker for the C&C parser, comparing the features of Charniak and Johnson (2005) against novel features motivated by our extensive analysis of CCG parsing errors.

The field-standard PARSEVAL metric is imperfect for a number of reasons. While parsers should be able to process text from a variety of domains with different characteristics, the PARSEVAL metric is run over a held-out portion of in-domain data. Given that parsers are *domain-dependent* and perform better on in-domain data that out of domain data, this raises the possibility that PARSEVAL scores are unreasonably optimistic and do not reflect how the parser truly performs on unknown text (Sekine, 1997). Also, PARSEVAL punishes discrepancies in surface form between a parsed sentence and the gold standard that may actually create no differences in the underlying meaning of a sentence (Carroll et al., 2002). These two concerns affect the validity of PARSEVAL results.

Evaluation over the underlying dependencies produced by parsers has increased in popularity to combat the shortcomings of the PARSEVAL metric (Carroll et al., 2002). However, dependency evaluation is not without its flaws and does not address the difficulty in *cross-formalism comparison* – evaluation of parsers based on different grammatical formalism. Conversion between grammatical formalisms is typically lossy and results in artificial upper bounds on parser performance (Clark and Curran, 2007a, 2009). Current evaluation methods also require comparison against gold-standard parses or dependencies; creating these resources is time-consuming and requires expert linguistic annotation. A clear need exists for a more representative parser evaluation metric that is less reliant on expert annotation in order to better gauge real-world performance.

1.1 Contributions

1.1.1 Improved Evaluation for the C&C Parser

We develop a new system for evaluating the C&C parser. We use the Parser Evaluation using Textual Entailments task at the 5th International Workshop on Semantic Evaluation (SemEval-2010; Yuret et al., 2010) as the basis of our approach. The evaluation uses an extrinsic textual entailment recognition task over sentences that are formalism-agnostic to the parser being evaluated.

Textual entailment recognition is the task of predicting whether a text sentence implies the meaning of a hypothesis sentence. This is a natural human competence that can be recognised by any fluent speaker of the language. Use of an extrinsic task addresses many of the issues of existing parser evaluation metrics like PARSEVAL. Entailment recognition is not tied to a particular grammar or output format, and as the task is based on a natural human competence, data can be created using non-expert annotators at a fraction of the cost of linguistic annotation.

Our evaluation system parses the text and hypothesis sentences using the C&C parser. We use the lists of dependencies produced by the parser to generate *predicates* that reflect the core dependencies in the text and hypothesis sentences. Following heuristic pruning of unwanted predicates, we attempt to match at least one predicate from the hypothesis sentence in the text sentence. If a match is found, a YES entailment is reported, otherwise NO is reported. The final score for the parser is the overall accuracy of entailment prediction using this system.

We placed 2nd in the task evaluation at SemEval-2010, with an overall accuracy of 70% on 301 test pairs. The first-placed system also evaluated the C&C parser using a different methodology, and achieved an accuracy of 72% (Rimell and Clark, 2010).

In testing, we found that our evaluation system introduced an upper bound of 87% accuracy for the C&C parser. This compares favourably with previous efforts at cross-formalism comparison using C&C; a many-to-many conversion of CCG dependencies to the grammatical relations schema of Briscoe and Carroll (2006) resulted in an upper bound of 84.8% (Clark and Curran, 2007a). If this upper bound can be further improved, the new scheme shows considerable potential due to its relative simplicity. With a larger data set, we conclude that it could provide a new platform for improved cross-formalism evaluation.

1.1.2 Parse Reranking for the C&C Parser

We reimplement the supervised reranker described in Charniak and Johnson (2005) for the C&C parser, and provide a comprehensive blueprint for developing a reranker. We experiment with classification and regression-based approaches as well as a number of feature pruning cutoffs to explore how well the Charniak reranker adapts to CCG. Our core finding is that features suited to reranking the Charniak Penn Treebank-style parser are unhelpful for reranking CCG parses, no matter the pruning strategy or machine-learning approach. Parsing accuracy drops significantly using the reranker against the baseline parser in all experiments using the features as described in Charniak and Johnson (2005).

Motivated by our detailed analysis of errors made by the C&C parser, we develop a set of novel features for CCG reranking. Our new features specifically reflect aspects of CCG rather than the phrase-structure grammar used in the Penn Treebank. We also target prepositional phrase ambiguities and conjunction errors in parses as our analysis finds that these two error classes contribute to over 25% of incorrect dependencies produced by the C&C parser.

Using the same reranker implementation with the new features achieves a statistically significant performance improvement over the baseline parser. However, the performance improvements we recorded are much smaller than those of Collins (2000) and Charniak and Johnson (2005). Our conclusion is that for a reranker to improve parsing accuracy, it must be carefully tuned to the parser and targeted to known sources of error. This insight will prove useful for future reranker implementations.

1.2 Outline

Chapter 2 gives an outline of syntax and parsing, along with a description of the parsing field and a review of previous work. Chapter 3 describes the Combinatory Categorial Grammar formalism, its CCGbank corpus, and the C&C parser. Chapter 4 describes various frameworks for parser evaluations, their deficiencies, and our work on developing a new evaluation system for the C&C parser using Textual Entailment Recognition.

Our initial reimplementation of the Charniak reranker for the C&C parser is described in Chapter 5. An in-depth analysis of dependency errors made by the parser follows in Chapter 6, providing motivation for developing the novel reranking features that are described in Chapter 7. Chapter 8 summarises our work, and describes how it impacts on the field of parsing and parser evaluation.

The evaluation system described in Chapter 4 was published in the Proceedings of the 5th International Workshop on Semantic Evaluation under the title SCHWA*: PETE using CCG Dependencies with the C&C Parser* (Ng et al., 2010). Preliminary reranking results from Chapters 5 and 7 have been accepted for publication under the title *Reranking a wide coverage CCG parser* at the Australasian Language Technology Workshop 2010.

CHAPTER 2

Background

2.1 Syntax and Grammars

Sentences in English are inherently structured. Despite containing the same words, sentences (1) and (2) have different meanings, just as sentences (1) and (3) express the same meaning with different words.

(1) Jack baked a cake for Jill.

(2) Jill baked a cake for Jack.

(3) Jill was the one for whom Jack baked a cake.

Clearly, it is not only the *surface form* of each sentence which shapes the meaning, but also the underlying order and structure with the sentence. Recognising this structure is necessary for accurate natural language interpretation.

The rules and principles that govern the way that words are combined to form a sentence are referred to as *syntax*. The syntax of a particular language is captured by a *grammar*, which describes the relationships between various constructions in the language. Changes in these underlying relationships create the similarities and differences in meaning between sentences (1), (2), and (3).

A wide variety of grammar formalisms with varying degrees of expressive power have been developed in the study of linguistics and parsing. Each grammar licenses *derivations* of valid sentences, which are a list of rules or steps that are applied to form analyses. *Constituent structure grammars* (also termed *context-free grammars* or *phrase-structure grammars*) and *dependency grammars* are two classes of grammar commonly used in natural language processing (NLP).

2.1.1 Constituent Structure Grammars

Constituents are components of a sentence that function as a single unit. They are combined together into *constructions*, with successively larger constructions being joined to create the full sentence. The hierarchy of constituents in a sentence is termed its *constituent structure* or *phrase structure*.

The smallest constituents of a sentence are the individual words. Each word in a sentence has a syntactic class known as a *part of speech* (POS) that defines its basic syntactic behaviour. *Part of speech tagging* is the process of assigning each word in a sentence with a POS *tag* under a given scheme of tags. The Penn Treebank POS tag set (Marcus et al., 1993) is the predominant scheme used today. Examples of parts of speech and their Penn Treebank tags include *nouns* (NN), $verbs$ (*VB*), *adjectives* (JJ), *determiners* (DT), and *prepositions* (IN). Each class has different distributional properties within sentences. The full Penn Treebank part-of-speech tag set is given in Appendix A.

Word-level constituents are combined into *phrase constituents* such as *noun phrases* and *verb phrases*. Phrases are named after the word that functions as the *head*, which is generally the 'core' word necessary for grammaticality. Noun phrases are headed by nouns and might consist of a determiner, followed by zero or more adjectives (or nouns), followed by the head noun (e.g. *the large green cake*). Verb phrases are headed by verbs and might consist of the head verb followed by a noun phrase (e.g. *baked a large green cake*). As constituents are combined, the head of the larger construction is determined by a number of guidelines; the usual head of English sentences is the head of the main verb phrase.

The combination of constituents into constructions is usually represented through *context-free production rules*. *Labels* such as NP for noun phrases and VP for verb phrases are assigned to constituents. Some typical rules using the Penn Treebank POS tags described in Appendix A may include

$$\begin{aligned} S &\rightarrow NP\ \ VP \\ NP &\rightarrow \left\{ \begin{matrix} DT\ \ JJ^*\ \ NN \\ NN \end{matrix} \right\} \\ VP &\rightarrow VB\ \ NP\ \ PP \\ PP &\rightarrow IN\ \ NP \end{aligned}$$

where S = sentence, NP = noun phrase, DT = determiner, JJ = adjective, NN = noun, VP = verb phrase, VB = verb, PP = prepositional phrase, and IN = preposition.

Figure 2.1 shows a tree corresponding to constituent construction in a sentence. The tree also represents a derivation of the sentence under the grammar. Each node represents a constituent or word, and nodes with children represent a direct application of one of the production rules with the parent on the left hand side and the children on the right hand side. In this way production rules can be used to describe how constituents can be used to form sentences in a language.

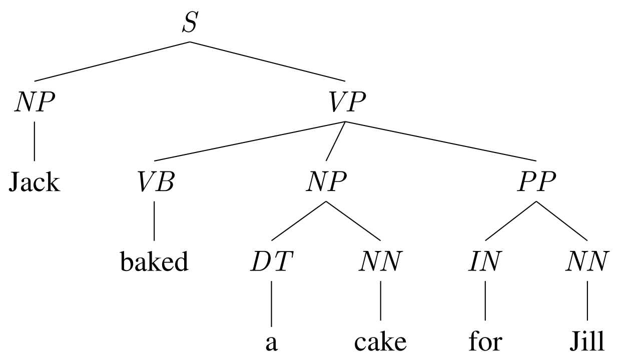


FIGURE 2.1: A constituent structure tree for sentence (1).

In a computational context it is common to use a bracketed format to serialise trees and maintain the constituency structure. The tree in Figure 2.1 may be represented by the bracketed string `(S(NP(NN Jack))(VP(VB baked)(NP(DT a)(NN cake))(PP(IN for)(NN Jill))))`.

2.1.2 Dependency Grammars

Dependencies are functional relationships between pairs of words in a sentence. Also termed *dependency relations*, the set of dependencies for a particular sentence represents its underlying structure in terms of word function. Dependency grammars are particularly useful for languages such as Czech and Finnish, which allow greater freedom of word order than English. Additionally, the constituent structure of a sentence alone is typically not useful for many tasks in NLP as several constituent structures may yield equivalent underlying dependencies.

The two words in a dependency relationship are termed the *head* (not to be confused with the head of a constituent) and the *dependent*. The head creates the dependency with an open *argument slot* for the dependent to fill. Heads may create multiple dependencies as well as dependencies with slots for multiple arguments.

In our first example (sentence (1)), the word *baked* is the head of two dependencies: `subject`, with dependent *Jack*, and `object`, with dependent *cake*. The dependencies directly encode that *Jack* is the subject of the sentence and a *cake* is what he is acting upon via *bake*. These relationships can be represented visually with a directed graph (see Figure 2.2) or as a list of string tuples with the format `(type head dependent)`, e.g. `(subj baked Jack)` and `(obj baked cake)`. Other common dependency types include `modifiers`, `determiners`, and `conjuncts`.

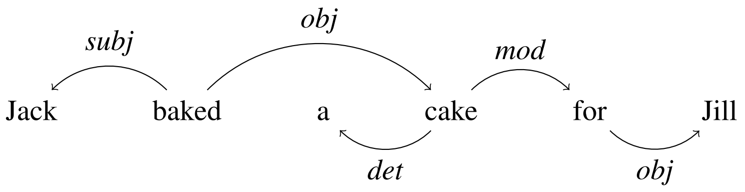


FIGURE 2.2: A directed graph representing the dependencies in sentence (1), where *subj* = subject, *obj* = object, *det* = determiner, and *mod* = modifier relationships.

Briscoe (2006) describes a scheme of dependency relations referred to as *grammatical relations* (GRs). GRs are used in the RASP system (Briscoe et al., 2006) and can be produced by the C&C parser (Clark and Curran, 2007b). While the term grammatical relations can be used to refer to general dependency relations of no specific scheme, we will use it exclusively to refer to Briscoe's dependency scheme.

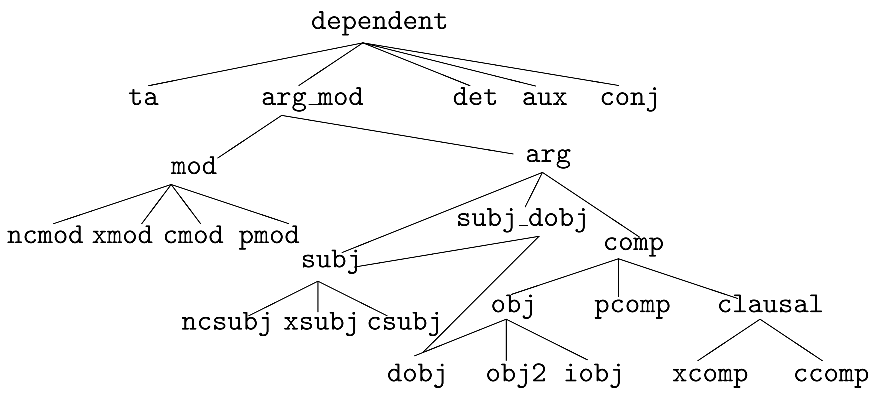


FIGURE 2.3: The grammatical relations hierarchy (Briscoe, 2006).

The GR scheme arranges relations into a tree-like hierarchy, as shown in Figure 2.3. The hierarchy is not strictly a tree due to the inclusion of the `subj_dobj` relation, which avoids disambiguating between the subject and direct object of a verb. There are different degrees of specificity for each relation: for example, the `subj` relation encompasses the `ncsubj`, `xsubj`, and `csubj` relations, each of which describes a more restricted domain of subject dependencies. Briscoe (2006) mentions that the more

GRs	Description	Example
`conj`	conjunction	Kim likes oranges **and** *apples*
`aux`	auxiliary	Kim *has* been **sleeping**
`det`	determiner	*The* **men** arrive
`ncmod`	non-clausal modifier	The *old* **men** slept
`xmod`	unsaturated predicative modifier	**Who** to *talk* to
`cmod`	saturated clausal modifier	*Although* he came, Kim **left**
`pmod`	PP modifier with a PP complement	He **went**, *off* into the darkness
`ncsubj`	non-clausal subject	*Kim* wants to **go**
`xsubj`	unsaturated predicative subject	To **win** the Cup *requires* cash
`csubj`	saturated clausal subject	That he *came* **matters**
`dobj`	direct object	I **gave** *it* to Kim
`obj2`	second object	I **gave** Kim *toys*
`iobj`	indirect object	Kim **flew** *to* Paris from Geneva
`pcomp`	PP which is a PP complement	Kim **climbed** *through* the attic
`xcomp`	unsaturated VP complement	Kim **thought** *of* leaving
`ccomp`	saturated clausal complement	Kim **asked** him *about* playing rugby
`ta`	textual adjunct delimited by punctuation	He made the **discovery**: Kim *was* the host

TABLE 2.1: A list of the GRs with examples (adapted from Briscoe (2006)). Heads are bolded and dependents are italicised.

general relationships (including `mod`, `subj`, `comp`, and `subj_dobj`) are included for the purposes of cross-formalism parser evaluation (as opposed to being generated by a particular parser). Examples of the various grammatical relations in sentences are presented in Table 2.1.

2.2 Parsing

Parsing is the process of assigning syntactic structure to sentences with respect to a grammar. Typically the parsing process will take a sentence as input, and produce the corresponding constituent structure tree or set of dependency relations for that sentence, though some grammar formalisms encode both kinds of information.

2.2.1 Syntactic Ambiguity and the need for parsing

Parsing is an important aspect of processing text as languages usually exhibit *compositional semantics*: the meaning of a sentence is determined by the meanings of its constituent parts and the rules used to combine them. In this way meaning is shaped and informed by syntactic relationships in sentences. Parsing also plays an important role in addressing the inherent ambiguities in natural language. For example, sentence (4) is ambiguous as to whether Jack attended a circus that had lions or if he went with lions to a circus.

(4) Jack attended the circus with the lions.

The ambiguity in sentence (4) arises directly from an ambiguity in syntax, namely *prepositional phrase attachment* ambiguity. In the case of Jack going to a circus that has lions, the prepositional phrase *with the lions* is said to *attach* to the noun *circus*; the meaning of the prepositional phrase complements or supplements the noun. The prepositional phrase would attach to the verb *attend* in the case where Jack was accompanied to the circus by lions. Syntactic ambiguities such as prepositional phrase attachment have a direct correspondence to constituent structure trees; each possible attachment location reflects another possible tree for the sentence. The different attachment locations for *with the lions* correspond to the trees seen in Figures 2.4 and 2.5.

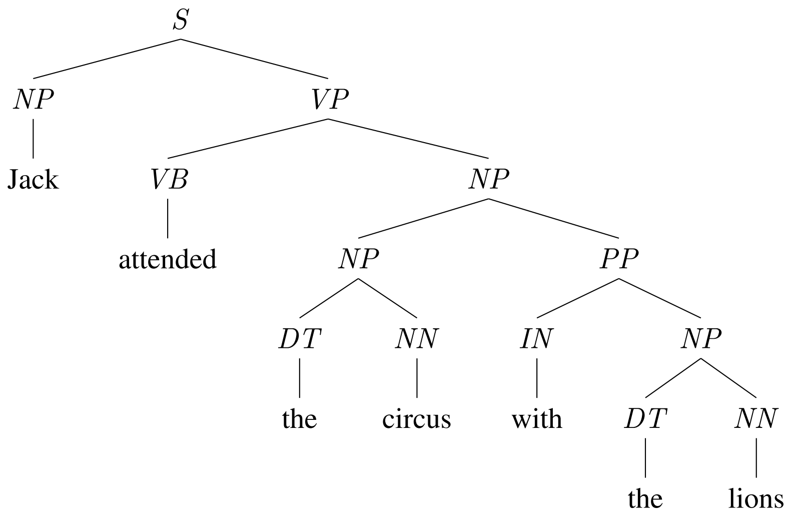


FIGURE 2.4: A tree for sentence (4), where the circus has lions.

The problem of syntactic ambiguity is often easy for humans to resolve through the use of contextual knowledge and common sense. It is commonly known that while lions are seldom taken as companions to a circus, they are a frequent attraction at the circus. Unfortunately, it is difficult to integrate this knowledge into a computational system due to the amount of knowledge required. This makes the resolution of syntactic ambiguity essential for many practical NLP tasks, including question answering systems (Echihabi and Marcu, 2003), statistical machine translation (DeNeefe and Knight, 2009), and dialogue systems (Nederhof et al., 1997).

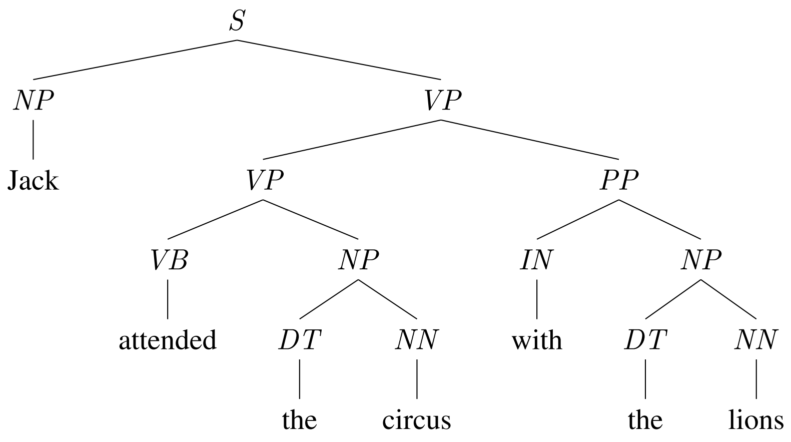


FIGURE 2.5: A tree for sentence (4), where Jack is accompanied by lions.

Ambiguity also affects the efficiency of parsers by increasing the possible number of trees that must be considered. Church and Patil (1982) describes how many common constructions in English, including prepositional phrases, coordination (two or more phrases combined with a conjunction such as *and*), and nominal compounds (phrases composed of multiple nouns, e.g. *credit card fraud*) can create a combinatoric explosion of possible trees with different attachment and branching structures. Parsing algorithms typically address this problem through memoization (see Section 3.3.1 for an explanation of one such method).

2.3 Corpora

The earliest parsers used a *knowledge engineering* approach, where linguists would curate a grammar to parse sentences. The grammar would typically be composed of context-free production rules (similar to those presented in Section 2.1.1) as well as guidelines for how to assign words to various structures. One example is the IBM Computer Manuals parser described in Black et al. (1993), which used a rule-based grammar developed at IBM.

Unfortunately, the process of manual curation is time-consuming and labour intensive, requiring linguistic expertise and incurring significant costs to create the grammar and maintain it. This influenced the move to statistical approaches, which dominate the field today. The most accurate parsers are fully *supervised* in that they use information gathered from annotated training data to select the most likely analysis for a given sentence. The training data is usually composed of collections of texts called *corpora* which are annotated with gold-standard syntactic analyses. Statistical parsers construct a *parsing model* based on the text in the corpora for analysing new sentences. As statistical parsers can automatically learn grammars from new training data, the linguist's role has shifted from defining rules for parsing to creating the gold-standard resources necessary for training parsers.

The reliance on expertly annotated corpora has provided some standardisation in approach and evaluation, defining a stable and consistent research environment. However, the general problem of standardisation – whether the whole field is "overfitting" to the evaluation – remains an open question. The concentration on one measure has prevented the uptake of alternative evaluation metrics that have been proposed to address a number of weakness of PARSEVAL. We will address this issue further in Chapter 4.

2.3.1 The Penn Treebank

The advent of statistical parsing came about with the first release of the Penn Treebank in the early 1990s. Among other resources, the Treebank contained 1.1 million words taken from the 1989 Wall Street Journal (WSJ) newswire annotated with constituent structure trees. The constituent structures were added to sentences using an automated annotation process followed by manual correction. A sample Penn Treebank tree in labelled bracketed format for sentence (5) is given in Figure 2.6. The annotation process for the Treebank is described in detail in Marcus et al. (1993).

(5) If not for a 59.6% surge in orders for capital goods by defense contractors, factory orders would have fallen 2.1%.

The Penn Treebank was the result of an 8-year (1989-1996) project to create a large, annotated corpus to aid research in linguistics (Taylor et al., 2003). It was unmatched in scale and detail at the time of its release, and the WSJ section has become the de facto standard training and testing data for statistical parsers. Many supervised parsers used today are trained directly on the WSJ data and produce output in a Treebank-style bracketed string format (such parsers will be referred to as *Penn Treebank parsers*).

Systems based on alternative formalisms are commonly trained over data derived from the WSJ data through conversion and/or additional annotation, for example, the Lexical Functional Grammar (LFG) corpus described in Cahill et al. (2008) or CCGbank, a Combinatory Categorial Grammar (CCG) corpus described in Hockenmaier and Steedman (2007). We will describe CCG in more detail in Chapter 3. The influence of the WSJ section in parsing is such that the term "Penn Treebank" is typically used to

```
( (S
    (SBAR-ADV (IN If)
      (FRAG (RB not)
        (PP (IN for)
          (NP
            (NP (DT a)
              (ADJP (CD 59.6) (NN %) )
              (NN surge) )
            (PP-LOC (IN in)
              (NP
                (NP (NNS orders) )
                (PP (IN for)
                  (NP (NN capital) (NNS goods) ))
                (PP (IN by)
                  (NP (NN defense) (NNS contractors) ))))))))
    (, ,)
    (NP-SBJ (NN factory) (NNS orders) )
    (VP (MD would)
      (VP (VB have)
        (VP (VBN fallen)
          (NP-EXT (CD 2.1) (NN %) ))))
    (. .) ))
```

FIGURE 2.6: An annotated tree from Section 00 of the Penn Treebank for sentence (5).

refer to the WSJ collection, even though this is only a subset of the entire corpus. We will adhere to this convention in this thesis.

Some pragmatic compromises were made during the annotation of the Penn Treebank in order to simplify the process. In particular, Marcus et al. (1993) notes that a major goal of the Treebank process was to allow annotators to only indicate the structures of which they were certain. This resulted in a skeletal analysis where much syntactic detail in sentences is left unannotated, and the most prominent area of deficiency is in noun phrases. The internal structure of noun phrases is not annotated in the Treebank; instead, noun phrases are left as flat trees.

(6) [[crude oil] prices]

(7) [world [oil prices]]

In sentence (6), the noun phrase has a left-branching structure, whereas in sentence (7) the phrase has a right-branching structure. In the Penn Treebank, sentence (6) would be assigned the flat bracketed structure of `(NP (NN crude) (NN oil) (NN prices))` rather than the correctly bracketed

`(NP (NML (NN crude) (NN oil)) NN prices)`. Any system trained on the Penn Treebank is unable to model this branching, directly affecting tasks such as named entity recognition that require accurate noun phrase structure. To address this problem Vadas and Curran (2007) present an updated version of the Penn Treebank with new gold-standard labels for ambiguous noun phrases.

2.3.2 The PARC 700 Dependency Bank

The PARC 700 Dependency Bank (or DepBank) consists of 700 randomly selected sentences from Section 23 of the Penn Treebank. Each sentence was automatically parsed before manual correction to produce gold-standard dependency output (King et al., 2003). The dependency relation scheme in the DepBank is broadly similar to that of Briscoe (2006) referred to in Section 2.1.2. Briscoe and Carroll (2006) describe a new version of the DepBank for evaluating the RASP system: this version is publicly available and reflects the Briscoe and Carroll GR scheme more closely. Figure 2.7 gives the DepBank and Briscoe et al. dependency relation annotations of sentence (8).

(8) Ten of the nation's governors meanwhile called on the justices to reject efforts to limit abortions.

2.4 PARSEVAL

The field-standard evaluation metric for parsers is the calculation of the PARSEVAL bracketed string metrics (Black et al., 1991) over a held-out section of the Penn Treebank. PARSEVAL measures the *constituent accuracy* of each individual sentence – how similar the constituents in the analysis produced by the parser are to the gold standard from the Treebank. Constituent accuracy is calculated by examining the similarity between the bracketed string representations of each individual sentence. The metrics of *precision (P)* and *recall (R)* are calculated for each sentence as follows:

$$P = \frac{\text{\# pairs of brackets placed correctly by the parser}}{\text{\# pairs of brackets in the parser output}}$$

$$R = \frac{\text{\# pairs of brackets placed correctly by the parser}}{\text{\# pairs of brackets in the gold standard}}$$

Precision and recall can be traded off against each other: increasing precision usually comes at a decrease in recall and vice-versa. It is common to combine the precision and recall scores into one figure that

DepBank:

```
obl(call~0, on~2)
stmt_type(call~0, declarative)
subj(call~0, ten~1)
tense(call~0, past)
number_type(ten~1, cardinal)
obl(ten~1, governor~35)
obj(on~2, justice~30)
obj(limit~7, abortion~15)
subj(limit~7, pro~21)
obj(reject~8, effort~10)
subj(reject~8, pro~27)
adegree(meanwhile~9, positive)
num(effort~10, pl)
xcomp(effort~10, limit~7)
```

GR:

```
(ncsubj called Ten _)
(ncsubj reject justices _)
(ncsubj limit efforts _)
(iobj called on)
(xcomp to called reject)
(dobj reject efforts)
(xmod to efforts limit)
(dobj limit abortions)
(dobj on justices)
(det justices the)
(ta bal governors meanwhile)
(ncmod poss governors nation)
(iobj Ten of)
(dobj of governors)
(det nation the)
```

FIGURE 2.7: The DepBank (King et al., 2003) and Briscoe and Carroll (2006) annotations for sentence (8). This figure is adapted from Briscoe and Carroll (2006).

balances between the two; this figure is known as the *F-score* . In PARSEVAL the F-score is defined as the harmonic mean of precision and recall as follows:

$$F = \frac{2PR}{P + R}$$

Aggregate scores are produced by summing these measures per sentence over all sentences in the test set. This produces a final combined score representing the overall performance of the parser in terms of constituent accuracy. The metric is very sensitive, and differences of a few tenths of a percentage point are typically significant.

Precision, recall, and F-score may be calculated in a *labeled* or *unlabeled* fashion. Unlabeled scores only consider the bracketing of the sentence compared to the gold standard. Labeled scores require the phrasal label of each constituent to match the equivalent label in the gold standard in addition to the bracketing. The more stringent labeled F-score is the standard number quoted as a measure of parser performance.

The Penn Treebank consists of 25 sections, numbered from 00 to 24. Sections 02-21 are conventionally used as parser training data, Section 00 as development data, and Section 23 as the held-out test set for PARSEVAL. The dominance of the Penn Treebank in the field means that calculation of a Penn Treebank-based parser's labelled PARSEVAL F-score over Section 23 allows for a direct comparison with over 15 years of research work. This provides a very strong impetus for parsers to produce Penn Treebank-compatible output that can be evaluated with PARSEVAL.

2.5 Parsing the Penn Treebank

Magerman (1995) describes SPATTER, the first parser trained upon the new data available from the Penn Treebank. SPATTER used decision tree models conditioned strongly on lexical information to produce constituent structure parse trees for individual sentences. Words are clustered with statistical techniques to provide the bases for decision tree predicates in the model. It outperformed all existing rule-based parsers on the PARSEVAL metrics, including some that had been under development for a decade. This result convincingly demonstrated that statistical parsing using large treebanks as training data was more viable and robust than knowledge engineering.

Magerman's work opened a new era of research work on statistical parsing, driven by performance on the PARSEVAL measure. Collins (1996) and Collins (1997) describe a parsing model that decomposes the construction of each constituent in a tree into a sequence of steps based on generating the head child first, followed by each other child constituent outward from the head. The generation of each child is assumed to be independent and is conditioned on all previously generated children. This approach is an extension of the process of parsing probabilistic context-free grammars; probabilities for the model are estimated from Penn Treebank trees. The parser became known as the Collins parser (Model 1) and it outperformed SPATTER in the PARSEVAL scores. Continued enhancements aimed at capturing more detailed linguistic phenomena such as subcategorisation and *wh*-movement in sentences resulted in Models 2 and 3 of the parser (Collins, 1999), which further improved performance. The Collins parser (particularly Model 2) has been widely used and particularly influential in the field. It continues to be one of the best-performing Penn Treebank parsers in the literature.

Charniak (1997) describes a probabilistic Penn Treebank parser that improved over SPATTER and the early Collins parser in PARSEVAL. Charniak's key improvement was the extraction of a context-free grammar as well as probabilities from the Penn Treebank, allowing a finer level of control over linguistic items generated by the parser. Charniak (2000) added a maximum-entropy inspired modelling framework (inspired by Ratnaparkhi (1997)'s work on linear-time parsing) and Charniak and Johnson (2005) proposed a reranking system of the top n-best parses for each sentence (covered further in Section 2.6). The Charniak parser as it is now known is another widely used system in the field; McClosky et al. (2006)'s work on reranking and self-training for the Charniak parser also make it the current state-of-the-art in parsing with a PARSEVAL labelled F-score of 92.1%.

2.6 Parser Reranking

Many parsers are capable of producing n-best output for a given sentence: a ranked list of n parses from the most probable to the least probable according to the parsing model. n-best parsing can be used to perform *parse reranking*: the process of taking an n-best list of parses for a sentence and reordering them based on an external model. Reranking is an increasing important technique in NLP as many tasks are naturally framed as ranking applications. It has been successfully applied to dependency parsing (Sangati et al., 2009), machine translation (Shen et al., 2004), and natural language generation with CCG (White and Rajkumar, 2009).

Parse reranking improves performance because the statistical models used in parsers usually rely on dynamic programming to discard equivalent or less probable derivations. This restricts possible features to a local context in each subtree of the derivation. Without dynamic programming, it is necessary to use pruning methods such as beam search to discard unlikely parse candidates. The exponential search space creates a trade-off between tractability and feature flexibility to the point where non-local features of subtrees are rarely included in models. This may cause problems when parsers make poor attachment choices due to the lack of greater context. In contrast, reranking operates over complete trees (the most probable derivations under the dynamic programming model), allowing arbitrary complex features of the parse to be incorporated without sacrificing efficiency. Rerankers can potentially disambiguate between parses based on more information than the parser's statistical model is able to capture.

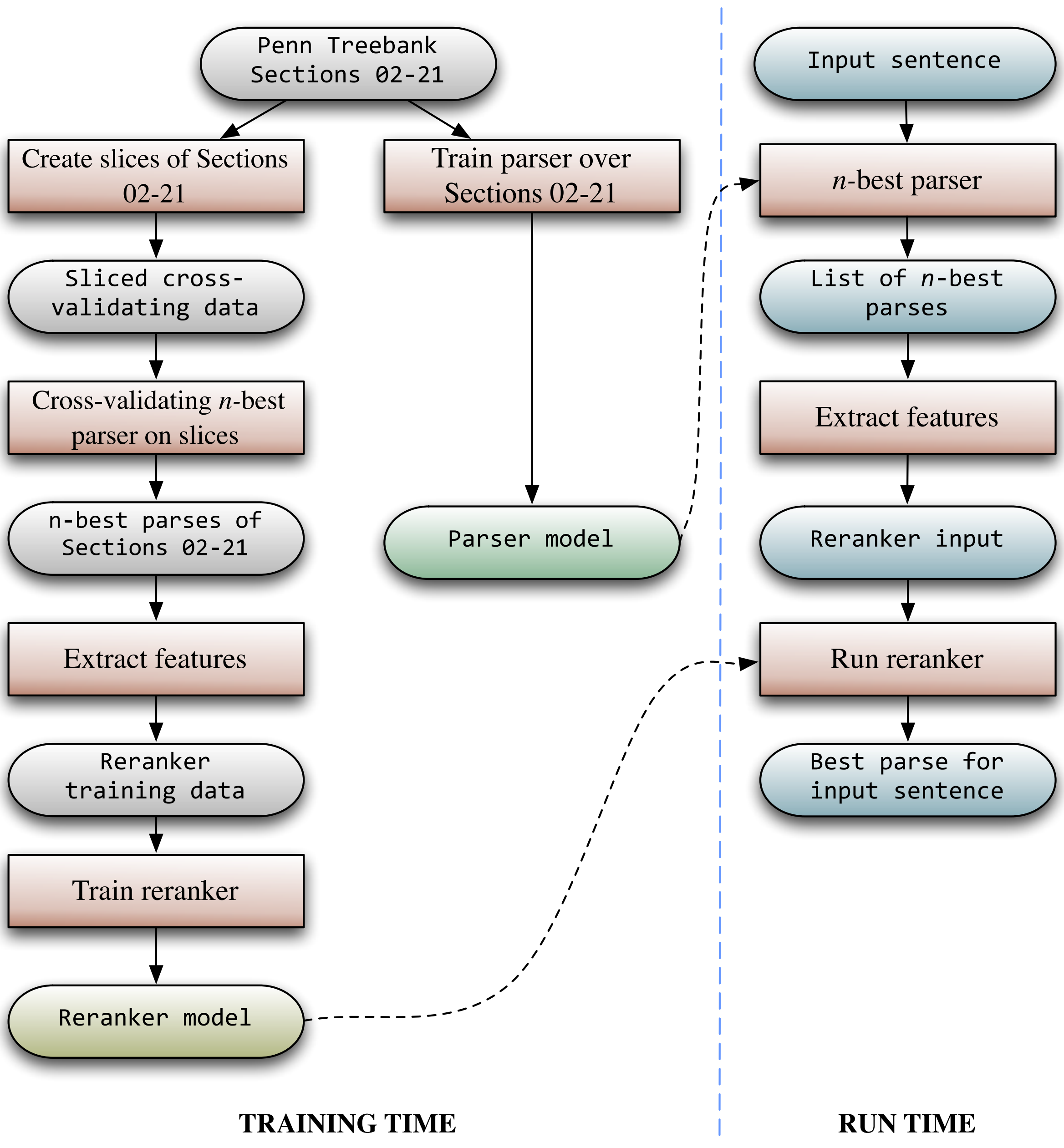
2.6.1 Components of a reranker

Collins (2000) introduces a supervised reranking process for the Collins parser and describes what has become the standard approach for generating the n-best parses used for reranker training. Sections 02-21 of the Penn Treebank are the standard data source; however, as these sections are also used for training the parser itself, care must be taken. Running the parser over previously seen sentences would mean that the confidence of the parser over the training data differs from its confidence over the (unseen) test data. The solution is an adaptation of the cross-validation evaluation approach. The entirety of Sections 02-21 are divided into consecutive slices. The parser is trained on all but one slice, and then it is used to parse the excluded slice to produce n-best output. This process is repeated for each slice, and the resulting parses concatenated to form the training data for the reranker. Producing reranker training data over the Penn Treebank trees has the advantage of allowing the accuracy of each parse via the PARSEVAL metric to be used as a natural optimization target.

In Collins (2000), a modified version of Collins Model 2 (Collins, 1999) was used as the base parser to generate reranker data. Dynamic programming in the parser was replaced with a large-width beam search as a simple way of retaining all possible analyses to produce n-best output. An average of 27 parses for 36,000 sentences from Sections 02-21 of the Penn Treebank were ultimately produced.

Features which capture various aspects of each parse are generated to represent them in the reranker. We showed in Section 2.2.1 how syntactic ambiguities correspond to multiple possible parse trees. In reranking the goal is to discriminate between the trees and resolve syntactic ambiguity; this can be achieved by using features which capture the topology of parses in the n-best list. In the Collins reranker the features included heads of constituents and the distances between them, context-free rules in the tree, consecutive bigrams and trigrams of words along with their ancestors in the tree, and parent-grandparent relationships (see Collins and Koo (2005) for a complete description). The final training data from the reranker consisted of these features extracted from the n-best list, with a simple count-based pruning scheme to retain a tractable number of features. The target value for each parse was its PARSEVAL score weighted by the size of the sentence in the gold standard.

Once a reranker model is trained, the reranker can be used to process new sentences. n-best parses of the new sentences are generated and features extracted from each. The reranker then uses its model to select an optimal parse to return. Collins reports a final PARSEVAL F-score of 89.75% using a boosting-based



Penn Treebank Sections 02-21
Create slices of Sections 02-21
Train parser over Sections 02-21
Sliced cross-validating data
Cross-validating *n*-best parser on slices
n-best parses of Sections 02-21
Parser model
Extract features
Reranker training data
Train reranker
Reranker model
Input sentence
n-best parser
List of *n*-best parses
Extract features
Reranker input
Run reranker
Best parse for input sentence
TRAINING TIME
RUN TIME

FIGURE 2.8: The reranking process from the generation of training data to parsing.

reranker, a 1.55% improvement compared to the baseline parser. This illustrates the general process of creating and running a parse reranker and is depicted in Figure 2.8.

The *oracle score* of an n-best list of parses is the maximum possible accuracy given a perfect reranker that always picks the best possible parse for each sentence. It serves as an *upper bound* on reranking performance – rerankers cannot possibly chose an analysis that is better than those present in the n-best list. The oracle score is a reflection of the quality of the n-best parses and represents the potential benefit from reranking.

Efficiently computing high-quality n-best parses is a significant challenge. Replacing dynamic programming with a beam search is prohibitively slow – the wide beam size necessary to produce an adequate number of parses results in a theoretically exponential time complexity. Huang and Chiang (2005) show how this simple method produces relatively low-quality n-best parses, and describe more efficient and accurate algorithms for n-best parsing based on a directed hypergraph analysis framework (Klein and Manning, 2001). By improving the quality of the candidate parses, Huang and Chiang demonstrate how oracle reranking scores can be dramatically improved compared to those reported in Collins (2000). These accurate algorithms provide an excellent framework for building parse rerankers.

n-best parsers have been developed for a number of grammatical formalisms. We will describe an n-best parser for CCG using the algorithms of Huang and Chiang (2005) in Chapter 3 (Brennan, 2008), and discuss our implementation of a parse reranking system based on this n-best parser in Chapters 5 and 7.

2.6.2 Charniak and Johnson's reranker

Charniak and Johnson (2005) describe n-best parsing and reranking for the Charniak parser. Rather than the algorithms described in Huang and Chiang (2005), a coarse-to-fine parsing approach is used which first produces a crude version of the parse, and then iteratively refines it using the most promising states. This allows n-best parses to be tractably computed while retaining dynamic programming in the parser. Unlike Collins' parser, which averages just 27 parses per sentence and yields only one analysis for some sentences, the Charniak n-best parser reliably returns up to 50 parses per sentence. The parses are also of high quality, with an oracle F-score of 96.8% on the PARSEVAL metric in 50-best mode – much higher than the 89.7% baseline. In comparison, the Collins n-best parser has an oracle score of 94.9% over a baseline of 88.2% (Huang and Chiang, 2005).

The training data for the Charniak reranker is generated in a similar fashion to Collins' reranker. A minor variation of the bracketed PARSEVAL score is used to measure how close each parse is to the gold standard in the training data. Charniak and Johnson propose an extended set of reranking features that include many of those used in Collins (2000). In addition, new features that attempt to model conventions of English such as parallelism in coordination and the shifting of larger constituents towards the end of sentences are used. We will provide an in-depth description of these features in Chapter 5.

The Charniak reranker produces a final F-score of 91.0% in 50-best mode. This is an improvement of 1.3% over the baseline model. Self-training over the reranked parses further improves performance to 92.1% F-score, which remains the state-of-the-art (McClosky et al., 2006). Self-training provides the additional benefit of improving the Charniak parser's performance on out-of-domain data – a known weakness of supervised parsing.

More recently, the Charniak reranking system has been adapted for the Berkeley parser (Petrov et al., 2006). Unlike the Collins and Charniak parsers, which are broadly similar and heavily based on lexicalised models, the Berkeley parser uses a split-merge technique to acquire a much smaller, unlexicalised grammar from its training data. Johnson and Ural (2010) report that reranking leads to negligible performance improvements for the Berkeley parser, and acknowledge that the reranker's feature set, adapted from Charniak and Johnson (2005), may be implicitly tailored to the Charniak parser over the Berkeley parser. In particular, the feature pruning process for reranking was conducted over output from the Charniak parser, which may have prevented useful features for the Berkeley parser from being chosen. A core contribution of this work explored in Chapters 5, 6, and 7 is the parser-dependence of reranking: whether reranking requires careful tuning to a parser before it can improve performance.

2.7 The Annotation Bottleneck

The current state-of-the-art PARSEVAL score of 92.1% seems to suggest that parsing may soon be a 'solved' problem. This is not the case for several reasons, not least of which is a number of criticisms of the PARSEVAL metric (see Chapter 4). Another issue that impedes statistical parsing and many other areas of computational linguistics is the *annotation bottleneck*, which remains an open problem in the field.

The shift from knowledge engineering to statistical methods in parsing moved the burden on linguists from defining grammars and word rules for complex languages to annotating sentences with a gold-standard structure. Annotated data remains the cornerstone of the most accurate parsers, creating a strong dependency on corpora.

Increasingly large amounts of text are available from areas as diverse as newswire texts, encyclopaedia articles, and scientific literature. It is reasonable to assume that practical parsers will need to process text from a variety of different domains. Unfortunately, Sekine (1997) demonstrated how supervised parsers suffer from the problem of *domain dependence*. This problem manifests when a parser trained over data from one domain (e.g. newswire text) performs poorly when analysing data from a different domain (e.g. scientific literature). The result implies that evaluation metrics which test over in-domain data may provide overly optimistic views of real-world parser performance.

Gildea (2001) provided strong evidence suggesting that accurate supervised parsing requires annotated corpora in the domain of interest. This means that large annotated corpora of at least the size of the Penn Treebank must be developed in every domain for effective cross-domain language processing. Unfortunately, the Penn Treebank project itself was highly labour-intensive, requiring several years to produce just 1.1 million words of annotated newswire text. Additionally, the annotation is relatively skeletal (see Section 2.3.1); more detailed annotation would require more time and linguistic expertise. The amount of raw text we have access to exceeds the size of the Penn Treebank by several orders of magnitude, e.g. Google has released a corpus of unannotated n-gram counts of one trillion words of web text (Brants and Franz, 2006). Manual annotation is simply too slow and inefficient for producing enormous quantities of highly-detailed annotated text – but it is exactly this scale of data which is necessary for building the most accurate parsers.

As a static corpus, the Penn Treebank also suffers from problems of obsolescence. The text itself is sourced from the 1989 Wall Street Journal and contains mainly news reports on the financial and business sectors. Language use has changed in the intervening period; for example, vocabulary change has introduced email addresses and URL s into everyday language, whereas neither of these are prevalent in the Penn Treebank. This outdatedness potentially limits the utility of the Treebank in parsing even current in-domain newswire text. Corpora will continually age and require modernisation or replacement to remain up-to-date, and manual annotation is again too slow to fulfill this requirement. The constraint of manual annotation is referred to as the *annotation bottleneck*.

The annotation bottleneck prevents the deployment of parsing systems across a wide variety of domains. Maintaining ageing annotated corpora is prohibitive, as is the creation of corpora based on web-scale data. A clear and imminent challenge in NLP is addressing the bottleneck and removing the constraints on the field that it causes.

2.8 Summary

In this chapter, we have given a broad background for the field of parsing. We have described different kinds of grammars and how they are used to analyse sentences and discussed important corpora and parsers in the field. We have introduced parser rerankers and described the framework used to build them. Finally, we have introduced the core challenge of the annotation bottleneck and how it constrains the continued development of parsing on a wide-scale.

In the next chapter, we will describe the Combinatory Categorial Grammar formalism used by the work covered in this thesis, and how it relates to the general background presented here.

CHAPTER 3

Combinatory Categorial Grammar

In this chapter we introduce the Combinatory Categorial Grammar (CCG) formalism, which is the basis for the parser used in this thesis.

3.1 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a linguistically expressive grammar formalism based on formal logic. CCG is *lexicalised* in that the grammar for each language is directly encoded in the lexicon. The basic lexical item in CCG is the *category*, which completely defines the syntactic behaviour of a word in the context of a sentence. The formalism is mildly context-sensitive and incorporates both constituent structure and dependency information in its analyses.

3.1.1 Categories in CCG

Atomic CCG categories represent constituents that are syntactically complete, including S (sentence), N (noun), NP (noun phrase), and PP (prepositional phrase). A number of other atomic categories indicate punctuation marks and coordinators (which are labelled as $conj$). Additionally, atomic categories may carry features that provide additional linguistic information. For example, declarative clauses are labelled $S[dcl]$ while passive clauses are labelled $S[pss]$. Noun phrases beginning with a determiner are labelled $NP[nb]$ to indicate that they are *non-bare*.

Complex categories represent constituents that are incomplete and require additional arguments to form grammatical constructions. They are constructed recursively from other categories and take the form $X\backslash Y$ or X/Y, where X and Y may be atomic or complex. X is the result category produced when the category is combined with its argument Y. The backward and forward slashes specify that Y is expected to the left or right of the category respectively. Multiple arguments can be specified by returning nested complex categories. For example, transitive English verbs such as ***baked*** in sentence (9) require a subject (something to do the baking) and an object (something that was baked) in order to form a grammatical sentence. English word order dictates that subjects should be to the left of the verb and objects to the right, and this structure can be represented by the category $(S[dcl]\backslash NP)/NP$. This indicates that the verb requires one noun phrase to the right, and produces a construction that requires one noun phrase to its left to form a declarative clause.

(9) Jack baked a cake with raisins.

3.1.2 Combinatory Rules

Constituent structure grammars generate rules to combine constituents. There is no syntactic information encoded in each constituent, meaning that many hundreds of partially redundant rules may be necessary to capture the entire spectrum of behaviour in a language – for example, intransitive, transitive, and ditransitive verbs all require separate rules even though all require a subject argument to their left.

The lexicalisation of CCG means that the categories assigned to words are combined together with just a few generic *combinatory rules* to form an analysis. The two most basic combinatory rules, which come from the original Categorial Grammar (Bar-Hillel, 1953), are two *function application* rules specifying that a constituent with a complex category can be combined with the outermost of its nested arguments to form its result category. These are termed *forward* ($>$) and *backward* ($<$) *application* and can be denoted as follows:

$$X/Y \quad Y \quad \Rightarrow \quad X \quad (>)$$
$$Y \quad X\backslash Y \quad \Rightarrow \quad X \quad (<)$$

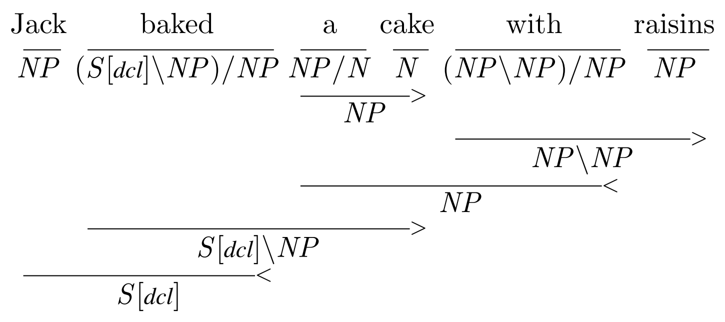


FIGURE 3.1: A simple CCG derivation.

A *derivation* of sentence (9) using these rules is given in Figure 3.1. It can be seen that at each stage of the derivation one of the function application rules is used to combine two categories together to form a result category, which is then available for further combination. While CCG derivations are conventionally written with the sentence at the top, they are quite similar to phrase structure trees. Each category in the CCG analysis corresponds to a node in a tree that is at most binary branching due to the single result/argument structure of each category. Figure 3.2 illustrates the equivalent tree for the derivation in Figure 3.1.

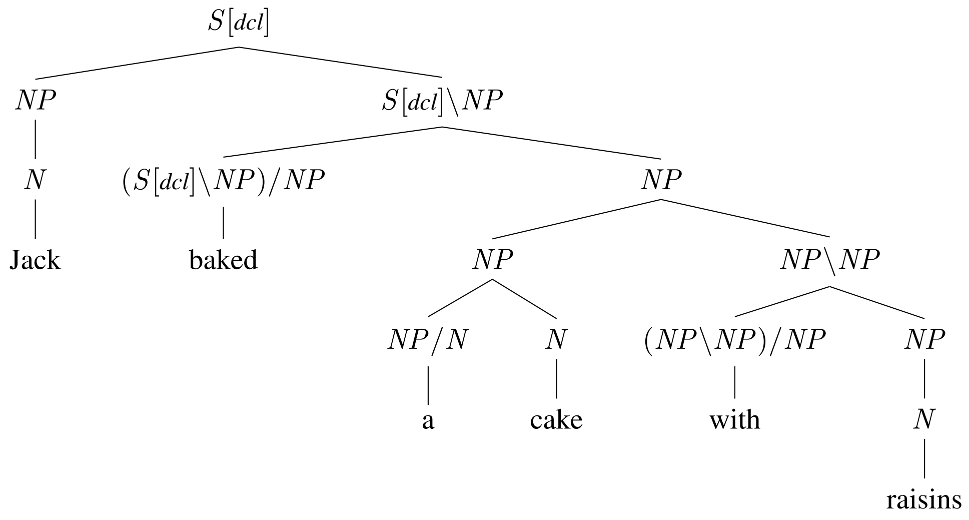


FIGURE 3.2: The corresponding tree for the derivation in Figure 3.1.

In addition to forward and backward application, CCG includes a number of other combinatory rules that increase the expressive power of the grammar from context-free to mildly context-sensitive (Vijay-Shanker and Weir, 1994). Steedman (2000) shows how these combinators can be used together to account for a diverse range of linguistic phenomena in English without overgeneralisation. These rules include:

Forward composition

$$X/Y \;\; Y/Z \;\Rightarrow\; X/Z \quad (>\mathbf{B})$$

Backward composition

$$Y\backslash Z \;\; X\backslash Y \;\Rightarrow\; X\backslash Z \quad (<\mathbf{B})$$

Forward crossed composition

$$X/Y \;\; Y\backslash Z \;\Rightarrow\; X\backslash Z \quad (>\mathbf{B}_X)$$

Backward crossed composition

$$Y/Z \;\; X\backslash Y \;\; \Rightarrow \;\; X/Z \quad (<\mathbf{B}_X)$$

Coordination

$$X \;\; conj \;\; X \Rightarrow X \quad (<\mathbf{\Phi}>)$$

Type-raising

$$X \;\; \Rightarrow \;\; \mathrm{T}/(\mathrm{T}\backslash X) \quad (>\mathbf{T})$$

$$X \;\; \Rightarrow \;\; \mathrm{T}\backslash(\mathrm{T}/X) \quad (<\mathbf{T})$$

The T used in the type-raising rules may be any CCG category (subject to language-dependent restrictions), while the ternary coordination rule is used where constituents are linked together with a coordinator such as *and* or *or*. In addition, Steedman (2000) bans the use of the forward composition combinator in English. An example illustrating the use of some of these rules is presented in Figure 3.3.

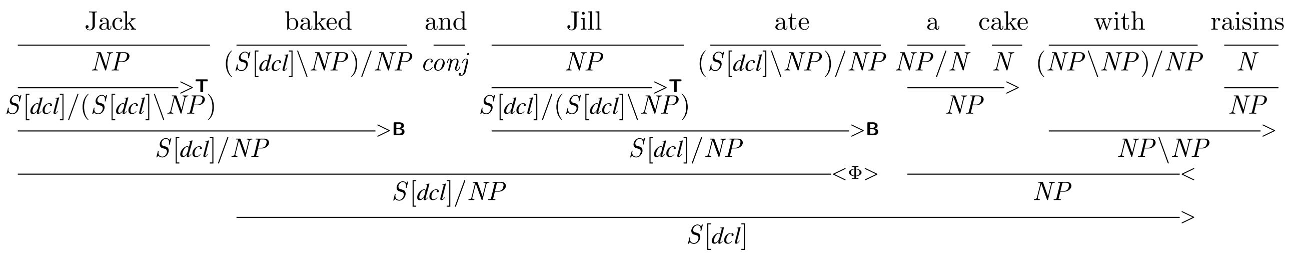


FIGURE 3.3: A CCG derivation using type-raising, forward composition, and coordination.

3.1.3 Spurious Ambiguity

The expressiveness afforded by these combinatory rules leads to what has been termed 'spurious ambiguity' (Wittenburg, 1986), whereby even simple sentences can have several analyses which yield identical dependencies (or logical representations). Such analyses are thus semantically equivalent to each other, and one such alternative analysis for the sentence in Figure 3.1 is presented in Figure 3.4. In particular, the type-raising and composition combinators allow the same categories to be combined in different ways.

The problem of spurious ambiguity has led to criticisms that efficient parsing with CCG is not possible. However, several solutions to the problem have been proposed that partially or wholly restrict the

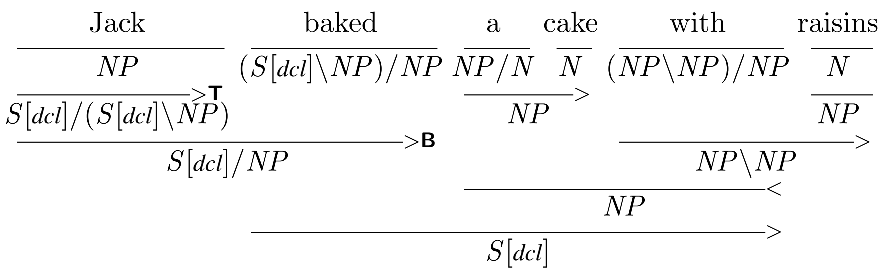


FIGURE 3.4: An alternate but semantically equivalent ‘spurious’ derivation for sentence (9) using type raising and composition.

combinatoric explosion of derivations. Wittenburg (1987) suggested replacing the type-raising and composition combinators with ‘predictive’ forms that are only used when necessary. Eisner (1996) noted that this proposal introduced non-standard constituents rejected by traditional CCG, and instead described a technique that defined a single *normal-form derivation* for each semantically equivalent analysis (termed an *equivalence class*). This technique completely eliminates ambiguity when the type-raising combinator is not used, and significantly reduces the space of derivations when it is used (Clark and Curran, 2004a).

3.1.4 Dependencies in CCG

Category-level annotations are used to represent head and dependency information in CCG. As discussed in Section 2.1.1, each constituent in a sentence contains a single word which can be identified as the head. Figure 3.5 gives the CCG derivation of sentence (9) with the head of each constituent annotated in subscript on each category.

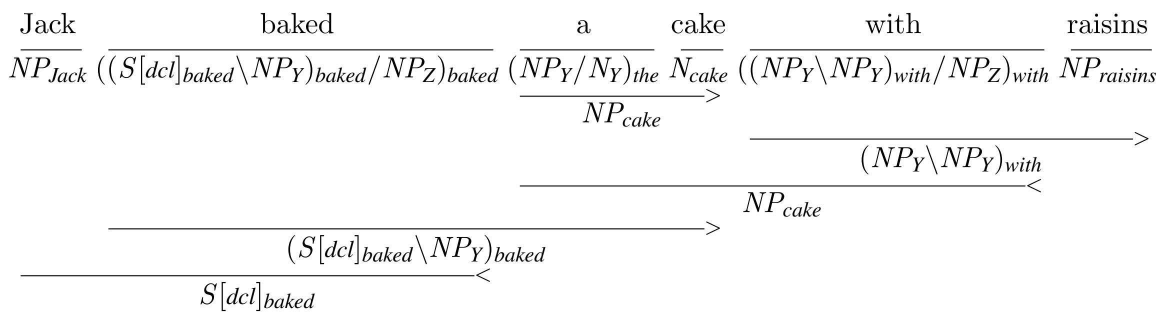

Jack baked a cake with raisins
NP_{Jack}
$((S[dcl]_{baked}\backslash NP_Y)_{baked}/NP_Z)_{baked}$
$(NP_Y/N_Y)_{the}$
N_{cake}
$((NP_Y\backslash NP_Y)_{with}/NP_Z)_{with}$
$NP_{raisins}$
NP_{cake}
$(NP_Y\backslash NP_Y)_{with}$
NP_{cake}
$(S[dcl]_{baked}\backslash NP_Y)_{baked}$
$S[dcl]_{baked}$

FIGURE 3.5: CCG analysis of sentence (9) with head annotations.

At the word level, each word is its own head. For words that have complex categories, the head of the result is indicated using variables on the subparts of each argument category. For example, the word *the* bears the category $(NP_Y/N_Y)_{the}$. This indicates that the word is its own head, and that it combines with a word to the right with category N and head Y to product a constituent with category NP and head Y, e.g. the head of *the cake* is *cake*.

Dependencies in CCG are created by complex categories. Each argument slot in a complex category is numbered and associated with a corresponding head variable. The dependencies themselves are 5-tuples $\langle h_f, f, s, h_a, l\rangle$. In this notation, f is the CCG category which creates the dependency. h_f is the word to which that category was assigned. s specifies which slot is being filled, and h_a is the head word of the constituent filling slot s; h_a will fill the variable that is co-indexed with slot s. Finally, l may contain either $(-)$ or another category, indicating whether the dependency is *local* or *long-range*. Long-range dependencies occur when the structures that are related are separated by intervening constituents. If the dependency is long-range, then l contains the CCG category via which the dependency was formed.

$$\textit{baked} := (S[dcl]_{baked}\backslash NP_{Y,1})/NP_{Z,2}$$

$$\langle \textit{baked},\ (S[dcl]\backslash NP_1)/NP_2,\ 1,\ \textit{Jack},\ -\rangle$$
$$\langle \textit{baked},\ (S[dcl]\backslash NP_1)/NP_2,\ 2,\ \textit{cake},\ -\rangle$$

FIGURE 3.6: Argument slot numbering and dependencies for the word *baked* in sentence (9).

A complete example follows. The transitive verb *baked* requires a subject noun phrase argument and an object noun phrase argument, and hence its fully annotated category is $(S_{baked}\backslash NP_{Y,1})/NP_{Z,2})_{baked}$. This category indicates that argument slot 1 is associated with variable Y on the subject NP, and so when the slot is filled a dependency will be established between *baked* and the word that is assigned to Y. In Figure 3.5, argument slot 1 of *baked* is filled by *Jack* and argument slot 2 is filled by *cake*, creating dependencies between the words that are similar to the `subj` and `obj` relations in a dependency grammar. Numbered argument slots and the full dependency tuples created by *baked* are shown in Figure 3.6.

3.2 CCGbank

CCGbank is the primary corpus used for English parsing with CCG. Derived from the Wall Street Journal section of the Penn Treebank through a semi-automated process, CCGbank contains 99.4% of the sentences from the Penn Treebank annotated with CCG normal-form derivations.

Hockenmaier (2003) describes in detail the process used to transform the Penn Treebank into CCGbank, and notes that many aspects of the Penn Treebank annotation require careful treatment in the resulting conversion. For example, rather than the ternary coordination rule presented in Section 3.1.2, CCGbank represents coordination with a set of binary rules as follows:

$$\begin{aligned} conj\ X &\Rightarrow X[conj] \\ ,\ X &\Rightarrow X[conj] \\ X\ X[conj] &\Rightarrow X \end{aligned}$$

This has the advantage of ensuring that all trees in CCGbank are at most binary-branching, rather than including ternary trees where coordination is present.

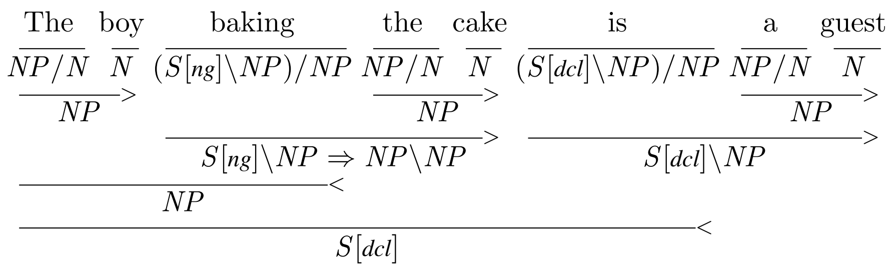


FIGURE 3.7: Unary type-changing of a verb phrase ($S\backslash NP$) into a post-noun modifier ($NP\backslash NP$). Without the type-change, the verb *baking* would require a different category.

CCGbank also includes a number of additional unary rules that are non-standard in CCG to deal with a variety of linguistic phenomena found in the Penn Treebank. These include the conversion of a bare noun N into a noun phrase NP and various punctuation absorption rules. Finally, there are a number of unary type-changing rules that allow several kinds of verb phrase constituents to become modifiers, which are constituents that produce a result category that is the same as the argument category (e.g. $\underline{(S\backslash NP)}\backslash\underline{(S\backslash NP)}$). These constructions are called *clausal adjuncts* and are difficult to encode in CCG due to the different syntactic behaviour of the verb phrase compared to the clausal adjunct. The list of unary type-changing rules used in CCGbank is as follows:

$$\begin{array}{rcl} N & \Rightarrow & NP \\ S[\mathit{pss}]\backslash NP & \Rightarrow & NP\backslash NP \\ S[\mathit{adj}]\backslash NP & \Rightarrow & NP\backslash NP \\ S[\mathit{ng}]\backslash NP & \Rightarrow & NP\backslash NP \\ S[\mathit{ng}]\backslash NP & \Rightarrow & (S\backslash NP)\backslash(S\backslash NP) \\ S[\mathit{dcl}]/NP & \Rightarrow & NP\backslash NP \end{array}$$

Figure 3.7 demonstrates an example derivation featuring a unary rule application. The size of the category set would explode without these type-changing rules as all verbs that could be used as modifiers would require a different category (Hockenmaier and Steedman, 2007).

The section numbers of CCGbank are equivalent to those in the Penn Treebank, and so the standard split of data remains the same – Section 00 for development, Sections 02-21 for training, and Section 23 for testing. However, due to fundamental differences between the structure of trees in CCGbank and those in the Penn Treebank, the PARSEVAL scores for a CCG parser are not directly equivalent to those of a Penn Treebank parser (see Section 4.2). Instead, the standard CCG evaluation is precision, recall, and F-score over *labeled dependency recovery*, similar to the calculations discussed in Section 4.1.

As CCGbank is based on the Penn Treebank, it inherits some problems from the older corpus. Particularly problematic is the lack of noun phrase structure (see Section 2.3.1). While the Penn Treebank annotates noun phrases as flat constructions with flat branching, the binary branching nature of CCG means that this cannot be directly emulated in CCGbank. Instead, all noun phrases in CCGbank are annotated as being exclusively right-branching. This creates incorrect analyses for left-branching noun phrases and has unwelcome side-effects when combined with coordination within the phrase; the following nonstandard rule is included in CCGbank to handle these cases as depicted in Figure 3.8.

$$conj\ \ N \ \Rightarrow\ N$$

The improvements in noun phrase annotation from Vadas and Curran (2007)'s reannotated version of the Penn Treebank were transferred to CCGbank in Vadas and Curran (2008). This allows parsers trained on CCGbank to more accurately model noun phrase structure.

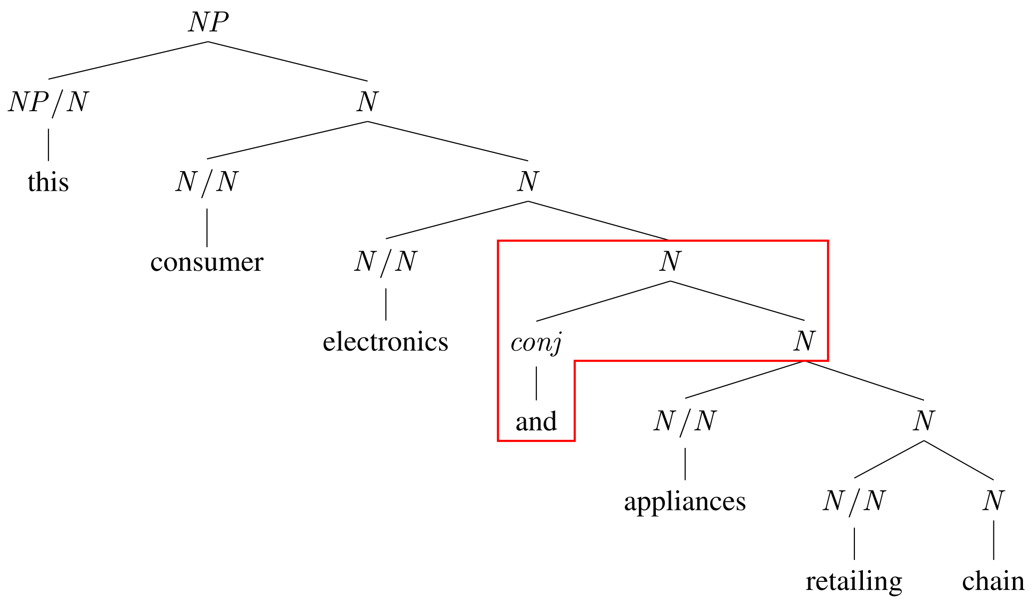


FIGURE 3.8: A noun phrase derivation from CCGbank taken from Hockenmaier (2003). The assumption of right-branching has created an awkward and incorrect structure when combined with coordination. The highlighted rule application is nonstandard in CCG but required to analyse this fragment.

3.3 The C&C parser

The C&C parser (Clark and Curran, 2007b) is a state-of-the-art parser based on the CCG formalism and trained on CCGbank. The parser is highly efficient and can analyse sentences an order of magnitude faster than other linguistically-motivated parsers, demonstrating that spurious ambiguity is not a problem for practical CCG parsing.

Parsing with CCG is a two-step process: first, the lexical categories are assigned to each word in the sentence, and then the parser combines the categories together to form a spanning analysis. In the C&C parser, a *supertagger* performs the first step, while the parser itself takes the tags produced by the supertagger to form an analysis. The division of labor between these steps gives the C&C parser its combination of fast and accurate parsing.

The supertagging process has been described as ‘almost parsing’ (Bangalore and Joshi, 1999) since the lexical categories assigned by the supertagger encode detailed syntactic information. With perfect categories, the parser only has to determine how to combine them to form a correct analysis. The C&C supertagger considers a restricted set of categories determined by a frequency cut-off of 10 in Sections

02-21 of CCGbank. The frequency restriction results in a set of 425 possible categories that has wide coverage on unseen text (Clark and Curran, 2004b).

State-of-the-art supertagging accuracy is around 92% on Section 00 of CCGbank (Curran et al., 2006). Unfortunately, sentence accuracy (where all words in a sentence are correctly tagged) is only 36.8%, which is too low for parsing – it is likely that the parser will be unable to find any analysis for the sentence given so many incorrect categories. To address this problem a multi-tagger that can assign multiple categories to each word is used; the tagger returns all categories within some factor β of the highest probability category for the word. Using the multi-tagger per-word accuracy increases to 98.5% and sentence accuracy to 78.4% using all categories within 1% of the most probable category. On average this assigns 2.9 categories per word. (Clark and Curran, 2004b).

During parsing, the supertagger is initially run with a large β level, assigning a highly restricted set of categories to each word. The parser attempts to find an analysis using this small set of categories. If an analysis cannot be found the supertagger is run again with a smaller, less restrictive β value to get more categories per word. The parser then tries to find an analysis again using this larger category set. This process continues for a number of iterations until a spanning analysis is found, or else the parser gives up and moves on to the next sentence. Clark and Curran (2007b) show that this tight integration between parser and supertagger results in very efficient parsing with wide coverage and high accuracy.

3.3.1 Chart Parsing with the CKY algorithm

The Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967) is a dynamic programming algorithm that can be used to parse a context-free constituent structure grammar. It is called a *chart parsing algorithm* because it relies on a triangular data structure called a *chart* to memoise possible analyses. The chart allows all possible derivations of a sentence to be efficiently stored and ranked during the parsing process, eliminating the need for backtracking.

The C&C parser uses the CKY algorithm as described in Steedman (2000) to build a packed chart, where equivalent constituents are grouped together and considered collectively.

In the chart, the horizontal indices along the bottom are called the *position* (*pos* for short), and indicate the beginning of a constituent. The vertical indices along the side are called the *span* and indicate the length of a constituent. Each block in the chart is called a *cell*. A cell with pos i and span j covers the tokens $[i, i+j)$ in the sentence. For example, the cell marked with a diamond in Figure 3.9 spans over

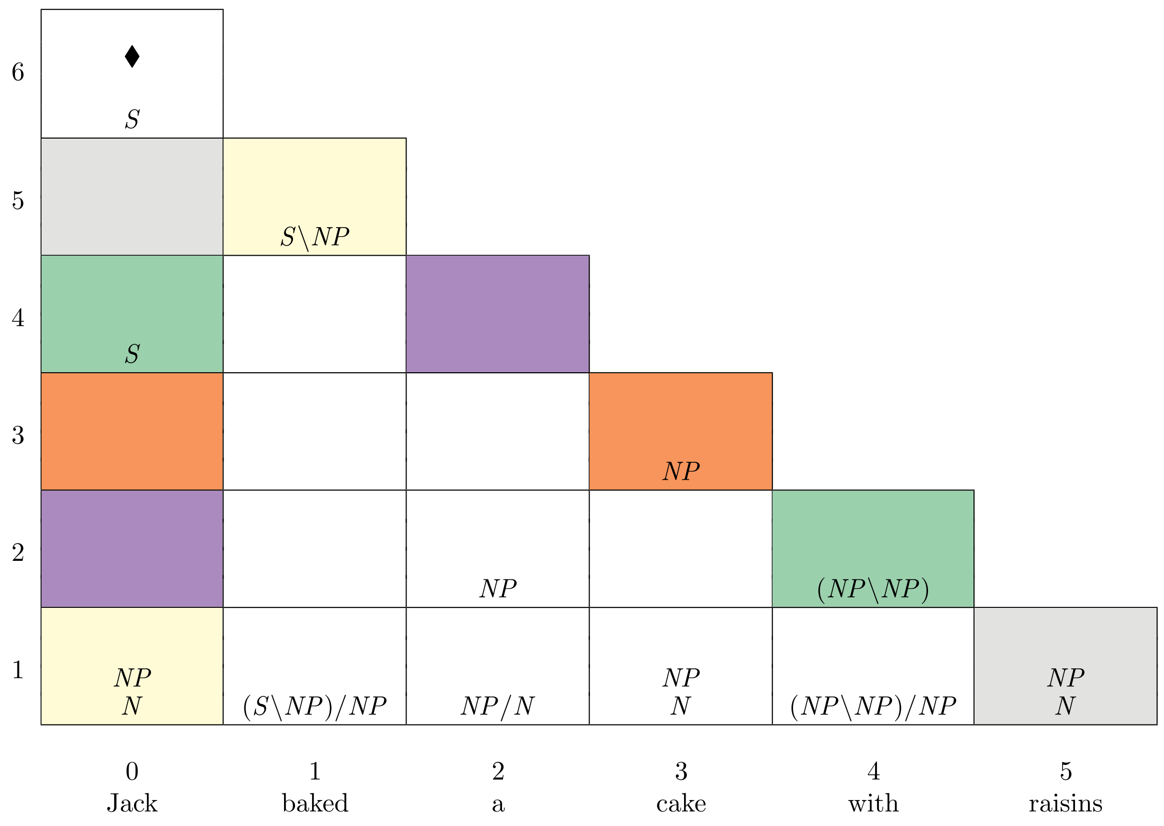


FIGURE 3.9: A partially filled CCG chart. Cells are colour-coded to indicate the various combinations considered for the top cell – those that span the entire sentence.

the entire sentence as it begins at position 0 with span 6. Conversely, the cells at span 1 cover only each individual token.

Each cell in the chart contains all possible analyses for the tokens that it covers. When a cell (i, j) is built, the contents of all pairs of non-overlapping cells that together span $[i, i + j)$ are considered. The use of dynamic programming ensures that each cell is only built once and memoised for further use; this also ensures that considering pairs of non-overlapping cells is sufficient for enumerating all possible analyses. In Figure 3.9, the pairs of colour-coded cells are those which are considered when building the cell marked with a diamond – that is, these combinations of cells begin at position 0 and span the entire sentence.

The chart parsing process for CCG proceeds bottom-up from spans of length 1 up to the length of the sentence. The categories from the supertagger are assigned to the bottom row of cells in the chart (i.e. the cells with span 1). Then, for each increasing span length j and position i, all possible pairs of cells that can cover the span $[i, i + j)$ are considered by the parser. Each pair of cells that can be combined with one of CCG's combinatory rules has its result placed in the cell (i, j), which also contains links back to the combined cells for the purposes of enumerating the derivation. Unary rules are also applied at each step where valid.

When the chart is completed, all valid derivations over the sentence will be found in the topmost cell. If a spanning analysis is not found (i.e. there is no S category in the topmost cell), then the supertagger is rerun with a more permissive β level and the process is repeated up to 3 more times. If the parser still cannot find an analysis then it is discarded as being unparseable.

Once the chart is filled, it is then *decoded* according to a statistical model to find the highest scoring derivation. Clark and Curran (2007b) describe a number of discriminative maximum entropy parsing models for the C&C parser, including a model based on CCG dependency structures and a model based on the normal-form derivations in CCGbank. The dependency model involves summing the probabilities of all possible derivations that yield a particular dependency structure, including the nonstandard derivations eliminated by normal-form constraints. The normal-form model scores normal-form derivations more highly than the alternate derivations. In this thesis, all experiments use the normal-form model.

3.4 The n-best C&C parser

Brennan (2008) describes the recent development of an n-best version of the C&C parser that can produce a ranked list of analyses for a sentence. The n-best C&C parser incorporates the algorithms described in Huang and Chiang (2005), and is almost as efficient as the baseline 1-best parser. This version of the C&C parser is used as the basis for the CCG reranker developed and extended in Chapters 5 and 7.

We mentioned in Section 2.6.1 how reranking is dependent on high-quality parses from the n-best parser. Poor n-best output means that a reranker cannot possibly improve performance. Table 3.1 shows that the oracle F-score of the n-best C&C parser given a perfect reranker is 91.43% over 10-best parses and 92.48% over 50-best parses. This is a substantial improvement over the baseline result of 86.75% F-score and provides an excellent basis for a reranker.

The C&C oracle score falls notably short of the 50-best oracle of 96.8% reported by Charniak and Johnson (2005), over a baseline of 89.7%. However, Charniak's numbers refer to the PARSEVAL score

	LP	LR	LF	AF
Baseline	87.19	86.32	86.75	84.80
Oracle 10-best	**91.98**	**90.89**	**91.43**	**89.47**
Oracle 50-best	**93.43**	**92.26**	**92.84**	**90.96**

TABLE 3.1: Baseline and oracle n-best parser performance over Section 00 of CCGbank.

for constituency parses, so they are not directly comparable to our more stringent dependency recovery metric.

3.5 Summary

In this chapter, we described Combinatory Categorial Grammar, a lexicalised grammar formalism that incorporates both constituency and dependency information in its analyses. We described CCGbank, a corpus of CCG derivations and dependencies derived from the Penn Treebank, as well as a fast, accurate, and state-of-the-art parser (C&C) that is trained on CCGbank. We introduced the n-best version of the C&C parser as well as its oracle performance; this parser will be used for our development of a CCG reranker in Chapters 5 and 7.

CHAPTER 4

Evaluation

In this chapter, we describe a number of other evaluation metrics for parsers, including dependency-style evaluation, CCG evaluation, and evaluation using textual entailments. We discuss how difficulties in cross-formalism comparison and problems with the field-standard PARSEVAL metric have led to the development of these other measures.

The first major contribution of this thesis is the implementation of a new evaluation system for the C&C parser based on textual entailment prediction. This work was published at the 5th International Workshop on Semantic Evaluation in Uppsala, Sweden under the title SCHWA*: PETE using CCG Dependencies with the C&C Parser* (Ng et al., 2010).

4.1 Dependency Evaluation

The PARSEVAL metric (see Section 2.4) has dominated parser evaluation for over two decades. However, the metric has not been without criticism. Since PARSEVAL only accounts for an exact phrasal match between the computed parse and the gold standard, the measure unfairly punishes certain kinds of simple attachment errors (Lin, 1998). Additionally, the single score is an aggregate over all types of constituents; evaluations which examine individual types of constituents show much lower accuracies on some semantically important constructions such as coordination and prepositional phrase attachment (Collins, 1999; Rimell et al., 2009). These problems have led to calls for more rigorous evaluation measures for parsing.

In many cases, the constituent structure tree is less informative than the set of dependencies in a sentence. This is because the underlying dependencies more closely model the semantics of the sentence; we saw in Section 2.1 how an underlying similarity can be present between sentences with different constituent structures. This has influenced the development of metrics calculated over lists of dependencies, originating with Lin (1995)'s description of how a dependency-based evaluation can avoid penalising inconsequential differences between the gold standard and parser output whilst providing a fairer comparison of parser performance.

Other proposed evaluation measures for parsers include *application-based evaluation*, where parsers are evaluated based on how they contribute to an external system. This type of evaluation targets the actual impact that parsers have in practical applications, but systems are often tailored to one particular formalism, and converting them to accept various kinds of analyses under different formalisms is often too difficult for this evaluation to be practical.

4.2 CCG Evaluation

The PARSEVAL metric is particularly uninformative when applied to CCG parsers. Penn Treebank trees are usually very flat, with the longest non-terminal in the Treebank having 51 children. In comparison, the binary-branching nature of CCG creates trees that are naturally deep, with only two children per non-terminal. This difference makes a direct comparison against the constituency structure of the Penn Treebank difficult as the deeper CCG trees contain substantially more brackets. Additionally, the spurious ambiguity of CCG can lead to derivations with dramatically different tree structures than the gold standard that are still equally correct. This leads to a poor score from the PARSEVAL metric; however, as these derivations still license the same semantics, they are equally valid and should not be penalised.

To counter these problems the standard evaluation for CCG parses is dependency recovery over a held-out test set from CCGbank. This involves a comparison of the dependencies in the parser output for a test sentence against the dependencies present for the sentence in CCGbank. *Labeled* dependencies take into account the category containing the dependency relation, the argument slot, the word associated with the category, and the head word of the argument (see Section 3.1.4 for definitions of these items). All four elements must match the gold standard to be scored. *Unlabeled* dependency evaluation only requires the head and argument in the dependency relationship to match, though we choose to focus on the more rigorous labeled score. The calculation of precision, recall, and F-score for dependency recovery is described in Clark et al. (2002) as follows:

$$P = \frac{\text{\# dependencies correctly recovered by the parser}}{\text{\# dependencies in the parser output}}$$

$$R = \frac{\text{\# dependencies correctly recovered by the parser}}{\text{\# dependencies in the gold standard}}$$

$$F = \frac{2PR}{P + R}$$

There are a number of other, less important measures of CCG parser performance. Sentence accuracy is the percentage of sentences whose dependencies are entirely correct. Category accuracy is the percentage of categories which are correct. Coverage is the percentage of sentences in the test set which receive an analysis, regardless of the accuracy of the parse.

4.2.1 Evaluation on DepBank

Clark and Curran (2007a) present an evaluation of the C&C parser using DepBank GRs. The DepBank evaluation is achieved by converting the native CCG output of the C&C parser to GRs using a number of rules and heuristics. The resulting conversion is a many-to-many mapping between CCG dependencies and GRs developed by comparing gold-standard dependencies in CCGbank Section 23 with those in DepBank. The metrics of precision, recall, and F-score are calculated in a similar way to the CCG dependency recovery score presented in Section 4.2.

Developing the GRs mapping is difficult and lossy Clark and Curran (2007a). Post-processing heuristics were necessary to improve the accuracy of the mapping scheme, resulting in an upper performance bound for the C&C parser of 84.8% in this evaluation. This relatively low bound illustrates the difficulty of cross-formalism comparison, which we will describe in more detail in the next section.

4.3 Deficiencies in Parser Evaluation

While Cahill et al. (2008) notes that more meaningful parser evaluation is possible with dependency representations instead of PARSEVAL scores, the evaluations are still imperfect. Even allowing for the more general approach pursued in dependency representations, there is little consensus on an ideal set of dependencies to use, and a number of schemes other than Briscoe and Carroll GRs have emerged. The traditional procedure of using a held-out test set for PARSEVAL and dependency evaluation is reliant on expert linguistic annotation to create the gold-standard, rendering it subject to the annotation bottleneck (see Section 2.7). Larger scale evaluation across different domains requires more annotated data that cannot be produced by non-linguists, making it prohibitively expensive and time-consuming.

Cross-formalism parser comparison presents additional challenges. The Penn Treebank annotation and CCG are just two examples of grammar formalisms used to build parsers; there are many more employed in the field and each formalism specifies different and often incompatible output. Obviously, downstream users would find it useful to fairly compare all of these parsers in order to choose the best one for a particular application. Unfortunately, there is scant agreement on an ideal representation of a parse, making accurate cross-formalism parser comparison very difficult.

Most approaches to cross-formalism evaluation attempt to convert the output of one parser to another through a mapping scheme, and then compare the two parsers against the established test data and gold-standard. Such conversions are lossy and create artificial upper bounds on performance. For example, the difficult conversion of CCG dependencies to GRs described in Section 4.2.1 restricts the C&C parser to an upper performance bound of 84.8% F-score on the DepBank evaluation. The conversion of CCG output from the C&C parser back to Penn Treebank trees in Clark and Curran (2009) is also challenging, and results in an upper bound of less than 95% F-score on the PARSEVAL metric. Since performance improvements of a few tenths of a percentage point are considered significant in the parsing literature, even this upper bound guarantees inequality in cross-formalism comparison. Clearly, a new approach to evaluation is needed to adequately address these issues and provide a fair comparison between different formalisms.

4.4 Parser Evaluation using Textual Entailment

The International Workshop on Semantic Evaluation (SemEval) is a triennial workshop in computational linguistics that provides a standardised set of tasks and evaluations for semantic analysis systems. In 2010 the Workshop included a task entitled Parser Evaluation using Textual Entailments (PETE; Yuret et al., 2010), which proposed a new approach to parser evaluation. Rather than an *intrinsic* comparison of parser output against a manually annotated in-domain gold standard, it considers an *extrinsic* evaluation, where parsers are judged on their performance in an external task that is formalism-independent and less reliant on expert annotation. This differs from the application-based evaluations mentioned in Section 4.1 in that it is not the evaluation system which is being recast to suit different parsers. Instead, each parser uses its own representation to produce the necessary output for the task.

The proposed task is that of recognising textual entailment (RTE) between sentences. Given *text* and *hypothesis* sentences, textual entailment recognition is the task of determining whether the *meaning* of the hypothesis is logically inferable (i.e. *entailed*) from the text. Entailment recognition captures a broad range of inferences that are useful in applications such as Question Answering, Information Retrieval, and machine translation (Dagan et al., 2009). Figure 4.1 gives an example text and hypotheses from the task (Yuret et al., 2010).

Text:
The man with the hat was tired.

Hypothesis-1 (YES):
The man was tired.

Hypothesis-2 (NO):
The hat was tired.

FIGURE 4.1: Example text and hypotheses for Textual Entailment evaluation. Here, the text entails the first hypothesis (answer YES), but not the second (answer NO).

Unlike more general RTE tasks, PETE restricts the entailments to those which can purely be inferred from syntactic information alone. This is done to ensure that the task can function purely as an evaluation of a parser. The full RTE task typically requires external knowledge such as the WordNet lexical database of synonyms (Fellbaum, 1998) to disambiguate unknown entities. In this way entailment evaluation automatically focuses attention on syntactic constructions that affect semantic interpretation. The labels assigned to constituents are formalism-specific and accounted for in labeled evaluation, but in unlexicalised formalisms like the Penn Treebank grammar, an incorrect label does not always affect the meaning of the sentence. By focusing on semantic interpretation of syntactic ambiguity it is ensured that semantically relevant differences between parses leads to different entailments; semantically irrelevant differences do not and are effectively ignored.

4.4.1 Data

The PETE evaluation requires gold-standard entailment pairs annotated by humans. However, since entailment is modelled on a natural human competence, it is practical to collect large amounts of annotated entailment data quickly and inexpensively. A native speaker's grasp of language should be sufficient for accurate entailment prediction, in contrast to the expert linguistic knowledge required for treebank annotation.

```
<entailment-corpus>
        <pair id="2082.N" entailment="YES"
            word1="trading" word2="resume">
                <t>
                    Trading in AMR shares was suspended shortly after
                    3 p.m. EDT Friday and didn't resume.
                </t>
                <h>
                    Trading didn't resume.
                </h>
        </pair>

          ...

</entailment-corpus>
```

FIGURE 4.2: A text and hypothesis sentence pair in XML format. This sentence is taken from the PETE development data; line breaks have been added for clarity.

Amazon's Mechanical Turk (AMT) is an online labour market named after a fake chess-playing machine constructed in the late 18th century. In Amazon's Turk market, *Human Intelligence Tasks* consisting of an arbitrary number of questions may be placed by *Requesters*. *Workers* (also known as *Turkers*) can browse the tasks, and may elect to complete any of them for a small amount of money specified by the Requester of the task. Requesters must approve the work completed by Workers before payment, and may also choose to assign bonuses for particularly exceptional work. Necessary qualifications can be stipulated for tasks, such as a minimum performance threshold on an external test or a minimum percentage of previously accepted work. Payments per task may be as low as 1 US cent, making AMT an ideal place to acquire large quantities of annotated data for little expense.

The PETE dataset was drawn from several sources in order to gather syntactic dependencies that were challenging for state-of-the-art parsers. In contrast to PARSEVAL and dependency evaluations, Textual Entailment evaluation is based on natural human linguistic competence: the ability to comprehend a sentence and answer a simple YES or NO question about it. The relative simplicity means that expert annotation is not required, so the chosen sentences could be annotated using AMT at a very low cost. Quality control is an important aspect of completing any tasks on AMT and so the following stipulations were made:

- Each text-hypothesis pair was annotated by 5 Workers
- All annotations with unanimous agreement from at least 3 Workers were kept

The task provided two sets of data in an XML format. A 66 sentence development set was initially provided to introduce the data format and provide examples of the types of entailments used for the task. The development data included gold-standard entailment labels and marked the content words relevant to the entailment test for purposes of clarity.

The test data consisted of 301 sentences in the same format as the development data. The test data did not contain labels and did not mark content words of interest. The content words were omitted as the task was conceptualised as closely modelling the natural human interpretation of language – a task done without convenient markings on relevant words.

4.4.2 Conducting entailment evaluation

Evaluating a parser using the entailment task is a simple matter. The parser is first used to analyse the given text and hypothesis sentence pair. Then, the analyses are processed in order to produce an entailment decision of YES or NO. This decision is marked as either correct or incorrect against the true entailment. An aggregate score over all the text-hypothesis pairs are calculated and returned as a measure of performance.

Conducting the entailment evaluation requires every parser to implement a system to convert syntactic analyses into entailments. Common methods for making the entailment decision include heuristic comparisons of dependencies, paths in the dependency graph of the sentence, or supervised classification using syntactic elements as features (Yuret et al., 2010). The external nature of entailment evaluation makes it fundamentally formalism-independent; as parser is deciding between two analysed sentences within its own formalism, no conversion scheme is necessary.

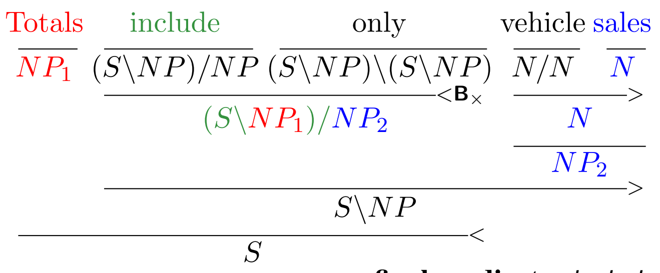
Entailment evaluation automatically focuses attention on syntactic constructions that affect semantic interpretation. The labels assigned to constituents are formalism specific and accounted for in labeled evaluation, but in unlexicalised formalisms like the Penn Treebank grammar an incorrect label will not always affect the meaning of the sentence. By focusing on semantic interpretation it is ensured that semantically relevant differences between parses leads to different entailments; semantically irrelevant differences do not and are effectively ignored.

4.5 Evaluating the C&C parser with Textual Entailments

In this section we describe our work in implementing the SCHWA system for evaluating the C&C parser. This system was submitted for the SemEval 2010 PETE task and placed second in the evaluation. We described in Section 3.1.4 how dependencies are produced in CCG analyses; these dependencies are the basis of our entailment evaluation system. Our main intuition was that the core dependencies in the text should also be present in the hypothesis sentence. Our system constructed *predicates* from the dependencies in the hypothesis sentence and searched the text sentence for the predicates. If any of the predicates were found in the text, the system returned YES, otherwise, it returned NO.

4.5.1 Predicate Generation

Recall that each CCG dependency is represented as a 5-tuple encoding the head word, its category, the argument slot, the head of the argument, and whether the dependency is long-range. We generated predicates based on the head word of each dependency, and the concatenation of all argument heads for that head word.



$\langle$*includes*, $(S\backslash NP)/NP$, 1, *Totals*, $-\rangle$

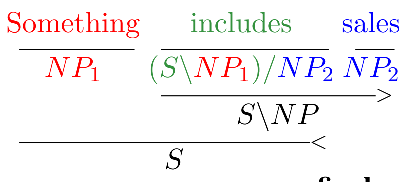
$\langle$*includes*, $(S\backslash NP)/NP$, 1, *sales*, $-\rangle$

final predicate: *include(total, sale)*

FIGURE 4.3: Predicate generation from CCG dependencies. The words in the final predicates are lemmatised to facilitate more consistent comparisons.

Consider the example derivation and selected dependencies in Figure 4.3. These dependencies share the same head word (*include*) and head category. A predicate is produced from this pair of dependencies by taking the head word as the *base* of the predicate, and the two argument heads as the *arguments* of the predicate. Arguments are sorted by the slots of the dependency that they fill.

Predicates were generated from the dependency analysis of the text and hypothesis sentences. We then attempted to *unify* the predicates from the hypothesis sentence with the predicates in the truth sentence. A successful unification of predicates a and b occurs when the head words of a and b are identical and their argument slots are also identical. If any predicate from the hypothesis sentence unified with



$\langle$*includes*, $(S\backslash NP)/NP$, 1, *Something*, $-\rangle$

$\langle$*includes*, $(S\backslash NP)/NP$, 1, *sales*, $-\rangle$

final predicate: *include(*, sale)*

FIGURE 4.4: Conversion of indefinites into wildcard markers for hypothesis dependencies.

a predicate from the truth sentence, our system returned YES, otherwise the system returned NO. All words used in predicates were lemmatised with the `morpha` tool (Minnen et al., 2000) included with the C&C parser. Lemmatisation removes inflections from a word and reduces it to its base form (e.g. *Totals* $\rightarrow$ *total*). This process eliminates surface differences between words and facilitates a more accurate matching between them.

We used the 66 sentence development set to tune the predicate generation process. While analysing the hypothesis sentences in the development set, we noticed that many examples replaced nouns from the truth sentence with indefinite pronouns such as *someone* or *something* (e.g. *Someone bought something*). In most of these cases the indefinite would not be present in the truth sentence at all, leading to a unification failure. To deal with this we converted all indefinite pronouns into wildcard markers that would match any argument, as depicted in Figure 4.4.

Several examples in the development data used entailments that were converted into the *passive* voice. This occurs when the usual order of subject and object in a sentence is inverted (e.g. *Jack likes Jill* vs. *Jill was liked*). We adjusted the argument slots of dependents in passive sentences to account for the inversion in subject and object.

4.5.2 Heuristic Pruning

In its most naïve form our system is heavily biased towards excellent recall but poor precision. This is because the system produces a YES answer if *any* predicate from the text matches a predicate in the hypothesis, even if the match is unrelated to the proposed entailment. Thus, the system is more likely to produce YES answers and hence improve recall at the expense of precision. To address this we evaluated a number of heuristics to prune the predicate space on the development set.

A number of CCG dependencies create predicates that are semantically irrelevant for the purposes of entailment prediction. For example, dependencies created between determiners and the noun they head are unlikely to affect entailment decisions. The presence of predicates generated from these dependencies combined with our requirement for only a single predicate match between text and hypothesis is a potential source of error in our system. Since determiner relations are commonly retained between text and hypothesis sentences, the corresponding predicates will unify and return a positive entailment prediction regardless of the other predicates. Using the part-of-speech tags of each word in the text and hypothesis sentences, we experimented with pruning predicates headed by various word classes. Based on performance on the development set, our final system pruned predicates headed by determiners, prepositions, and adjectives.

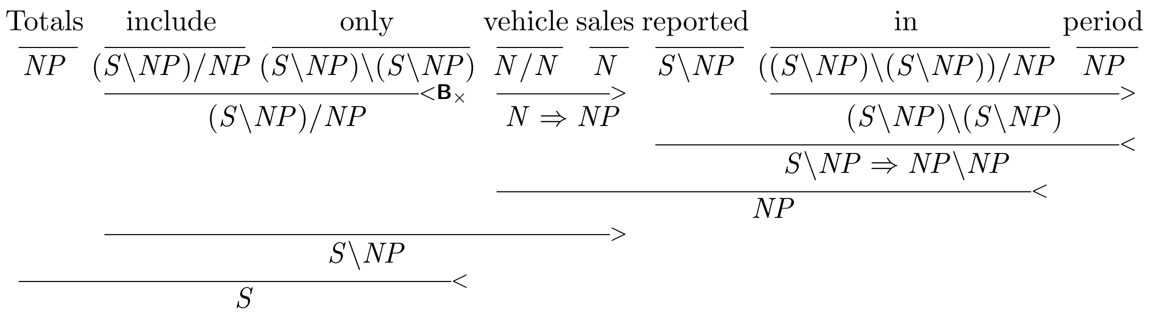

Totals include only vehicle sales reported in period
NP
$(S\backslash NP)/NP$
$(S\backslash NP)\backslash(S\backslash NP)$
N/N
N
$S\backslash NP$
$((S\backslash NP)\backslash(S\backslash NP))/NP$
NP
$<\mathbf{B}_{\times}$
$(S\backslash NP)/NP$
$>$
$N \Rightarrow NP$
$>$
$(S\backslash NP)\backslash(S\backslash NP)$
$<$
$S\backslash NP \Rightarrow NP\backslash NP$
$<$
NP
$>$
$S\backslash NP$
$<$
S

FIGURE 4.5: An example CCG derivation for the sentence, *Totals include only vehicle sales in period*.

Head	Category	Slot	Argument
only	$(S\backslash NP)\backslash(S\backslash NP)$	1	*include*
vehicle	N/N	1	*sales*
in	$((S\backslash NP)\backslash(S\backslash NP))/NP$	1	*reported*
in	$((S\backslash NP)\backslash(S\backslash NP))/NP$	2	*period*
reported	$S\backslash NP$	1	*sales*
include	$(S\backslash NP)/NP$	1	*Totals*
include	$(S\backslash NP)/NP$	2	*sales*

TABLE 4.1: The dependencies licensed by the derivation in Figure 4.5.

```
only(include)
vehicle(sale)
report(sale)
include(total, sale)
```

FIGURE 4.6: The predicates generated after pruning from the dependencies in Table 4.1.

Figure 4.5 gives the full CCG derivation of an example sentence. Table 4.1 lists the dependencies licensed by the derivation; these are converted into predicates for the sentence. Following lemmatisation, pruning is conducted, and all predicates headed by determiners, prepositions, and adjectives are excluded from the list. In this example all dependencies headed by the preposition *of* are pruned to produce the final predicate list in Figure 4.6.

Table 4.2 lists some examples from the development set, along with the key predicates found and unified by our system. The first example shows the conversion of an indefinite into a wildcard for correct unification. The second example shows an exact syntactic match between *is* and its arguments in both sentences (in the predicate it is lemmatised to its base form *be*). Finally, the third example presents an incorrect match, where a non-core dependency between *pegboard* and *go* causes the predicates to unify when the entailment is false.

	Sentence	Key Predicate	Predict	Correct
Text	*He has a point he wants to make, and he makes it, with a great deal of force.*	`make(he, point)`	YES	YES
Hypothesis	*Somebody wants to make a point.*	`make(*, point)`		
Text	*I reached into that funny little pocket that is high up on my dress.*	`be(pocket, high)`	YES	YES
Hypothesis	*The pocket is high up on something.*	`be(pocket, high)`		
Text	*That was where the pegboard would go on which he would hang his hand tools.*	`go(pegboard)`	YES	NO
Hypothesis	*He would hang something where the pegboard would go.*	`go(pegboard)`		

TABLE 4.2: Some example text and hypothesis pairs from the development data, along with the output of our evaluation system and the unifying predicate.

4.6 Results

We report results for our system (SCHWA) over the 301 sentence test set in Table 4.3, as well as the median, baseline, and lowest results from the SemEval 2010 Shared Task (Yuret et al., 2010). Our overall accuracy on the test set was 70%, which was nearly 20% better than the baseline system which answered YES for all entailments. Performance over YES entailments was roughly 20% higher than accuracy over NO entailments. This bias towards YES entailments is a reflection of our single match heuristic that only required one predicate to unify between the text and hypothesis sentences before predicting a YES entailment. The excellent performance of our system compared to the baseline was achieved using a simple predicate heuristic, demonstrating the potential for this task as an effective evaluation of parsers.

Our system placed second overall in the task evaluation. The best performing system was the Cambridge system, which achieved an accuracy of 72% and also used the C&C parser. Unlike our system, the Cambridge system used the GRs output from the C&C parser. As GRs are less specific than CCG dependencies, this may have provided better generalisation, leading to the 2% performance advantage (Rimell and Clark, 2010). However, it is encouraging that two different evaluation approaches using two different output formats achieved similar overall results for the C&C parser. Our system and the Cambridge system were the only two systems that achieved accuracies above 70% in the task. Of note is that our system outperformed the far more complex MARS system, which used an SVM classifier with features generated from the dependency graphs of the text and hypothesis sentences to predict entailment (Wang and Zhang, 2010).

	YES entailment			NO entailment			Overall	
System	correct	incorrect	A (%)	correct	incorrect	A (%)	accuracy (%)	F-score
Cambridge	98	58	63	120	25	83	72	71
SCHWA	**125**	**31**	**80**	**87**	**58**	**60**	**70**	**74**
MARS	116	40	74	85	60	59	67	70
median	71	85	46	88	57	61	53	50
baseline	156	0	100	0	145	0	52	68
low	68	88	44	76	69	52	48	46

TABLE 4.3: Final results in the task evaluation over the test set for YES and NO entailments. The performance of the top three systems as well as the median, baseline, and last system are shown.

	YES entailment			NO entailment			Overall	
System	correct	incorrect	A (%)	correct	incorrect	A (%)	accuracy (%)	F-score
Gold deps	34	6	85	22	4	90	87	87
Parsed deps	32	8	80	20	6	77	79	82

TABLE 4.4: Results for the SCHWA entailment evaluation over the development set for YES and NO entailments.

4.7 Discussion

Table 4.4 shows our results over the development corpus. The 17% drop in accuracy and 8% drop in F-score between the development data and the test data suggest that our heuristics may have overfitted to the development data. This problem is a natural consequence of the small amount of data. The development set is only 3% the size of the usual development data for parsers, Section 00 of the Penn

Treebank. The test set is slightly larger, at 12.5% the size of Penn Treebank Section 23. The small size of the corpora makes it difficult to test a conversion scheme from parser output to entailments. 66 development sentences is an insubstantial number, and any change in one entailment decision results in a more than 1% change in final accuracy. Given larger development and test corpora, more sophisticated heuristics could be developed for further fine-tuning.

4.7.1 Analysis with gold standard parses

Our entailment system's errors could be broadly divided into two classes: those due to incorrect parses, and those due to incorrect comparison of the parses. To investigate the relative contributions of these two classes of errors, we conducted another test using manually annotated CCG dependencies over the 66 sentence development set. This allowed us to evaluate our system using gold standard analyses. It is expected that with gold standard dependencies accuracy should increase; performance closer to 100% reflects a transparent mapping scheme that does not introduce any errors into the analysis.

The gold standard parses were prepared with the annotation tool used by Honnibal et al. (2009). The tool presents the annotator with a CCG derivation produced by the C&C parser. The annotator can then correct the lexical categories assigned by the parser or add bracket constraints to the analysis using the algorithm described by Djordjevic (2006). The sentence is then reparsed with the new categories and constraints, and this process is repeated until the desired derivation is produced. Unfortunately, only one annotator was available, so we were unable to calculate inter-annotator agreement scores to examine the quality of our annotations.

Our results with gold standard dependencies are shown in Table 4.4. The final accuracy is 87%, which is an 8% increase over the parsed results. This demonstrates that our mapping from dependencies to predicates is not perfect. However, the result is consistent with previous attempts at cross-formalism evaluation using the C&C parser. For example, the GRs evaluation conducted by Clark and Curran (2007a) was restricted to an upper bound of 84.8%, suggesting that even in its infancy textual entailment evaluation is a promising avenue for cross-formalism comparison of parsers. Additionally, the predicate system we implemented for this task was significantly easier to construct and test than the many-to-many GRs mapping developed in Clark and Curran (2007a); with a larger and more varied development set we could expand and refine our relatively simple set of heuristics.

Manual inspection of the remaining errors in the development set showed that some were due to incorrect parses for the hypothesis sentence, and others were due to entailments which the parser's dependency analyses could not resolve, such as *They ate whole steamed grains* ⇒ *The grains were steamed.* Overall, the largest source of errors was our matching heuristics, suggesting that our approach to the task must be improved with more development data before it can be considered a transparent evaluation of the parser.

4.8 Summary

In this chapter we have described a number of parser evaluation measures, including dependency evaluation and the CCG dependency recovery metric. We have discussed the problems with parser evaluation methodologies, particularly for cross-formalism comparison, and motivated the development of a new evaluation. We have used the SemEval-2010 Parser Evaluation using Textual Entailments task as an example of a task which addresses some of the faults of PARSEVAL and other evaluations.

We developed a system to evaluate the C&C parser using textual entailments. We converted the parser output into a set of predicate structures and used these to establish the presence of entailment. Our system achieved an overall accuracy of 79% on the development set and 70% on the test set. This placed our system second in the task evaluation and it was one of only two systems to achieve an accuracy above 70%. The gap between our development and test accuracies suggests our heuristics may have been overfitted to the development data; this problem will be overcome by using a larger, better documented data set for development and final testing.

Our investigation using gold-standard dependencies established an upper bound of 87% on the development set for our approach to the task. While this is not ideal, we note that previous efforts at cross-parser evaluation have shown that it is a difficult problem, and that this number is consistent with previous attempts at cross-formalism comparison with the C&C parser.

The experience of conducting an entailment evaluation of the C&C parser demonstrates that the concept of a minimal extrinsic evaluation is a promising avenue for formalism-independent parser comparison. Comparable accuracy results from previous work have been achieved with a fraction of the effort and complexity of past conversion attempts. With more data, we see potential in this task for performing better parser evaluation.

CHAPTER 5

CCG Reranking

In this chapter we describe the implementation of a reranker for the C&C parser based on the approach described in Charniak and Johnson (2005). This is the first work describing reranking for CCG parsing, and preliminary results based on this chapter and Chapter 7 are to appear at the Australasian Language Technology Workshop 2010 under the title *Reranking a wide-coverage* CCG *parser*.

CCG is very different to the Penn Treebank-style grammar that the Collins and Charniak rerankers were designed for. CCG parsers use different evaluation metrics and produce different tree structures to Penn Treebank parsers, which precluded a direct appropriation of an existing reranker for the C&C parser. The process of creating the reranker required some refinements of the methodology described in Section 2.6 as well as the redesign and reimplementation of the features and approach described by Charniak and Johnson (2005).

Our core contributions are the development of the reranking framework for CCG, a comprehensive blueprint for implementing reranking features, and strong evidence that the Charniak reranking system negatively impacts CCG parsing performance. This motivates an extensive analysis of errors made by the parser in Chapter 6, and the development of new features targeted at addressing these errors in Chapter 7.

5.1 Methodology

We frame the reranking task for CCG parsing as follows: given an n-best list of CCG parses, ranked by the parser, choose and return the parse that is as close as possible to the gold standard. The parser we use is the n-best C&C parser described in Section 3.4. The normal-form C&C model described in Clark and Curran (2007b) was used for all experiments described in this chapter and in Chapter 7. We use the CCG labeled dependency F-score as described in Section 4.2 to judge closeness to the gold standard, in line with previous work on reranking and on the development of CCG parsers.

As with previous work in reranker development, using the standard evaluation metric for the parser means that training data must be generated from sentences with an annotated gold standard. Following Collins (2000) and Charniak and Johnson (2005) we used the established cross-validating method over Sections 02-21 of CCGbank to creating reranker training data (see Section 2.6.1). We excluded each section of the collected training data in turn, training the parser on the remaining 19 sections, and parsed the excluded section to produce n-best parses. This procedure was repeated twenty times – once for each section – and the n-best sentences collated together to form reranker training data.

Since the expressiveness of CCG allows multiple derivations to yield an equivalent dependency structure (see Section 3.1.2), distinguishing parses based on structure alone is insufficient. Differences in surface forms are irrelevant in a CCG derivation if the same dependencies are licensed (see Section 3.1.3). There is extra ambiguity in the CCG reranking task since the n-best parse list may contain several distinct parses that are semantically equivalent. We address this by filtering the output of the n-best parser to eliminate all analyses that share the same dependencies with a previously generated parse for the sentence. This process is a departure from previous work, and was carried out during both training and run-time phases for the reranker.

Using the real-valued F-score metric as a target value allows a choice between two widely used machine-learning techniques – *binary classification* and *logistic regression*. In binary classification, the task is to predict the most probable *class* for each unknown instance; the aim is to minimise the number of incorrect predictions. We model reranking as a two-class problem, where the parses with the highest predicted F-score for each sentence are labeled as POSITIVE, and all other parses are labeled as NEGATIVE. The role of the reranker is to predict the most probable POSITIVE parse from the n-best input. At training time, a training instance is labeled POSITIVE if it has the highest F-score in the n-best list for that sentence, and NEGATIVE otherwise. This means there may be multiple positive parses for one parse if they all have an equally high F-score.

For logistic regression, the task is similar to that of classification, but instead of choosing one class from a fixed set, the intention is to predict a real value so as to minimise the sum of errors. Reranking as a regression task has the natural target value of the F-score of each parse. To clarify this scheme, Table 5.1 lists some example 3-best parses with their F-scores and the labels they would be assigned in our system.

The binary classification approach is a more literal reflection of the run-time reranking task – the reranker must ultimately choose one parse to return, and discard the rest. However, classification also results in

Technique	Sentence	Parse	F-score	Class
Classification	α	1	88.50	POSITIVE
		2	88.50	POSITIVE
		3	86.20	NEGATIVE
	β	1	96.35	POSITIVE
		2	89.00	NEGATIVE
		3	85.05	NEGATIVE
Regression	α	1	88.50	0.8850
		2	88.50	0.8850
		3	86.20	0.8620
	β	1	96.35	0.9635
		2	89.00	0.8900
		3	85.05	0.8505

TABLE 5.1: Example 3-best lists of parses, their F-scores, and the assigned labels for binary classification and regression experiments.

much of the information in the label (F-score) for each parse to be ignored. In this aspect, it is logistic regression that retains the most information from the parse's F-score. Both classification and regression approaches were implemented using the MEGAM[1] maximum entropy modelling package.

We experimented with values of 10 and 50 for n to balance between the potential accuracy improvement and the efficiency of the reranker. n was kept constant between the training data and the final test data (i.e. a reranker trained on 50-best parses was then tested over 50-best parses) to maintain a consistent distribution of parses. Features were generated over the n-best parses in the training data and the appropriate label assigned based on the F-score and whether classification or regression was being employed. We parsed Section 24 of CCGbank using the full C&C model for use as tuning data for MEGAM. At run-time, features were generated over the n-best parses of the test data, and the most probable POSITIVE parse (classification) or the parse with the highest predicted F-score (regression) was returned.

There are nearly 40,000 sentences in Sections 02-21 of CCGbank; with up to 50 parses per sentence, the size of the training data is substantial. *Feature pruning* is often necessary with such large datasets to eliminate unreliable or redundant features. Following Charniak and Johnson (2005) we implemented feature pruning on the reranker training data as follows. For each sentence, define a feature as being *pseudo-constant* if it takes the same value over all the n-best parses for that sentence. We retain all features that are not pseudo-constant in at least t sentences in the training data. We experimented with values of 0, 2, and 5 for t to investigate the effect of this simple count-based feature pruning.

[1]`http://www.umiacs.umd.edu/~hal/megam`

5.1.1 Statistical Significance

In evaluating the performance of reranking we conducted significance testing to determine if changes in performance over the baseline parser were meaningful. We used the test described in Chinchor (1992) and accepted significance at $p = 0.05$ and lower.

The significance testing takes the output of the baseline parser and the reranked parser and randomly interchanges entries between them, counting how many times the difference in a given metric between the two sets has increased. If the responses for the baseline and reranked parser are drawn from different distributions then randomly mixing them should bring the scores for the metric closer together as the two sets combine into an intermediate distribution. Random variations will result in movement in either direction when the sets are from the same distribution. If the change is observed to increase less than 5% of the time then the difference between the distributions is considered to be significant[2].

5.2 Features for Reranking

Features are used to represent each parse in the reranker model. Our initial reranker implementation used the features described in Charniak and Johnson (2005); this allowed us to evaluate the effectiveness of features which had already found to be useful in reranking Penn Treebank parses.

The C&C parser produces multiple output formats. Each parse has a corresponding CCG derivation tree as well as a list of dependencies licensed by the derivation. All features in this section are calculated over the derivation tree for each parse.

Most features were implemented as simple boolean indicator functions with a value of 1 if the feature is present in a tree and 0 otherwise. Some feature types described in the section have multiple forms based on words, part-of-speech tags, categories, and distances in the tree; each form is counted as a separate feature. The greater influence of real-valued features in maximum entropy modelling (due to the exponentiation within the framework) was mitigated by taking the logarithm as the final feature value. Additionally, the C&C parser assigns a *score* to each parse that is roughly equivalent to its log probability under the parser model. The logarithm of the score and the initial log rank of the parse as assigned by the parser were used as core features for each derivation.

[2]The script used is based on David Vadas' Python implementation of Dan Bikel's Perl script: `http://www.cis.upenn.edu/~dbikel/software.html#comparator`

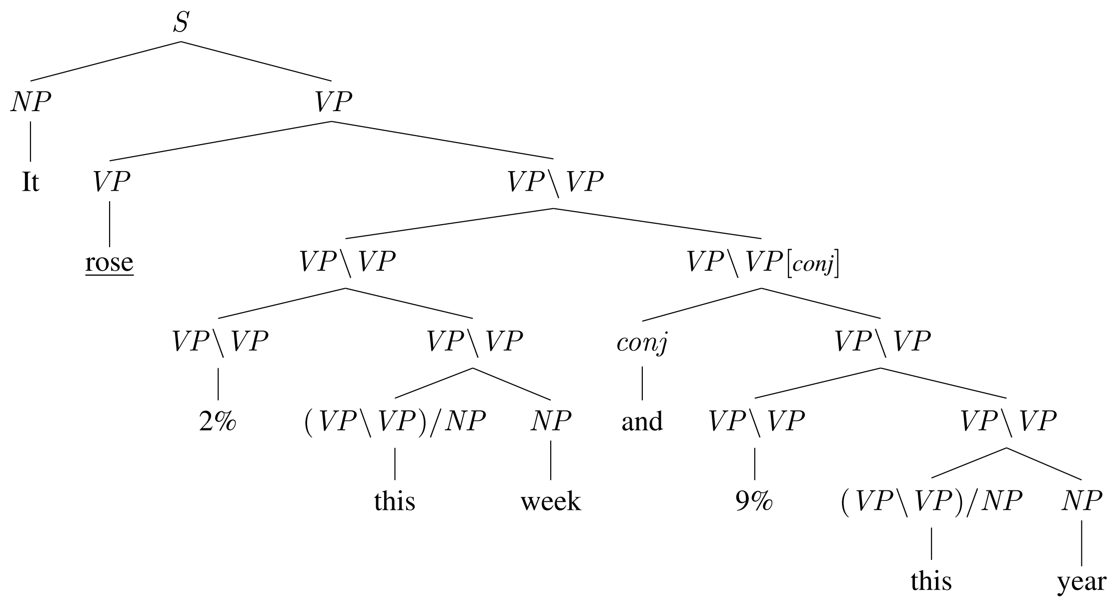


FIGURE 5.1: The correct parse for the sentence.

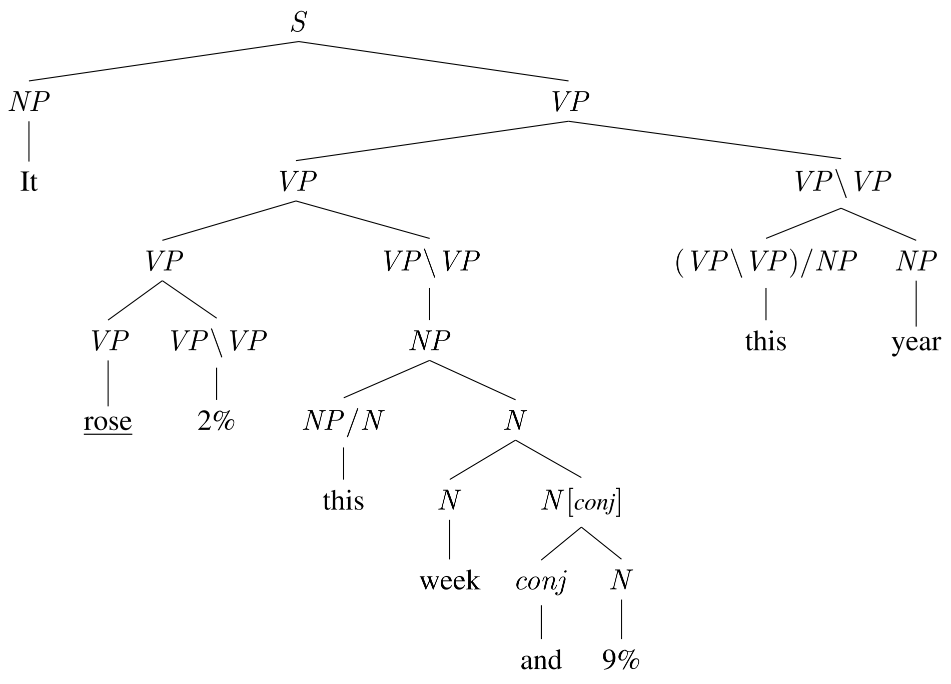


FIGURE 5.2: An incorrect parse featuring a conjunction error.

FIGURE 5.3: Two CCG derivations for the sentence, *It rose 2% this week and 9% this year*. The category $S\backslash NP$ is shortened to VP and the head word *rose* is underlined.

Figure 5.3 depicts a correct and incorrect CCG derivation in tree form for sentence (10), *It rose 2% this week and 9% this year*. The two trees represent an ambiguity in the potential coordination structure where a parser can make an incorrect choice early in the derivation and form a poor analysis. The correct tree in Figure 5.1 conjoins the phrases *2% this week* and *9% this year*, to form a modifier that attaches to the verb *rose*. The incorrect tree in Figure 5.2 conjoins *week* and *9%*, forcing the application of a unary rule to transform the local tree into a modifier to attach to *rose*. The fragment *this year* has been moved out of the conjunction entirely, losing the 9% quantifier (i.e. similar to *It rose* <u>*quickly*</u> *this year*).

(10) It rose 2% this week and 9% this year.

In our description of reranker features we will use these example trees to illustrate the feature values that are extracted and used in the reranker. For brevity, the category $S\backslash NP$ is shortened to VP (verb phrase) where necessary in the parse trees depicted in this section.

5.2.1 Tree Topology Features

Tree topology describes the overall shape of the parse tree and attempts to capture general conventions of English. It is known that English generally favours right-branching parse trees, with larger ('heavier') constituents occurring towards the end of the sentence. This makes the fragment *cars, trains, and enormous silver flying machines* sound more natural than *enormous silver flying machines, trains, and cars*. In coordination it is expected that both sides of the conjunction will have a parallel structure, or else the larger conjunct is again preferred towards the right. These guidelines distinguish the correct parse tree in Figure 5.1 from the incorrect parse tree in Figure 5.2 – the incorrect tree is more left-branching than the correct tree, with a shallower depth of balance in the coordination.

The following features attempt to model tree topology with reference to the general conventions of English. They allow the reranker to give preference to trees which better reflect these conventions.

Coordination Parallelism (CoPar): records coordination parallelism at various depths. For sentences containing any coordination constructions, this feature indicates whether both sides of a coordination are identical in structure and category labels at depths of 1 to 4 from the coordinator. A placeholder feature with a constant value is generated if the sentence does not contain any coordination structures.

Coordination in CCGbank is represented using the binary rules described in Section 3.2. This means that coordination is more difficult to visualise in the tree format; one half of the conjunction bears the same parent as the coordinator in the sentence, but the other half is attached at a higher level in the tree. This can be seen in Figure 5.4.

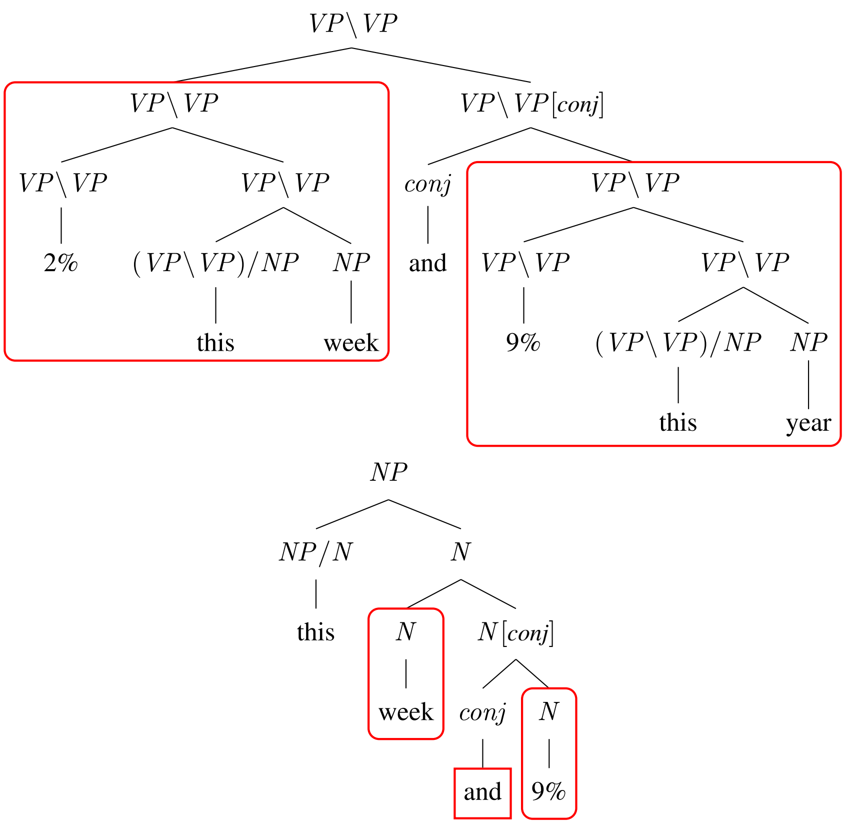


FIGURE 5.4: Coordination parallelism to depth 3 in the correct parse and to depth 1 in the incorrect parse.

Coordination Length Parallelism (CoLenPar): when coordination constructions are not balanced in a tree, the right-branching bias of English tends to place the largest constituent on the right of the structure. This feature indicates the difference in size between two halves of a coordination (where size is the number of words in each half) as well as whether the latter half is the final element in the coordination.

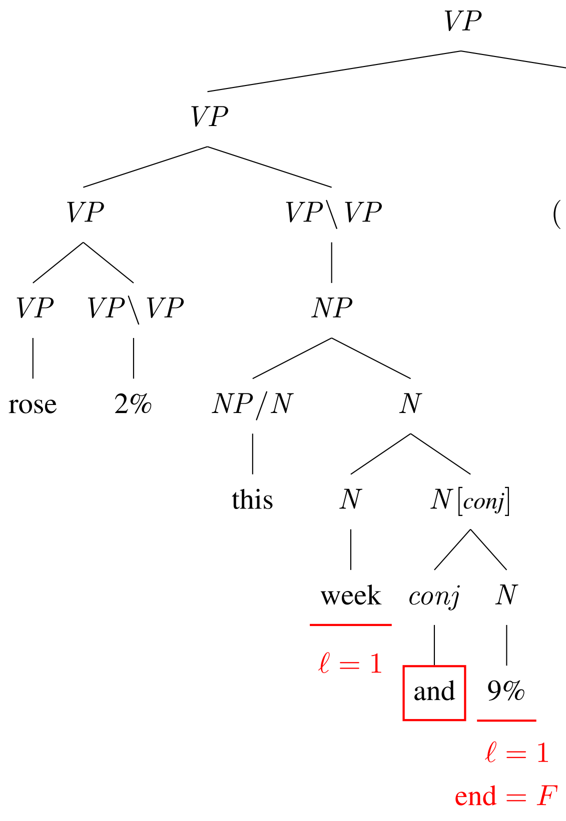


FIGURE 5.5: Coordination length parallelism for the incorrect parse, where the two halves of the coordination have length 1 and the latter half is not at the end of the sentence. In contrast, the correct parse has lengths of 3, and the latter half is at the end of the sentence.

Heavy: encodes the category and size of the subtree rooted at each non-terminal, whether the non-terminal is at the end of the sentence, and whether it is followed by punctuation. This crudely captures the tendency for larger constituents to lie further to the right in a tree.

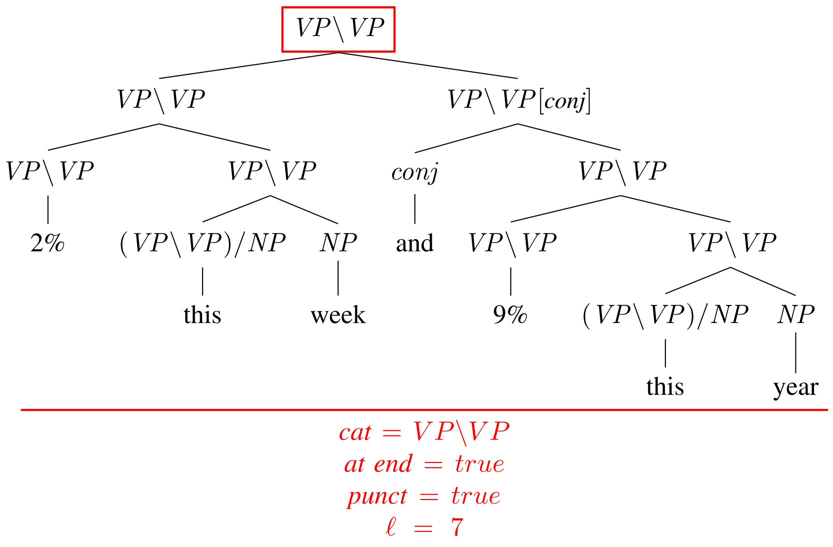


FIGURE 5.6: Constituent heavyness for the correct parse. The more left-branching structure in the incorrect parse would result in a number of heavier non-terminals being non-final in the sentence.

SubjVerbAgr: the Penn Treebank part-of-speech tags (see Appendix A) distinguishes between singular and plural nouns and verbs. In a grammatical sentence the pluralisation should agree between the subject noun and verb (e.g. *Jack bakes a cake* against *Jack bake a cake*). This feature captures the conjoined POS tags of the subject noun and verb in a sentence to identify cases where the pluralisation does not agree. The subject is assumed to be the final NP before the verb phrase ($S\backslash NP$) in a sentence.

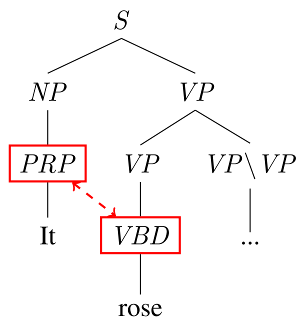
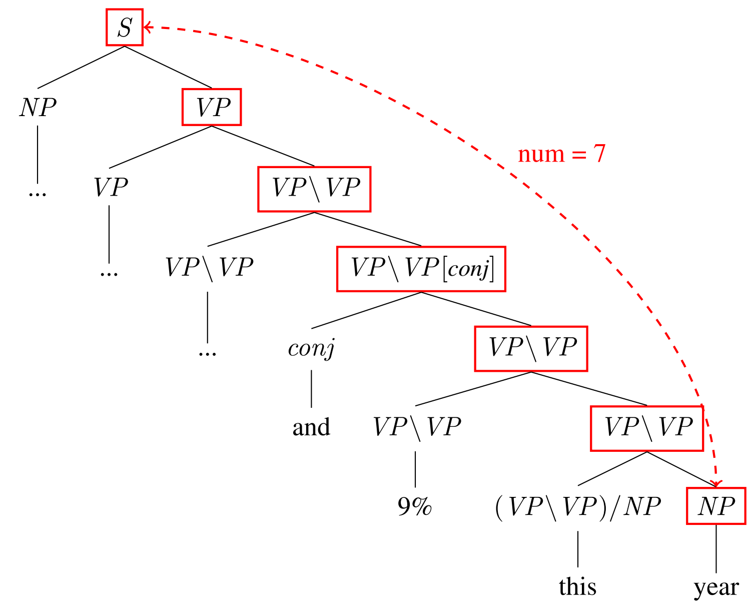


FIGURE 5.7: Subject-Verb Agreement captures the part-of-speech tags of the subject and verb in a sentence. This feature value is unchanged between the correct and incorrect parses as the POS tags are the same.

RightBranch: encodes the number of non-terminals on the longest path from the root of the tree to the right-most non-punctuation node in the tree, and the number of non-terminals in the tree that are not on this path. This distinguishes trees that are more right-branching and thus preferred in English.



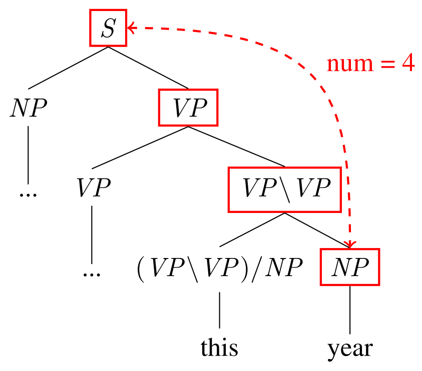


FIGURE 5.8: Non-terminals on the right-branching path for the correct and incorrect parse trees.

5.2.2 Local Context Features

These features (adapted from Charniak and Johnson (2005)) attempt to incorporate layers of vertical and horizontal context that are difficult to encode in the parser model due to dynamic programming. The goal of these features is to identify parts of the tree that are not sensible in a wider context.

Edge: captures the terminals that immediately precede and follow the subtree rooted at each non-terminal in the tree. This crudely captures poor attachment decisions between local trees and their context. The terminals are encoded as lexical items (the words) and as part-of-speech tags.

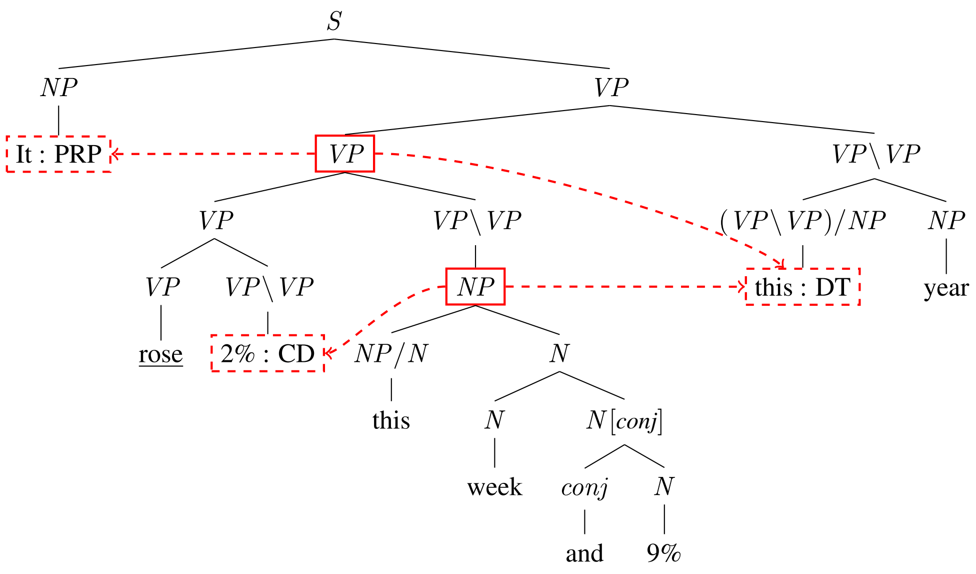


FIGURE 5.9: Edges features for a non-terminal in the incorrect parse. Part-of-speech tags are marked on the edge words.

Rule: captures the equivalent CCG combinator application represented at each non-terminal node. This is equivalent to a context-free rule and may identify cases where the parser has selected a poor rule.

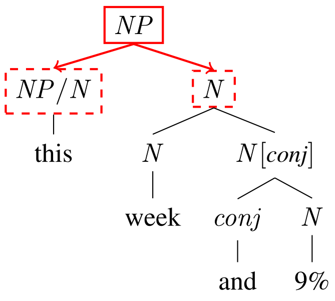


FIGURE 5.10: A Rule feature. In this subtree the rule application is $NP \rightarrow NP/N\ N$.

Heads: represents the heads of pairs of constituents in the tree as another way of capturing poor tree attachment. At each non-terminal in the tree, the *head child* is defined as the child which provides the eventual head of the non-terminal (this should not be confused with the *head word* of the sentence). This feature attempts to capture poor head interactions at every non-terminal in the tree. It ascends n levels from the non-terminal, and if the previous node is not the head child of the current node, the head *words* of the starting node and the new node are encoded as a feature. The heads are encoded in their lexical form as well as generalised using the POS tags assigned to the words.

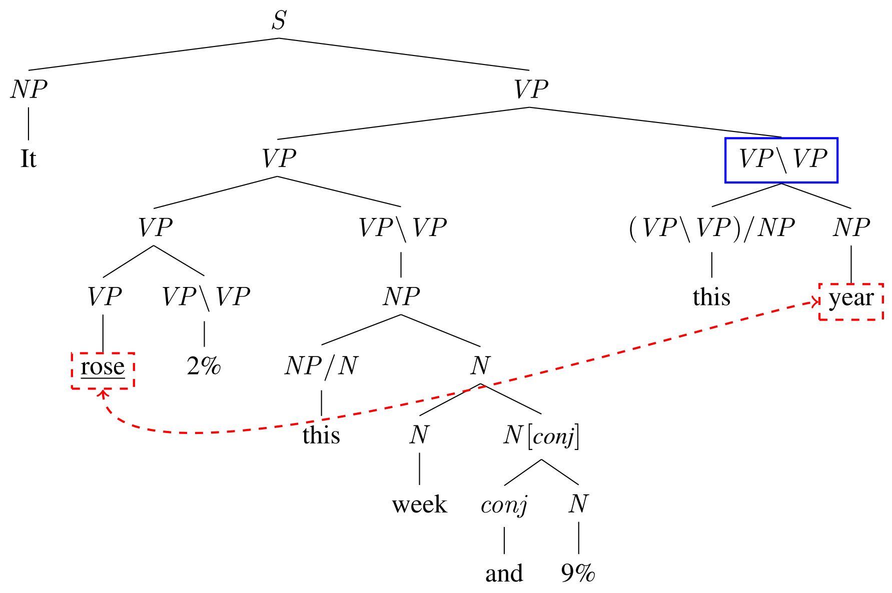


FIGURE 5.11: Heads features for a fragment of the incorrect parse. The feature is generated from the non-terminal marked in blue and ascending $n = 1$ levels. There is no equivalent feature generated in the correct tree due to the deeper right branching.

Head Tree: records the entire tree fragment (in a bracketed string format) projected from the head word of the sentence. The projection is constructed by taking the head word and its sibling (if it exists), and proceeding up the tree to the root, including the sibling node at each level. The head tree attempts to capture the 'essential element' of the sentence as indicated by the parse.

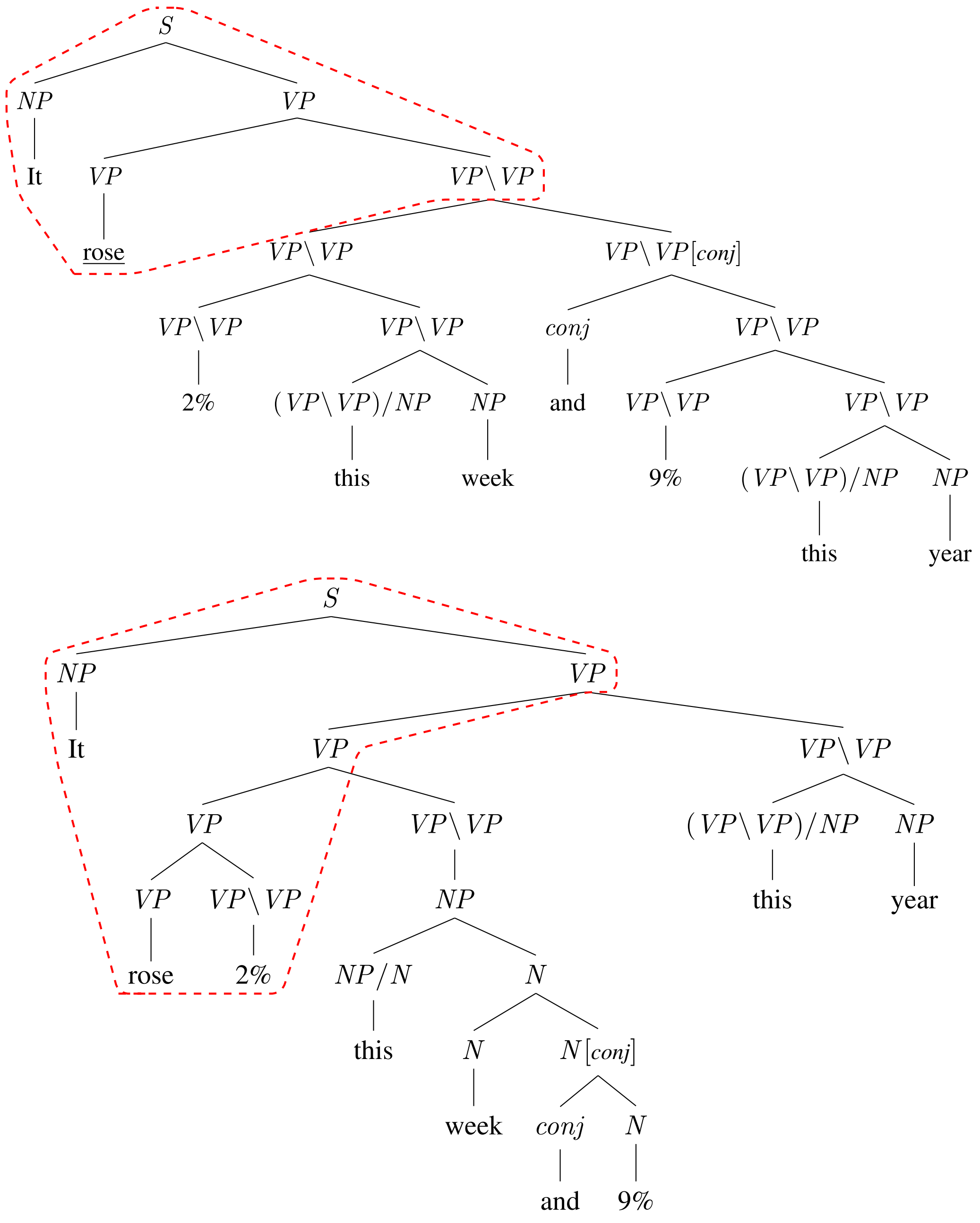


FIGURE 5.12: HeadTree feature for both the correct and incorrect parses.

Neighbours: like the Edge feature, the Neighbours feature attempts to represent poor tree attachment decisions. The feature encodes the category of each non-terminal, the binned number of leaf nodes in the subtree rooted at the non-terminal, and the POS tags of the ℓ_1 preceding words and the ℓ_2 following words if they exist, where $\ell_1 = 1$ or 2 and $\ell_2 = 1$. The buckets used for binning are 0, 1, 2, 4, or 5+ (following Charniak and Johnson (2005)). By encoding the size of the tree as well as additional leading and trailing context, this feature weights the importance of poor attachments by the number of words ultimately affected in the sentence.

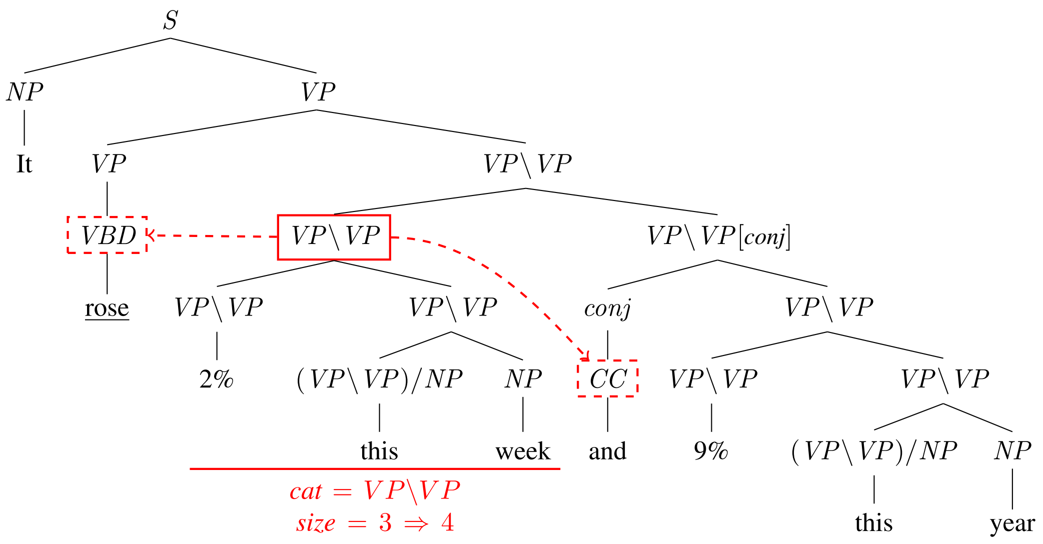


FIGURE 5.13: Neighbours features for the correct parse.

Word: yields each word in a sentence along with the categories of $\ell = 2$ or 3 of its immediate ancestor nodes in the tree. This gives some vertical context to each word in the tree, encoding the series of productions that use the word.

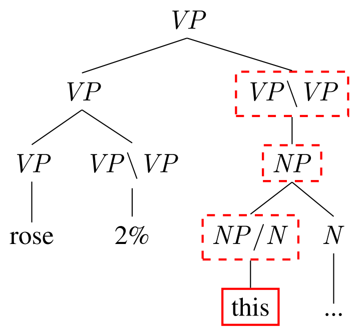
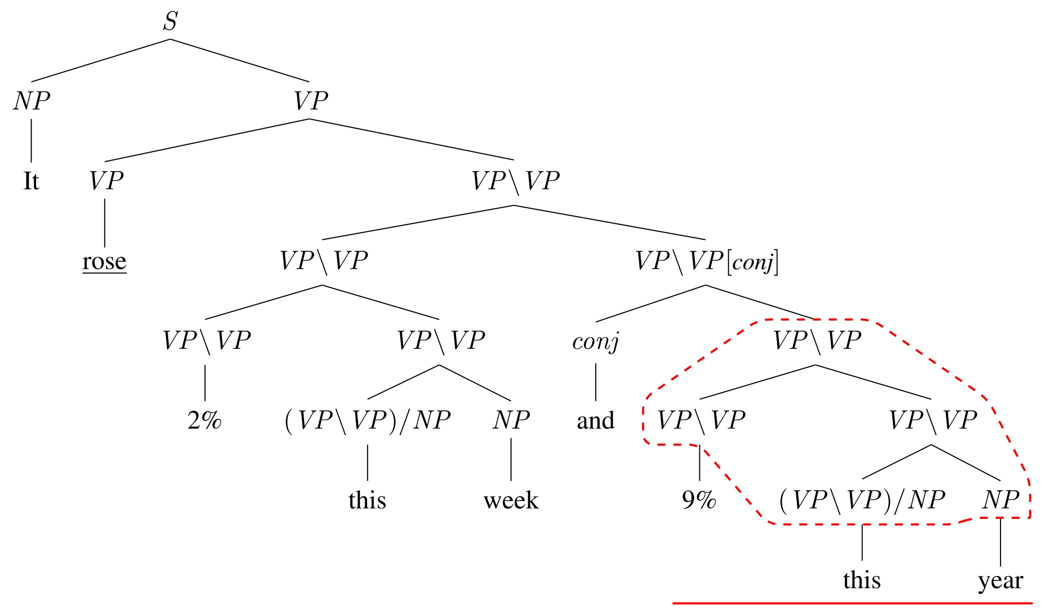


FIGURE 5.14: Word features for $\ell = 3$ ancestors in the incorrect tree.

NGram Tree: records tree fragments rooted at the lowest common ancestor node of $\ell = 2$ or $\ell = 3$ contiguous terminals in the tree. The contiguous words are known as n-*grams* . This feature represents the subtree encompassing each sequence of ℓ words in the sentence, identifying instances such as the second diagram below where there is an especially deep and complex structure encompassing the last three words of the example sentence (shown with a line underneath).



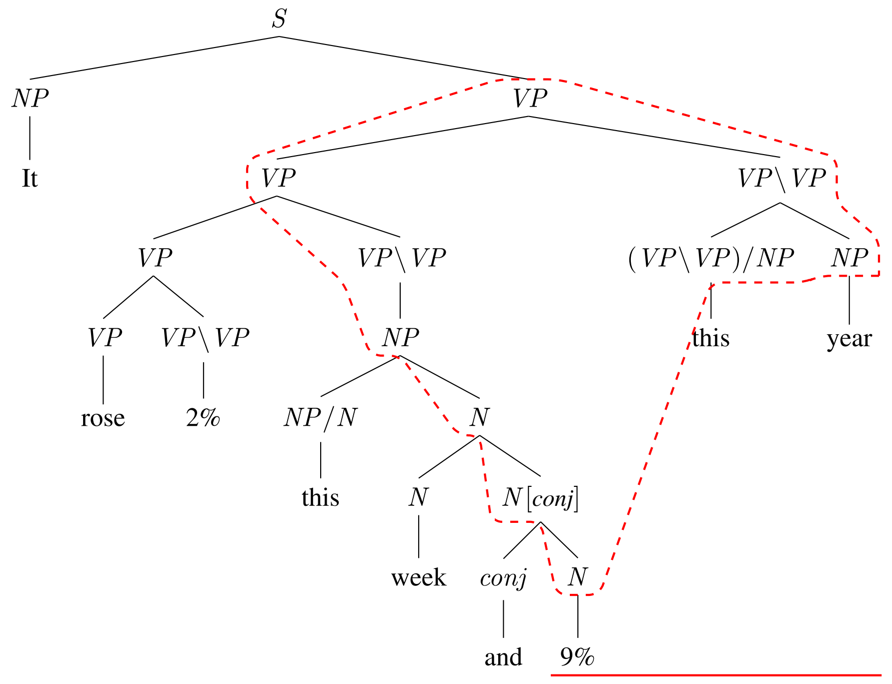


FIGURE 5.15: NGramTree features for $\ell = 3$ consecutive constituents (here the last three constituents in the sentence) for the correct and incorrect parses.

Syntactic-Semantic Heads (SynSemHeads): the *semantic head* of a subtree is the word which functions as the core with respect to meaning (e.g. the rightmost noun in a noun phrase, i.e. *the big red car*). The *syntactic head* (or *functional head*) is the function word which specifies the semantic head (e.g. the determiner in a noun phrase). When the two heads are distinct, they should subcategorise for each other, i.e. it is expected that good pairs of heads will appear together more often that poor pairs of heads. This feature yields pairs of semantic and syntactic heads found in the subtrees of each non-terminal in the parse. Heads are encoded as lexical items and POS tags for further generalisation.

Semantic heads are located using the C&C parser's head annotations. Functional heads are located with the same heuristic used by Charniak and Johnson (2005): find the leftmost child of each non-terminal that has a POS tag in a predetermined list of function word tags.

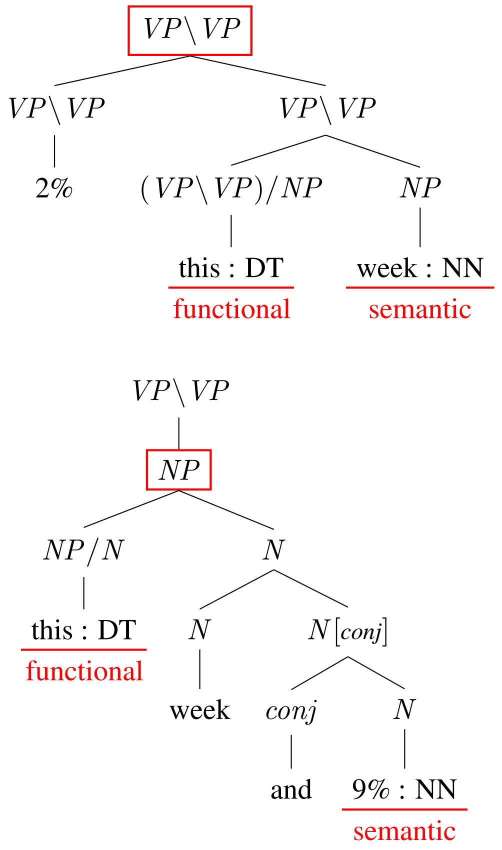


FIGURE 5.16: SynSemHeads features for fragments taken from the correct and incorrect parses.

Word Projection (WProj): This feature projects each word in the tree upwards until the child is no longer the head child of its parent (termed the *maximal projection*). This captures how important the subtree is in the context of the sentence. The word and the category of its maximal projection parent are encoded by the feature.

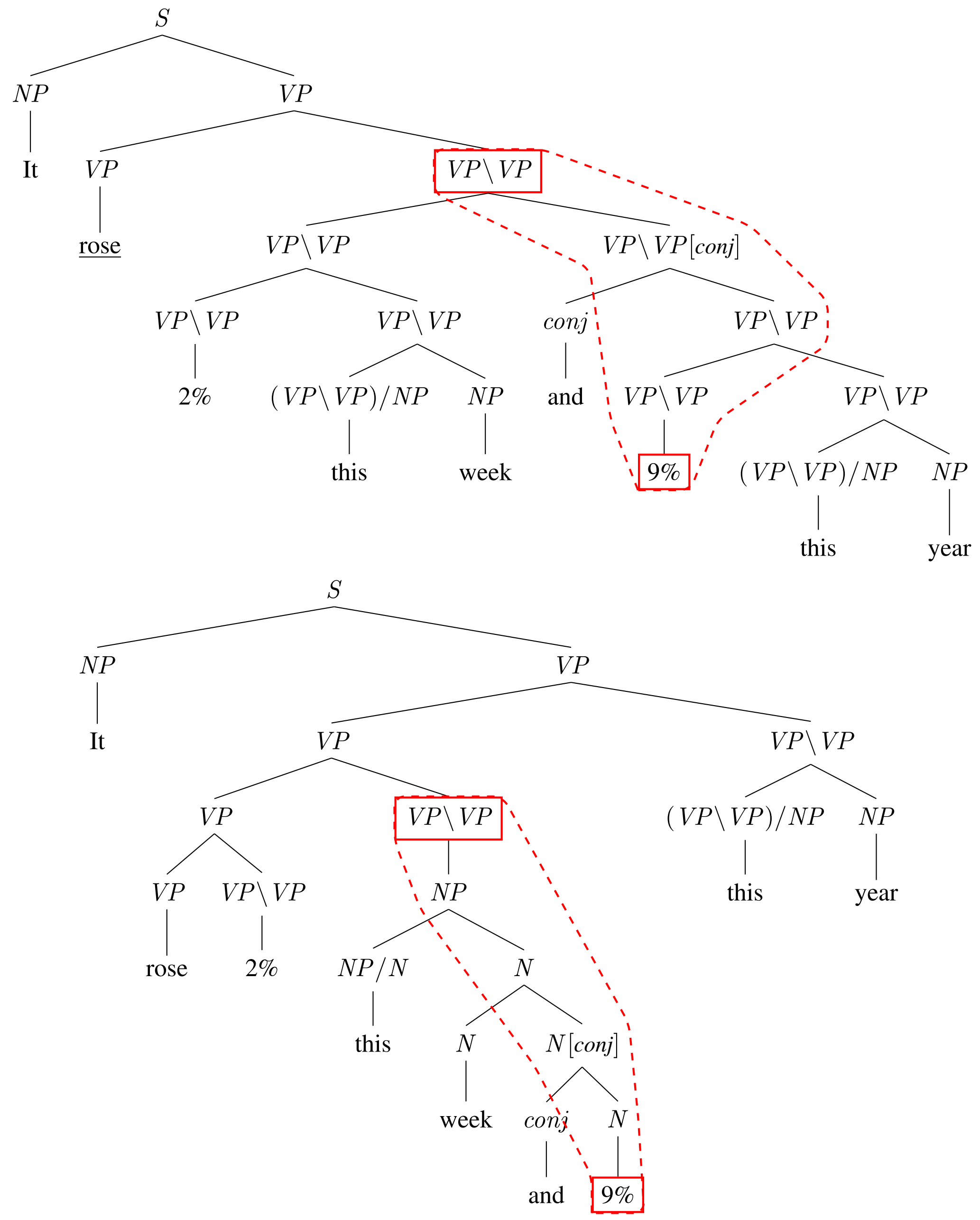


FIGURE 5.17: WProj features for both the correct and incorrect parses. The path to the maximal projection parent is boxed using dotted lines; the values encoded in the feature are indicated with solid boxes.

5.3 Results

We used Section 00 of CCGbank as the evaluation corpus for our initial implementation of the Charniak reranker for CCG. Results are recorded in Tables 5.2 and 5.3. We use the CCG labeled dependency metric as described in Section 4.2 to measure performance. Each table records labeled precision (LP), labeled recall (LR), and labeled F-score (LF) results over gold standard POS tags, and labeled F-score over automatically assigned POS tags (AF).

Our results include experiments using classification and regression, and values of 0, 2, and 5 for the pruning parameter t. For both 10-best and 50-best experiments we compare our reranking results against the baseline parser using the normal-form model, as well as a randomised baseline which selects a parse at random from the top n.

	t	LP	LR	LF	AF
Baseline	-	**87.19**	**86.32**	**86.75**	**84.80**
Random	-	83.40	81.97	82.68	81.10
Classification	0	**86.17**	**84.84**	**85.50**	**83.73**
	2	86.09	84.77	85.42	83.62
	5	85.93	84.57	85.25	83.50
Regression	0	**85.97**	**84.20**	**85.08**	**83.24**
	2	85.63	83.82	84.72	82.82
	5	85.54	83.75	84.63	82.77

TABLE 5.2: 10-best reranking performance on Section 00 of CCGbank for various combinations of pruning values t, and classification and regression experiments. Bolded scores are the highest for the approach.

	t	LP	LR	LF	AF
Baseline	-	**87.19**	**86.32**	**86.75**	**84.80**
Random	-	81.69	79.97	80.82	79.04
Classification	0	**85.76**	**84.27**	**85.01**	**83.36**
	2	85.68	84.10	84.88	83.17
	5	85.44	83.77	84.60	82.92
Regression	0	**85.09**	**82.89**	**83.97**	**82.18**
	2	84.80	82.49	83.63	81.97
	5	84.89	82.57	83.72	81.96

TABLE 5.3: 50-best reranking performance on Section 00 of CCGbank for various combinations of pruning values t, and classification and regression experiments. Bolded scores are the highest for the approach.

The randomised baselines show that choosing an arbitrary parse from the n-best list severely decreases performance. While the oracle score for the parser increases with larger n (see Section 3.4), the overall parse quality deteriorates. Both the 10-best and 50-best rerankers improved over the randomised scores by a significant margin on all metrics, indicating that reranking is more useful than simply choosing at random.

Unfortunately, parsing performance worsens by a significant amount in every experiment, even though we implemented features that have been used to achieve state-of-the-art reranking performance. The best labeled F-scores were 85.50% and 85.01% for 10-best and 50-best reranking respectively, against a baseline result of 86.75%. The classification experiments significantly outperformed regression by roughly 0.5% in all metrics. Pruning the feature space to remove less informative features did not produce a statistically significant difference in the classification experiment, though its effect was more pronounced in the regression setup.

The negative result has a number of implications. Firstly, even though the features we used were state-of-the-art, they have not proved useful for CCG reranking. The oracle scores shown in Section 3.4 demonstrate that there is a very large potential performance improvement that this reranker configuration is not capturing. Clearly, the current features are not adequately distinguishing between good and bad parses, and this is most likely due to their origin as features for reranking Penn Treebank parses against a bracketed string metric. We need to develop features that are better suited to CCG and its dependency evaluation instead of the Penn Treebank and PARSEVAL. To do this, we must first discover what errors are being made by the parser, and how we can address these in the reranker.

5.4 Summary

We have implemented a reranking system for CCG, and comprehensively documented the methodology and approach used to construct the system. We provide a clear description of the features used in the reranker of Charniak and Johnson (2005), and describe how they have been adapted for CCG reranking.

We have found that an initial reranker implementation using only the features described in Charniak and Johnson (2005) performs poorly and substantially worsens parsing accuracy. These results mean that we remain unsure whether reranking can work for CCG parsing. This motivates our analysis of errors in CCG parsing in Chapter 6, and the development and evaluation of new features based on these errors in Chapter 7.

CHAPTER 6

Analysis of Incorrect Parses

In this chapter, we perform an in-depth analysis of common classes of syntactic ambiguities and their impact on parsing accuracy. We examine the effect of incorrect prepositional phrase attachment, complement/adjunct distinctions, and coordination on dependency errors in the C&C parser. Our intention is to uncover errors that can be addressed through reranking, and in the process motivate the development of more informative CCG reranking features.

Our contribution is a detailed analysis of parsing error conducted on the C&C parser. We find that over 25% of dependency errors made by the parser is directly or indirectly attributable to prepositional phrase or conjunction errors. This finding motivates the development of reranker features in Chapter 7 that can capture these constructions in parse trees.

6.1 Prepositional Phrases in CCG

Prepositional phrases (*PP*s) are phrases which are headed by *prepositions* such as *with*, *by*, *to*, and *of*. Typically, *PP*s consist of the preposition and one noun phrase argument. We first introduced *PP*s in Section 2.2.1. In this chapter we describe them in more detail as the focus of an analysis of parse errors. Prepositional phrases are very common in English, and their flexibility in a sentence leads to many potential problems in a parse.

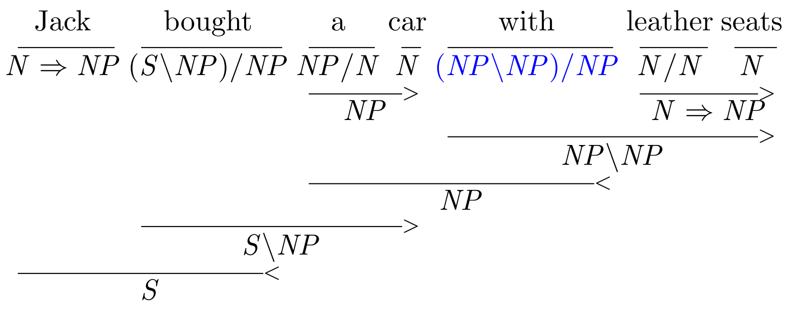
(11) Jack bought a car <u>**on**</u> **Friday.**

We described in Section 2.2.1 how prepositional phrase attachment can cause ambiguity in sentences, resulting in multiple possible constituent trees. This particular kind of ambiguity occurs when the prepositional phrase is an *adjunct* – a constituent that is *optional* in the sentence. Prepositional phrase adjuncts commonly describe the contextual circumstances of a situation or activity, such as in sentence (11). They may also be moved (or removed) from the sentence without affecting grammaticality, as in sentences 12 and 13.

(12) Jack bought a car **<u>with</u> leather seats**.

(13) Jack bought a car.

The optional nature of the PP creates a potential ambiguity in sentence (12), where the phrase *with leather seats* may correctly attach *car* or incorrectly attach to *bought*. In CCG, this attachment ambiguity manifests as *category ambiguity*, where different attachments create different category structures and derivations.

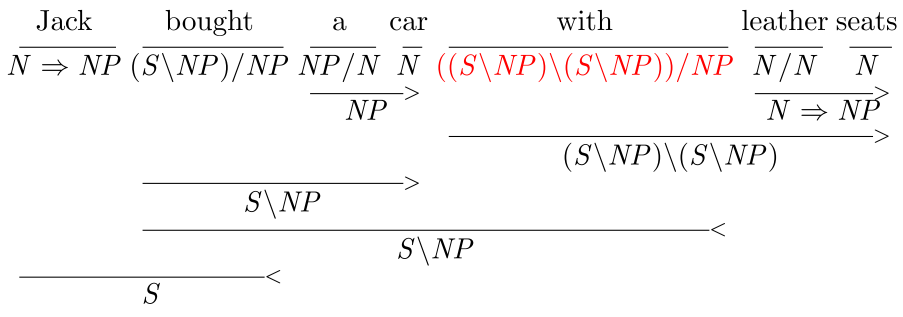


$\langle with,\ (NP\backslash NP)/NP,\ 1,\ car,\ -\rangle$
$\langle with,\ (NP\backslash NP)/NP,\ 2,\ seats,\ -\rangle$

FIGURE 6.1: A CCG derivation where the PP headed by *with* correctly attaches to *car*. The resulting dependencies headed by *with* are also shown. The attachment selection is indicated by the category in blue, which is looking for an NP.

Figure 6.1 depicts the correct derivation for sentence (12). In this derivation the word *with* is assigned the category $(NP\backslash NP)/NP$, indicating that it requires one noun phrase to its right, and when it finds that phrase it will return a *post noun-phrase modifier* $(NP\backslash NP)$. It is this result category that indicates the attachment to the noun *car*; it can be said that the attachment decision is encoded (*subcategorised* for) in the category of the preposition. The resulting dependencies indicate that *with* takes *seats* as one argument in slot 2, and *car* as the other argument in slot 1. The dependency to *car* is the attachment to the noun in this sentence.

Figure 6.2 shows an alternate derivation for sentence (12), where the prepositional phrase incorrectly attaches to the verb *bought*. Semantically, this means that the car was purchased while accompanied by leather seats, which as a human reader we know is highly unlikely. The only difference between the derivations is the category assigned to *with*, which is now $((S\backslash NP)\backslash(S\backslash NP))/NP$. The structure



$$\langle \textit{with}, ((S\backslash NP)\backslash(S\backslash NP))/NP, 2, \textit{bought}, -\rangle$$
$$\langle \textit{with}, ((S\backslash NP)\backslash(S\backslash NP))/NP, 3, \textit{seats}, -\rangle$$

FIGURE 6.2: A CCG derivation where the PP headed by *with* incorrectly attaches to *bought*. Both dependencies headed by *with* are now incorrect against the gold standard.

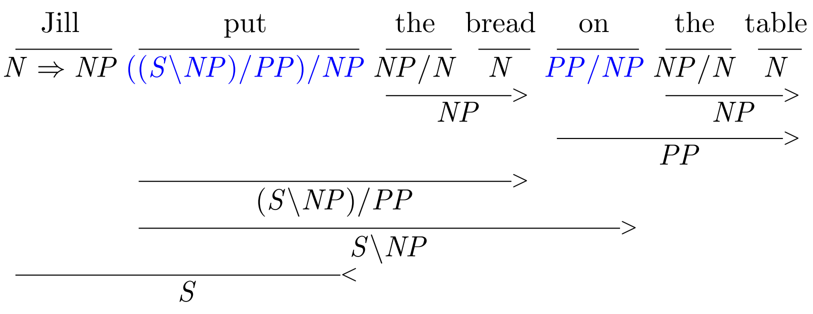
of this new category is almost identical to the correct one, but it is now a *post-verbal modifier* that is looking to attach to a verb rather than a noun. The dependency between *with* and *car* no longer exists in this derivation; it has been replaced with a dependency to *bought* indicating the different choice of attachment. The slot numbering in the dependency has also changed due to the presence of the complex verb category; the correctly attached NP argument *seats* has shifted from slot 2 in the dependency to slot 3.

Errors in PP attachment in the derivation propagate into the evaluation as *incorrect head categories, slots*, and *arguments* in the dependencies headed by the prepositions. The correct dependencies in Figure 6.1 are invalidated by the incorrect head category for *with*, the incorrect choice of attachment point, and the incorrect argument slot in Figure 6.2. As prepositional phrases are usually involved in at least two dependencies (one to create the PP, and one for the attachment), incorrect attachments are heavily penalised in the CCG dependency evaluation and can severely reduce accuracy.

6.1.1 Prepositional Phrase Complements

(14) Jill put the bread **on the table**.

In contrast to adjuncts, *complements* are constituents that are necessary to complete the meaning of a sentence. For example, the bolded prepositional phrase in sentence (14) is required or else the sentence is not grammatical – items cannot merely be “put” in English. Complements, unlike adjuncts, usually cannot be moved or removed as they are needed for grammaticality.



$$\langle \textit{on},\ PP/NP,\ 1,\ \textit{table},\ -\rangle$$
$$\langle \textit{put},\ ((S\backslash NP)/PP)/NP,\ 2,\ \textit{on},\ -\rangle$$

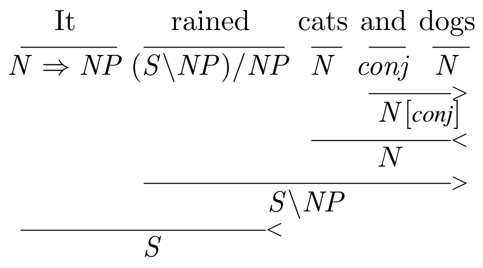
FIGURE 6.3: A CCG derivation where the PP headed by *on* is a complement. The main verb *put* subcategorises for the PP in its category and creates a dependency for it.

In CCG, the obligatory nature of PP complements is encoded in the categories for *both* the head of the *predicate* in the sentence (usually the main verb) and the child PP. In Figure 6.3, the category for the verb *put* requires one argument that is a PP, indicating the necessity of the complement. The category for the preposition *on* changes to return a PP that will be found by the verb. This is in contrast to the adjunct case in Figures 6.1 and 6.2, where the category of the corresponding verb and preposition do not stipulate a mandatory PP.

The corresponding dependencies in Figure 6.3 show how a complement PP is the *child* of a dependency headed by the constituent it attaches to. In the adjunct cases of Figures 6.1 and 6.2, it is the preposition that is the head of the corresponding dependency. Incorrect distinction between PP complements and adjuncts can create multiple errors in the dependency output of the parser; in the worst case, every single dependency that contains the subcategorising verb or PP will be incorrect due to an wrong head category.

6.2 Coordination and CCG Dependencies

Coordination occurs when two phrases are linked with a conjunction such as *and*. When coordination is present in a sentence, two constituents X and Y are combined into a single constituent Z that behaves as if it were one item in the tree. However, whenever Z is part of a dependency in the tree, there will be *two* dependencies generated with the same slot and head category: one where X appears, and another where Y appears. This duplication occurs both in the case where Z is the head of the dependency and when it is the child; duplicates for X and Y will appear in the corresponding head or child position of the



$\langle$*and*, $conj$, 1, *cats*, $-\rangle$

$\langle$*and*, $conj$, 1, *dogs*, $-\rangle$

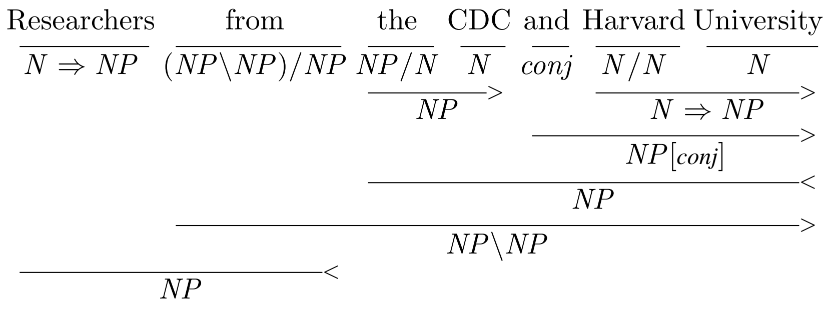
$\langle$*rained*, $(S\backslash NP)/NP$, 1, *It*, $-\rangle$

$\langle$*rained*, $(S\backslash NP)/NP$, 2, *cats*, $-\rangle$

$\langle$*rained*, $(S\backslash NP)/NP$, 2, *dogs*, $-\rangle$

FIGURE 6.4: A CCG derivation and the corresponding dependencies created by coordination. $conj$ dependencies link the coordinated words, and pairs of dependencies with the slot exist for each coordinated constituent.

dependency tuple. Additionally, a pair of $conj$ dependencies will be created between the conjunction and the two combined constituents. The relationship between coordination and dependencies is illustrated in Figure 6.4 using the binarised CCGbank coordination rules (see Section 3.2).



$\langle$*and*, $conj$, 1, *CDC*, $-\rangle$

$\langle$*and*, $conj$, 1, *University*, $-\rangle$

$\langle$*the*, NP/N, 1, *CDC*, $-\rangle$

$\langle$*the*, NP/N, 1, *University*, $-\rangle$

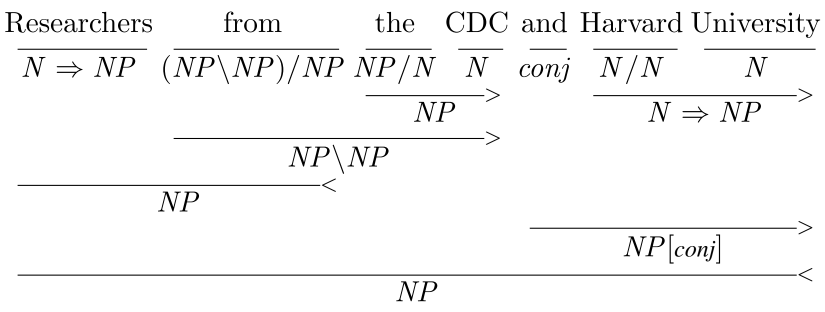
$\langle$*from*, $(NP\backslash NP)/NP$, 1, *Researchers*, $-\rangle$

$\langle$*from*, $(NP\backslash NP)/NP$, 2, *CDC*, $-\rangle$

$\langle$*from*, $(NP\backslash NP)/NP$, 2, *University*, $-\rangle$

FIGURE 6.5: A CCG derivation where *the CDC* and *Harvard University* are correctly coordinated.

There are two main causes of coordination errors in CCG parsing. The first occurs where the categories assigned to the coordination structure are correct, but combined incorrectly in the derivation. Figures 6.5



$\langle$*and*, *conj*, 1, *Researchers*, $-\rangle$

$\langle$*and*, *conj*, 1, *University*, $-\rangle$

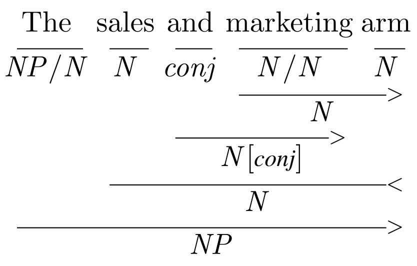
$\langle$*the*, NP/N, 1, *CDC*, $-\rangle$

$\langle$*from*, $(NP\backslash NP)/NP$, 1, *Researchers*, $-\rangle$

$\langle$*from*, $(NP\backslash NP)/NP$, 2, *CDC*, $-\rangle$

FIGURE 6.6: A CCG derivation where *Researchers* and *Harvard University* are incorrectly coordinated.

and 6.6 depict a coordination within a prepositional phrase; in Figure 6.5, the coordination is correctly located within the PP, but in Figure 6.6 the coordination has been moved outside the PP to conjoin two higher-level constituents. This leads to the semantically awkward (but syntactically valid) conjunction of *Researchers* and *Harvard University* rather than the correct conjunction of *the CDC* and *Harvard University*.



$\langle$*and*, *conj*, 1, *sales*, $-\rangle$

$\langle$*and*, *conj*, 1, *arm*, $-\rangle$

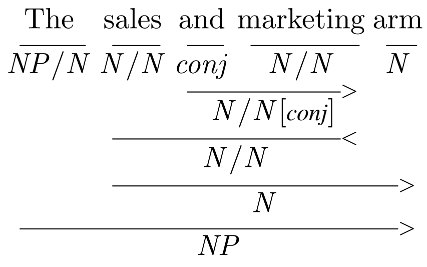
$\langle$*marketing*, N/N, 1, *arm*, $-\rangle$

$\langle$*The*, NP/N, 1, *sales*, $-\rangle$

$\langle$*The*, NP/N, 1, *arm*, $-\rangle$

FIGURE 6.7: A CCG derivation where *sales* is incorrectly assigned the category N. This results in an incorrect coordination between *sales* and *arm* which propagates through the dependencies.

As with prepositional phrase attachment errors, coordination errors propagate and can invalidate or spuriously duplicate every dependency in which they are involved. The incorrect derivation is missing nearly half of the dependencies found in the correct derivation, and has introduced an additional incorrect dependency (indicated in red).



$$\langle \textit{and},\ conj,\ 1,\ \textit{sales},\ -\rangle$$
$$\langle \textit{and},\ conj,\ 1,\ \textit{marketing},\ -\rangle$$
$$\langle \textit{sales},\ N/N,\ 1,\ \textit{arm},\ -\rangle$$
$$\langle \textit{marketing},\ N/N,\ 1,\ \textit{arm},\ -\rangle$$
$$\langle \textit{The},\ NP/N,\ 1,\ \textit{arm},\ -\rangle$$

FIGURE 6.8: A CCG derivation where *sales* and *marketing* are correctly coordinated.

The second form of coordination error occurs due to an incorrect category assignment. In CCG, constituents that do not bear the same category cannot be coordinated. Category error can force the parser to produce semantically unnatural coordinations, such as that in Figure 6.7. The only difference between the correct derivation in Figure 6.8 and the incorrect derivation is the category assigned to *sales*; however, this forces a different coordination structure as an N cannot be conjoined with an N/N. Again, this results in a large number of errors that sum over each appearance of the coordination in a dependency.

6.3 Analysing Parser Error

We analyse the impact of PP and coordination errors on the C&C parser by parsing Section 00 of CCGbank, and then comparing the dependency output to the gold standard. We detect the various kinds of prepositional and coordination errors described by comparing each dependency generated by the parser against the gold standard. If the dependency is not in the gold standard, we attempt to permute the head category, slot, and argument word to detect the types of errors that had occurred. We specifically examined dependencies headed by prepositions, duplicate dependencies resulting from coordination, and dependencies headed by a predicate (verbal) category. All other dependency errors are grouped into a separate class.

Head Category	Error Type	Frequency	Sum
Coordination	Duplicate dependency	320	320
Predicate	Slot incorrect	4	1688
	Head category incorrect	679	
	Head category and slot incorrect	282	
	Other	723	
Prepositional Phrase	Slot incorrect	0	1604
	Head category incorrect	85	
	Head category and slot incorrect	753	
	Other	766	
Non PP/coordination/predicate	Slot incorrect	1	1401
	Head category incorrect	218	
	Head category and slot incorrect	325	
	Other	857	
Total		**5013**	**5013**

TABLE 6.1: Breakdown of parser errors by type over Section 00 of CCGbank. This table only analyses dependencies produced by the parser.

The results of our analysis are depicted in Table 6.1. We break down our results into a number of different classes based on the head of the erroneous dependency. Coordination errors are detected when they result in duplicate dependencies. All other errors are classified based on the permutations required to "fix" the dependency, i.e. transform the incorrect dependency into a dependency present in the gold standard. The permutations we tried included permuting the argument slot (*Slot incorrect*), assigning the correct head category (*Head category incorrect*), and assigning the correct head category and permuting the argument slot (*Head category and slot incorrect*). If these permutations failed we classified the error as 'Other'.

Notably, we find that over 30% of dependency errors made by the parser are headed by PPs, while duplicate coordination dependencies account for another 6% of errors. Errors headed by predicates account for a further 30%. Independently, PP errors and predicate errors outnumber all other types of dependency errors.

We further investigated the set of PP errors where the head category and the slot were incorrect (which comprises almost half of all PP errors). This is the class of error that captures the distinction between PPs attaching to verbs ($(S\backslash NP)\backslash(S\backslash NP)$), nouns ($NP\backslash NP$), and standing alone as a PP complement (PP). Specifically, where the head category or slot of the PP was incorrect, we classified the error based on what the *result* of the head category should have been. All errors in this class with less than 10 occurrences are grouped together into an 'Other' category and are not independently examined.

Incorrect Result Category	Correct Result Category	Frequency	Sum
$(S\backslash NP)\backslash(S\backslash NP)$	$NP\backslash NP$	143	283
	PP	122	
	Other	18	
$NP\backslash NP$	$(S\backslash NP)\backslash(S\backslash NP)$	129	231
	PP	66	
	Other	36	
PP	$(S\backslash NP)\backslash(S\backslash NP)$	138	174
	$NP\backslash NP$	34	
	Other	12	
Other		55	55
Total		**753**	**753**

TABLE 6.2: Breakdown of parser errors on PP dependencies by type over Section 00 of CCGbank. Errors occurring less than 10 times are grouped into the 'Other' class.

Table 6.2 classifies the incorrect PP dependencies where the head category and slot are wrong. In the table, an error in distinguishing between a complement PP and adjunct PP occurs when the incorrect result category is a (verb or noun) modifier and the correct result category is a standalone PP, or vice versa. An attachment error has occurred when both the correct and incorrect result categories are modifiers. Ignoring the 'Other' classes, there are at least 360 incorrect complement/adjunct distinctions in Table 6.2 and 272 attachment errors; these account for 7% and 5% of the total dependency errors respectively. Independently, PP errors already account for a significant proportion of the total errors made by the parser.

Complement/adjunct distinction errors also imply that the predicate category is incorrect, as complements must be subcategorised for in the verb category. We examined two classes of predicate dependency errors: where the head category was incorrect, and where the head category and slot where incorrect. Specifically, we classified the errors by whether they were caused by a PP that was missing or incorrectly present in the head category. For example, if the parser assigns the category of $(S\backslash NP)/NP$ to a verb, but the gold category is $((S\backslash NP)/PP)/NP$, this means that a complement/adjunct mistake has been made and all dependencies headed by the verb will be incorrect *as well as* those headed by the preposition. This error will also result in an incorrect argument slot for the final NP argument, which is why we examine the chosen two error classes.

The predicate analysis is summarised in Table 6.3. Bolded rows indicate cases where a PP should be present in the head category or is incorrectly present. We can see that almost half of the errors in the

Error type	PP in gold cat	PP in parser cat	Frequency	Sum
Incorrect head category	YES	YES	52	679
	YES	**NO**	**152**	
	NO	**YES**	**132**	
	NO	NO	343	
Incorrect head category and slot	YES	YES	17	282
	YES	**NO**	**94**	
	NO	**YES**	**62**	
	NO	NO	109	
Total			**961**	**961**

TABLE 6.3: Breakdown of parser errors on predicate dependencies by type over Section 00 of CCGbank. Bolded rows indicate errors due to missing or incorrect PP arguments in the predicate category (e.g. $(S\backslash NP)/NP$ -> $((S\backslash NP)/PP)/NP)$.

table (440 in total) are due to incorrect PPs in the head category, showing how PP errors propagate through to other dependencies.

6.4 Summary

Head category	Error type	Frequency	Sum
Coordination	Duplicate dependency	320	320
Predicate	Complement/Adjunct distinction	440	440
Prepositional Phrase	Attachment error	272	632
	Complement/Adjunct distinction	360	
Total		**1392 / 5013**	**1392 / 5013**

TABLE 6.4: Summary of errors caused by PP coordination errors as a proportion of total parser errors in Section 00 of CCGbank.

Table 6.4 summarises the errors that we have found. Prepositional phrase attachment, complement/adjunct distinction, and coordination all create problems for the C&C parser in the dependency evaluation, and these errors all propagate through into other dependencies. Our analysis has revealed that over 25% of the errors made by the parser can be attributed directly or indirectly to one of these three problems.

Our analysis motivates features for CCG reranking that attempt to capture these classes of error and distinguish between them. We will go one to describe the features we developed for this task in Chapter 7.

CHAPTER 7

Improved CCG Reranking

We showed in Chapter 5 how the reranking features of Charniak and Johnson (2005) resulted in a substantial decrease in parsing accuracy for the C&C parser. Following our analysis of parser errors in Chapter 6, we discuss in this chapter new features for the CCG reranker, and convincingly demonstrate how they contribute to improvements in parsing accuracy.

7.1 Better Features for CCG Reranking

CCG derivation trees have some important structural differences from the Penn Treebank trees targeted by Charniak and Johnson (2005). The most important difference is that CCG trees are at most binary branching. As the longest non-terminal in the Penn Treebank has 51 children, features designed to generalise long production rules are useful in the Charniak reranker but less relevant to CCG trees.

Another important difference is that CCG productions are constrained by the combinatory rules, whereas Penn Treebank productions combine unrelated atomic symbols. For instance, a Penn Treebank production NP→ NP PP would be translated into CCG as $NP \rightarrow NP\ NP\backslash NP$. In the CCG production, backward application is used to take the preposition's argument, NP, and yield its result, NP. Much of the information in the production is already present in the structure of the $NP\backslash NP$ category. We speculate that this will make the features that capture the vertical context of a rule less useful for CCG.

The features described in this section will continue to use the example parses in Figures 5.1 and 5.2, though some features use new examples where necessary for prepositional phrase constructions.

7.1.1 CCG-specific Features

We devised a number of new reranking features for CCG aimed at uncovering various combinator sequences or combinations that may indicate an overly complicated or undesirable derivation.

Coordination Heads (CoHeads): the coordination-related features described in Section 5.2.1 focus on capturing the structure and balance in coordinations, but ignore the dependencies formed by the structure. The base CoHeads feature encoded the heads of both halves of a coordination as lexical items and as part-of-speech tags to describe what constituents were being coordinated. Additionally, we encoded this base feature with the depths at which the heads were found in the subtrees of the coordination to measure the branching within both sides.

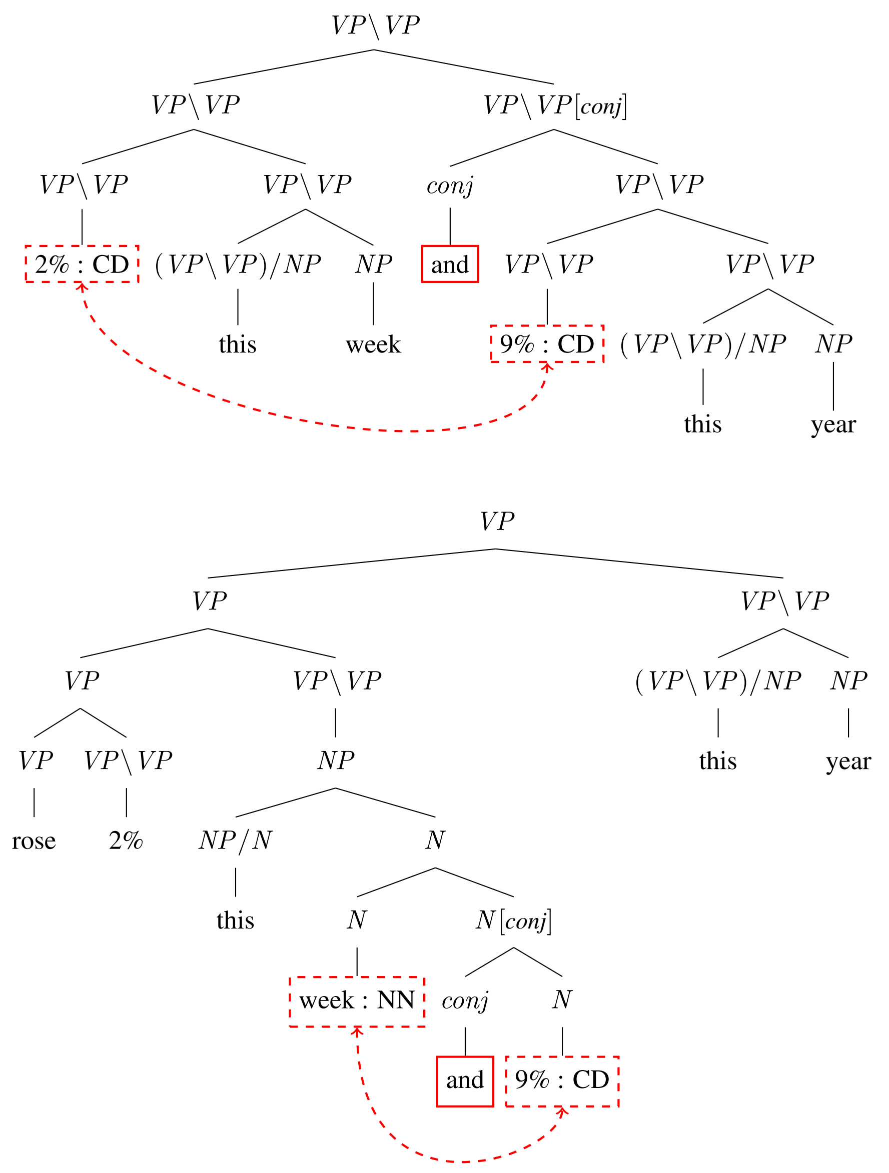


FIGURE 7.1: CoHeads features for both the correct and incorrect parses.

Coordination Duplicates (CoDup): we described in Section 6.2 how coordination in CCG results in pairs of dependencies being generated for each side of the conjunction. Poor coordination choices leads to duplicated but incorrect dependencies. This feature takes each pair of dependencies that contain a coordinated pair of words and encodes a tuple of $(other, conj\_1, conj\_2)$, where $other$ is the non-coordinated word common across the dependencies. Three features that use the lexical item, part-of-speech tag, and CCG category of $other$ are generated for each dependency pair.

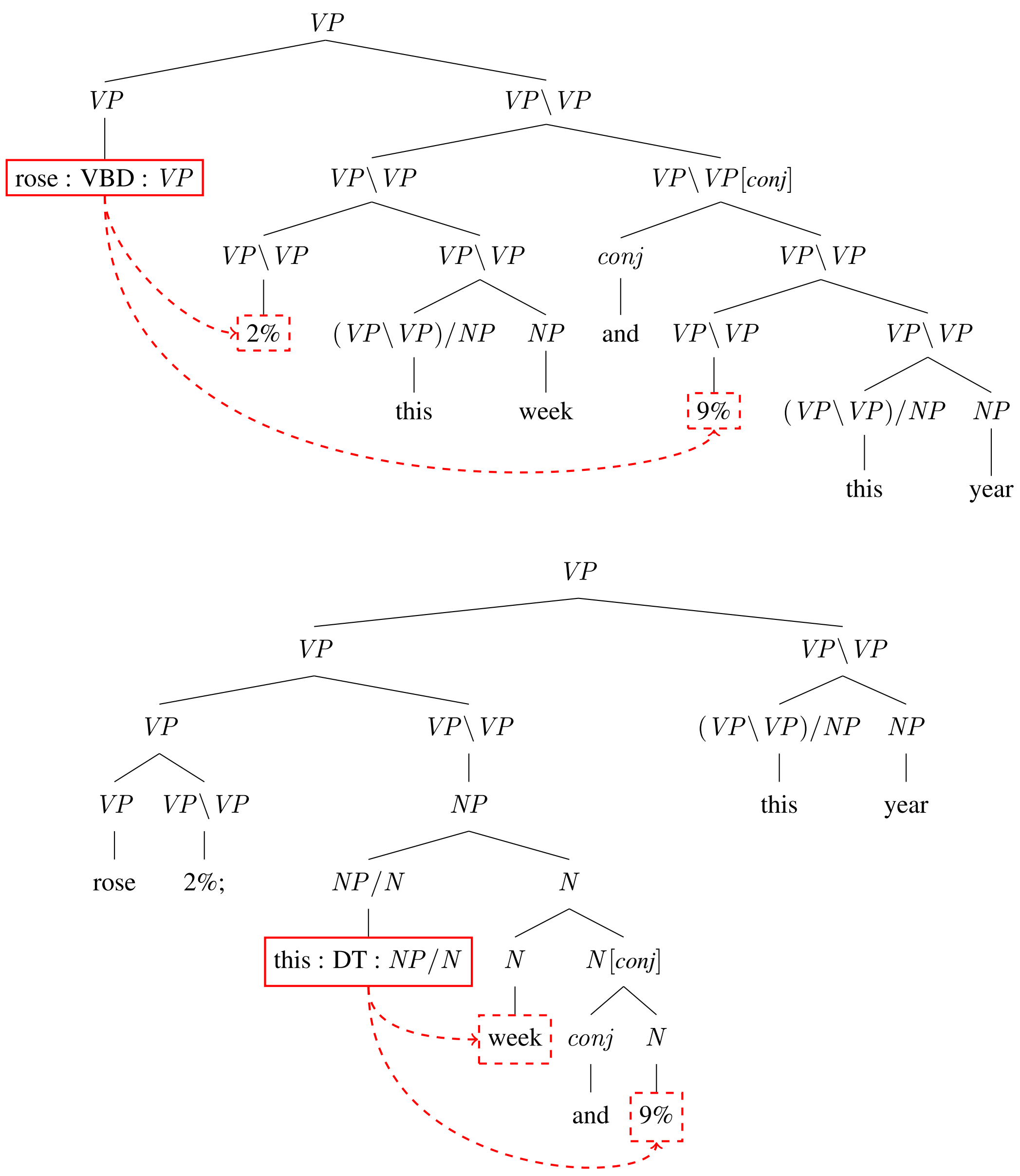


FIGURE 7.2: CoDups features. In the upper parse tree, *rose* is the argument of a pair of dependencies from the conjoined words *2%* and *9%*. In the lower parse tree, *week* and *9%* are the arguments of a pair of dependencies headed by *this*.

Preposition Result-Argument: in the parser model, prepositional phrase attachment choices are evaluated purely by considering the preposition and the attachment point; there is no consideration of the other words in the PP (typically a noun phrase). This is a significant shortcoming as there are many attachments which can be ruled out as unlikely due to the implied relationship between the words involved. For example, it is extremely unlikely that a *car* will be purchased while accompanied by *leather seats*, but much more likely that the *car* will contain *leather seats*.

This feature is active for all dependencies in the tree involving a prepositional phrase adjunct. The category of the attachment point of the PP (i.e. the result of the preposition's category) along with the attachment word is encoded with the heads of all other arguments in the PP.

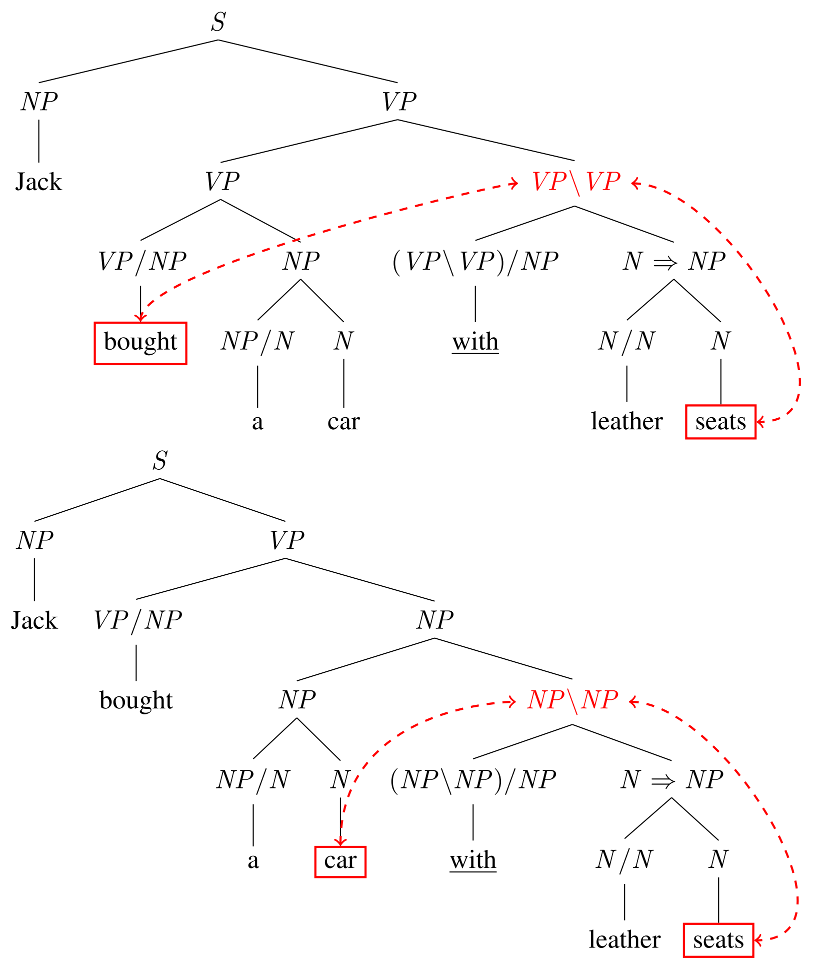


FIGURE 7.3: PrepResArg features for the prepositional phrase headed by *with*. Marked nodes are the categories and words encoded in the feature.

Complement-Adjunct: this feature encodes each prepositional phrase in a sentence in terms of its attachment point, the head preposition, and its arguments. This feature extends on the PrepResArg feature described in Figure 7.3 by being active for PP complements as well as adjuncts. In the complement case, the attachment point is found by locating the dependency which subcategorises for the PP phrase, while in the adjunct case the attachment point is a dependency headed by the PP itself.

(15) Jill *put* the bread on **the table** of **maple**.

The attachment point, preposition, and other arguments are encoded as lexical items, POS tags, and categories for generalisation and distinction between complement and adjunct assignment in parses.

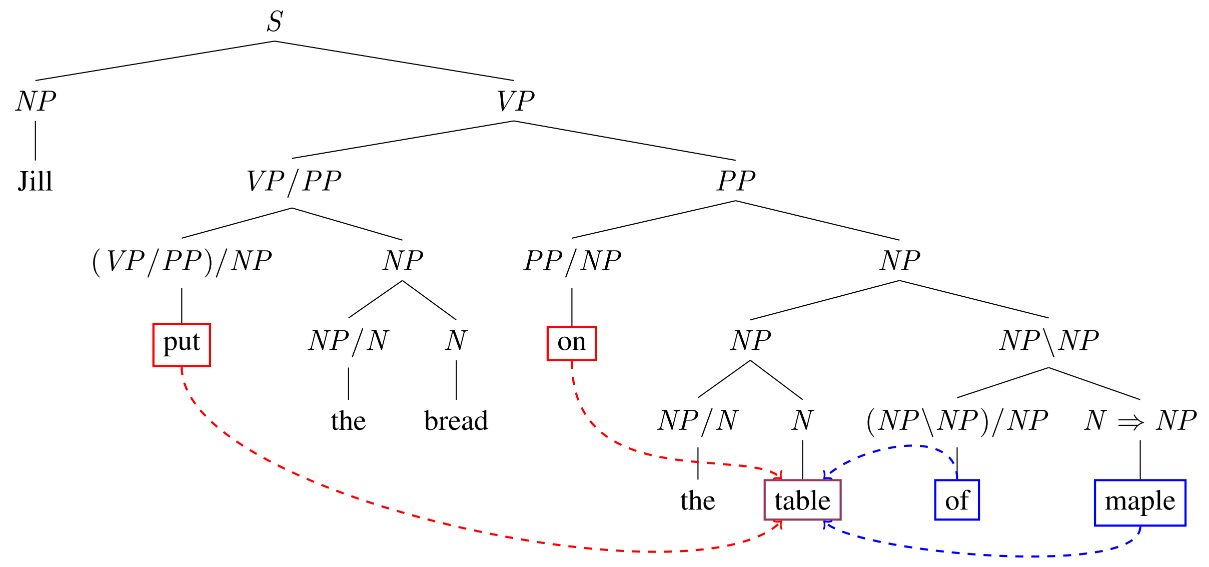


FIGURE 7.4: CompAdj features for the prepositional phrases headed by *on* and *of*. The words are colour-coded to indicate which feature they belong to.

Lexical Dependencies (LexDep): the topology of the parse tree partially captures the CCG dependencies licensed by the tree. This feature is active for non-terminals with two children and encodes the heads of the children as combinations of word-word, word-POS, POS-word, and POS-POS. Additional features encoding the categories of the words and the depth from the parent non-terminal are also generated. Dependencies involving punctuation are ignored as these are not assessed in the evaluation.

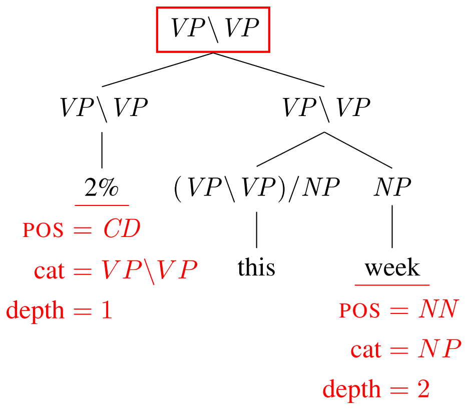


FIGURE 7.5: LexDep features for a fragment of the correct parse.

NumDeps: this feature distinguishes parses based on the number of dependencies that they yield (ignoring punctuation). Dependencies are located using the same process as the LexDep feature: all non-terminals in the parse tree that have two children are assumed to yield one dependency.

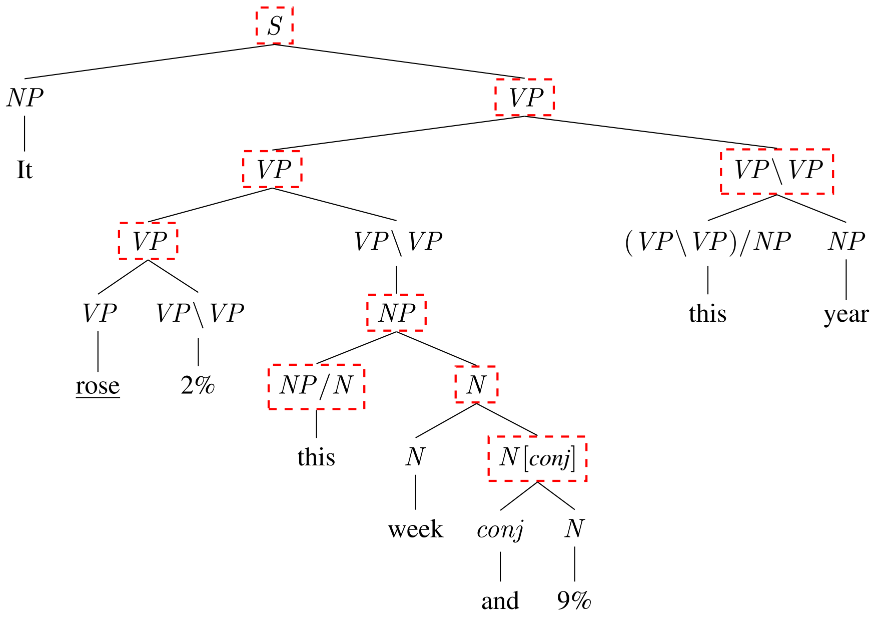


FIGURE 7.6: NumDeps features – marked nodes have two children and are counted as contributing one dependency to the sum.

Unary Rule features: this feature is active if any unary rules are used in the tree. Unary rule application is detected by identifying non-terminals in the tree with only one child. For each type of unary rule present in the tree, an indicator feature for the rule itself and for all pairs of the rule with other unary rules in the tree are generated. If only one unary rule is used, then no paired features are generated.

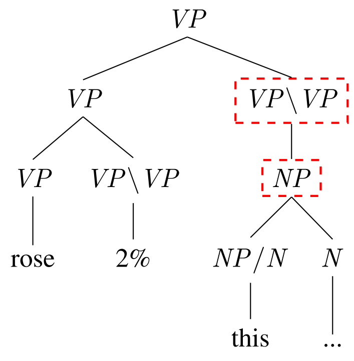


FIGURE 7.7: Unary rule application in the incorrect tree.

Type-Raising: this feature is active if unary type-raising is present in the tree. It is important to note that the nonstandard unary rules introduced in CCGbank (see Section 3.2) are not included as applications of type-raising. Thus, the unary rule application described in Figure 7.7 is distinct from this type-raising feature.

The type-raising combinator is defined in Section 3.1.2 as a unary rule application that transforms a category into a functor. The transformation retains the original category but reverses the result / argument structure as follows:

$$X \Rightarrow \mathrm{T}/(\mathrm{T}\backslash X) \quad (>\mathbf{T})$$

$$X \Rightarrow \mathrm{T}\backslash(\mathrm{T}/X) \quad (<\mathbf{T})$$

FIGURE 7.8: Type-raising combinators.

Category n-gram features: the reranking framework allows us to consider features from outside the sentence as well as within. This is not possible for the parser model due to dynamic programming, and so we can the averaged conditional probability of category n-grams in an external corpus as a feature.

The external corpus used for generating category n-gram counts was 4.5 million CCG-parsed sentences from the North American News Corpus NANC, described by Kummerfeld et al. (2010). We iterated over the sentences and counted the number of times each 2-, 3-, 4-, and 5-gram of categories occurred. Using these counts, we then calculated the probability of each category n-gram $c_1 ... c_n$ in the sentence conditioned as follows:

$$P(c_1...c_n) = P(c_n \mid c_1...c_{n-1})$$

We took the sum of the log probabilities for each n-gram in the sentence, and averaged it over the number of categories in the sentence to form our final feature value. We calculated the averaged log probability for each of 2-, 3-, 4- and 5-grams of categories in the sentence. A boolean marker feature for each length was also generated if any n-gram of that length in the sentence had a probability of 0.

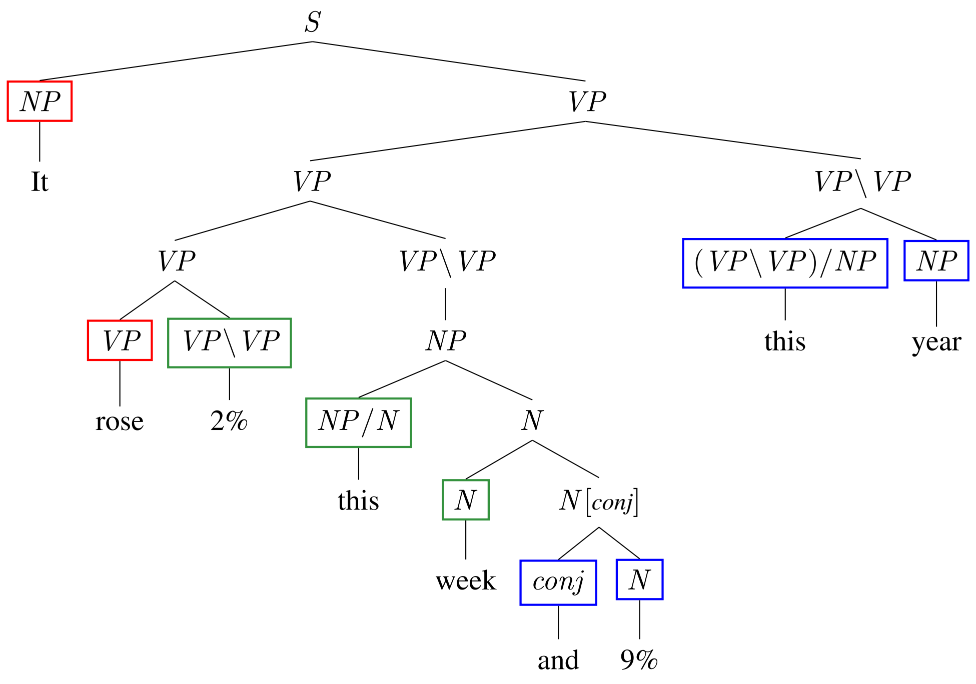


FIGURE 7.9: Category 2-grams (red), 3-grams (green), and 4-grams (blue) in the tree.

C&C Features: Finally, we also incorporate the features used in the dependency and normal-form models for the C&C parser. These features encode various combinations of word-category, word-POS, root-word, CCG rule, distance, and dependency information. Including these features gives the reranker the same information that the parser uses to produce the n-best list in addition to the topology and contextual features. For a full description of these features, see Clark and Curran (2007b).

The *distance* referred to in some of the features is calculated between the head words of the dependent words or the head words. Distance is expressed in terms of the number of *words*, *verbs*, and *punctuation symbols*.

Feature type	Example from correct parse tree in Figure 5.1
Category + Word	NP + It
Category + POS	NP + PRP
Root Category	$S[dcl]$
Root Category + Word	$S[dcl]$ + rose
Root Category + POS	$S[dcl]$ + VBD
Rule	$S[dcl] \rightarrow NP\ S[dcl] \backslash NP$
Rule + Word	$S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ + rose
Rule + POS	$S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ + VBD

FIGURE 7.10: Features common to the C&C dependency and normal-form models.

Feature type	Example from correct parse tree in Figure 5.1
Word-Word	$\langle$*rose*, $S\backslash NP_1$, 1, *It*, $-\rangle$
Word-POS	$\langle$*rose*, $S\backslash NP_1$, 1, *PRP*, $-\rangle$
POS-Word	$\langle$*VBD*, $S\backslash NP_1$, 1, *It*, $-\rangle$
POS-POS	$\langle$*VBD*, $S\backslash NP_1$, 1, *PRP*, $-\rangle$
Word + Distance(words)	$\langle$*rose*, $S\backslash NP_1$, 1, $-\rangle$ + 0
Word + Distance(punct)	$\langle$*rose*, $(S\backslash NP_1)$, 1, $-\rangle$ + 0
Word + Distance(verbs)	$\langle$*rose*, $(S\backslash NP_1)$, 1, $-\rangle$ + 0
POS + Distance(words)	$\langle$*VBD*, $S\backslash NP_1$, 1, $-\rangle$ + 0
POS + Distance(punct)	$\langle$*VBD*, $(S\backslash NP_1)$, 1, $-\rangle$ + 0
POS + Distance(verbs)	$\langle$*VBD*, $(S\backslash NP_1)$, 1, $-\rangle$ + 0

FIGURE 7.11: Features specific to the C&C dependency model.

Feature type	Example from correct parse tree in Figure 5.1
Word-Word	$\langle$*week*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP$, *this*$\rangle$
Word-POS	$\langle$*week*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP$, *DT*$\rangle$
POS-Word	$\langle$*NN*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP$, *this*$\rangle$
POS-POS	$\langle$*NN*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP$, *DT*$\rangle$
Word + Distance(words)	$\langle$*this*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP\rangle$ + 0
Word + Distance(punct)	$\langle$*this*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP\rangle$ + 0
Word + Distance(verbs)	$\langle$*this*, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP\rangle$ + 0
POS + Distance(words)	$\langle$DT, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP\rangle$ + 0
POS + Distance(punct)	$\langle$DT, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP\rangle$ + 0
POS + Distance(verbs)	$\langle$DT, $VP\backslash VP \rightarrow (VP\backslash VP)/NP\ NP\rangle$ + 0

FIGURE 7.12: Features specific to the C&C normal-form model.

7.2 Results

We present results in Tables 7.1 and 7.2 comparing the base C&C parser, randomized baselines, our best initial reranking results, and our new reranking results using the features described in this chapter. We follow the same experimental setup as described in Chapter 5. The results were obtained using the CCG labeled dependency metric over Section 00 of CCGbank.

	t	LP	LR	LF	AF
Baseline	-	**87.19**	**86.32**	**86.75**	**84.80**
Random	-	83.40	81.97	82.68	81.10
Prev. Best	-	86.17	84.84	85.50	83.73
Classification+CCG	0	86.76	85.84	86.30	84.27
	2	**87.27**	**86.35**	**86.81**	**84.86**
	5	87.24	**86.35**	86.79	84.82
Classification+ALL	0	87.20	86.33	86.76	84.76
	2	**87.22**	**86.35**	**86.78**	**84.78**
	5	**87.22**	**86.35**	**86.78**	**84.78**
Regression+CCG	0	**87.39**	**86.48**	**86.93**	84.94
	2	87.36	86.44	86.90	84.91
	5	87.33	86.44	86.88	**84.95**
Regression+ALL	0	**87.89**	**86.94**	**87.41**	**85.46**
	2	87.51	86.57	87.03	85.05
	5	87.25	86.29	86.77	84.91

TABLE 7.1: 10-best reranking performance on Section 00 of CCGbank for various combinations of pruning values t, sets of features, and classification and regression experiments. CCG indicates our novel features, and ALL indicates the union of CCG with those described in Chapter 5. Bolded scores are the highest for the approach.

Our top result with 10-best reranking was an F-score of 87.41% over the baseline of 86.75% using gold-standard POS tags. The top 50-best reranking result was 87.22%. Both of these results were achieved using the union of Charniak's features with our novel features and regression modeling. These are both statistically significant improvements over the baseline and our previous best reranking results presented in Chapter 5.

In contrast to our initial experiments, regression proved to be much more effective than classification with our new features. Regression outperformed classification in every experimental setup over varying pruning values and sets of features. While binary classification better models the actual task of choosing a best parse, it seems that retaining as much information from the F-scores of the training parses ultimately leads to better performance.

	t	LP	LR	LF	AF
Baseline	-	**87.19**	**86.32**	**86.75**	**84.80**
Random	-	81.69	79.97	80.82	79.04
Prev. Best	-	85.76	84.27	85.01	83.36
Classification+CCG	0	**87.37**	**86.45**	**86.91**	**84.91**
	2	87.23	86.35	86.79	84.78
	5	87.27	86.37	86.82	84.85
Classification+ALL	0			-	
	2	**87.20**	**86.33**	**86.76**	**84.76**
	5	**87.20**	**86.33**	**86.76**	**84.76**
Regression+CCG	0	**87.51**	**86.61**	**87.05**	84.96
	2	87.46	86.54	87.00	**85.00**
	5	87.49	86.57	87.03	84.95
Regression+ALL	0			-	
	2	**87.70**	86.72	87.20	**85.20**
	5	**87.70**	**86.75**	**87.22**	85.13

TABLE 7.2: 50-best reranking performance on Section 00 of CCGbank for various combinations of pruning values t, sets of features, and classification and regression experiments. CCG indicates our novel features, and ALL indicates the union of CCG with those described in Chapter 5. Bolded scores are the highest for the approach. Blank entries indicate experiments that contained too many features to train.

7.2.1 Features

We investigated the performance of two sets of features: the novel features described in this chapter along with those used by Clark and Curran (2007b) in the C&C parser (CCG), and the union of this set with the features adapted from Charniak and Johnson (2005) discussed in Chapter 5 (ALL). The log score and rank of each parse were included as core features in every experiment. In general, more features improved performance. The best results were produced using all of the possible features in the reranker model. In terms of the top F-score for each set of features, our new features resulted in a statistically significant improvement over the baseline parser in several experiments, whereas the features of Charniak and Johnson (2005) produced a drop in performance as described in Section 5.3.

Notably, every single experiment using our new features outperformed the best result using only Charniak and Johnson (2005)'s features, indicating that features tailored to a dependency evaluation and to specific errors made by the C&C parser were more discriminative between good and bad CCG parses.

We conducted a subtractive feature analysis on our top 10-best reranking model to investigate the contribution of individual features to parser accuracy. Features were individually removed and the reranker was retrained and retested on Section 00 of CCGbank.

	LP	LR	LF	AF
Best	**87.89**	**86.94**	**87.41**	**85.46**
-CoPar	87.77	86.87	87.31	85.37
-CoLenPar	87.56	86.66	87.11	85.09
-Heavy	87.92	87.02	87.47	85.47
-RightBranch	87.56	86.69	87.12	85.07
-SubjVerbAgr	87.64	86.76	87.20	85.16
-Edges	87.56	86.65	87.10	85.10
-Rule	87.88	86.97	87.42	85.40
-Heads	87.58	86.70	87.14	85.08
-HeadTree	87.77	86.86	87.31	85.35
-Neighbours	87.60	86.72	87.15	85.06
-NGramTree	87.48	86.61	87.04	85.02
-SynSemHeads	87.62	86.69	87.15	85.12
-Word	87.88	86.98	87.43	85.45
-WProj	87.46	86.61	87.03	85.06
-CoHeads	87.66	86.77	87.22	85.15
-CoDups	87.88	86.96	87.42	85.40
-PrepResArg	87.59	86.69	87.14	85.14
-CompAdj	87.66	86.80	87.23	85.22
-LexDep	87.56	86.67	87.12	85.11
-NumDeps	87.55	86.66	87.10	85.08
-UnaryRule	87.49	86.58	87.03	85.07
-TypeRaising	87.52	86.67	87.09	85.06
-CatNGram	87.83	86.91	87.37	85.32
-C&CCommon	87.62	86.62	87.12	85.10
-C&CDep	87.45	86.61	87.03	85.03
-C&CNorm	87.46	86.55	87.00	85.07

TABLE 7.3: Subtractive analysis on the top performing 10-best model over Section 00 of CCGbank.

Table 7.3 summarises the subtractive analysis. There are no features which, when removed, cause the F-score to drop below 87%; most performance changes are little more than noise. This shows that it is the combination of features which is providing the performance improvement.

Feature pruning had a mixed impact on the accuracy of 10-best reranking accuracy, ranging from statistically insignificant to substantial. No pruning appeared to work best with regression, while larger pruning levels worked better with classification. Pruning had no significant impact on 50-best reranking accuracy, though all 50-best experiments using all possible features and no pruning proved to be infeasible given our computational resources (2.26Ghz Intel Xeons with 16 GB RAM). Training times for the experiments using larger pruning values were cut almost in half compared to smaller pruning values; however, all training times were in the order of a few hours and is only required once, making the benefit of pruning with respect to actual parsing time negligible.

7.2.2 Analysis of *PP* and Coordination Errors

We reran the error analysis performed in Chapter 6 on the output of our top 10-best reranking model over Section 00 of CCGbank. We found that the number of errors dropped in all classes except for duplicate coordination dependency errors, which did not change between the original parser and the reranked parser. This shows that our conjunction features were not as effective as we had hoped; it can be seen in the subtractive analysis in Table 7.3 that the removal of the CoDups features had a smaller impact on reranking accuracy compared to most other features. We hypothesis that the number of conjunction dependencies in the training data is too small to gather a representative set of coordination features that is useful in reranking.

Head category	Error type	Parser	Reranked
Coordination	Duplicate dependency	320	320
Predicate	Complement/Adjunct distinction	440	**421**
Prepositional Phrase	Attachment error	272	**233**
	Complement/Adjunct distinction	360	**308**
Total		**1392 / 5013**	**1282 / 4735**

TABLE 7.4: Summary of errors caused by *PP* and coordination errors as a proportion of total parser errors in Section 00 of CCGbank. The reranked numbers are generated using our top 10-best reranking model.

7.3 Final Results

Table 7.5 summarises the performance of our top 10-best and 50-best reranker models against the baseline normal-form model on Section 23 of CCGbank. This is the field-standard evaluation for CCG parsing. Both reranker models achieve a statistically significant improvement in F-score over the normal-form baseline for statistical CCG parsing. Following results on Section 00, the 10-best model is around 0.2% more accurate than the 50-best model. Our performance improvement in absolute terms was only of the order of 0.3 – 0.4% F-score, which is much smaller than the 1.55% improvement of Collins (2000)

	LP	LR	LF	AF
Baseline	87.75	86.98	87.36	85.07
10-best reranking	**88.22**	**87.35**	**87.78**	**85.48**
50-best reranking	**88.05**	**87.19**	**87.62**	**85.26**

TABLE 7.5: Baseline and final reranker performance over Section 23 of CCGbank with the normal-form model, using the best performing 10-best and 50-best reranker models as judged over Section 00.

and the 1.25% improvement of Charniak and Johnson (2005). We hypothesise that this is due to the difference in evaluation metrics, and the lower overall potential improvement from the oracle scores in CCG parsing.

7.4 Summary

We have developed a set of novel reranking features targeted at CCG dependencies, motivated by the analysis of parser errors in Chapter 6. We have comprehensively shown how these new features improve reranking performance on Sections 00 and 23 of CCGbank, in contrast to the performance drop seen using only the features of Charniak and Johnson (2005).

The actual performance improvement that we have seen, while statistically significant, is small in absolute terms compared to the performance improvements that have been previously reported using reranking. We conclude that reranking does not guarantee more accurate parsing: it is important to tailor a reranking system and features to the errors made by a particular parser.

CHAPTER 8

Conclusion

In this chapter we describe future directions for the work described in this thesis, and summarise the conclusions that we can draw from our work.

8.1 Future Work

The initial Parser Evaluation using Textual Entailment task from SemEval-2010 is limited by the size and scope of its dataset. There are only 66 development sentences and 301 test sentences from the task. We have shown how entailment evaluation has considerable promise for better and more representative parser evaluation. The next steps to establish this would be to create a new corpus of entailments of a similar size to Sections 00 and 23 of the Penn Treebank. This would involve collecting on the order of 4000 text and hypothesis pairs, and annotating them with the same inexpensive method used to produce the initial dataset. This would create a large, standardised corpus of a scale where evaluation can be meaningfully carried out.

Our current reranker implementation is an entirely separate program to the parser, leading to an order of magnitude decrease in speed. Our next step of development is to integrate the reranker directly into the parser to minimise the impact on parsing speed.

The reranker has only been tested on English parsing thus far. However, it is a natural extension to consider other languages, with the caveat that a careful examination of where reranking can help is necessary for performance improvements. A recent development of a CCG corpus for Chinese (Tse and Curran, 2010) provides the foundation for wide-coverage CCG parsing of Chinese that can utilise our reranking system.

The annotation bottleneck described in Section 2.7 means that it is difficult to acquire the large corpora of annotated text necessary for training accurate parsers. Furthermore, accuracy improvements must come from better utilisation of the corpora we already possess, and this can be achieved using our flexible reranking framework. Arbitrarily complex features may be incorporated to assess parse quality, and the source of these features need not be solely from the parse itself. We have described a single feature in this thesis that incorporates external counts from a parsed section of the North American News Corpus. There are a wide variety of additional sources from which we could source new features for the reranker, including the WordNet lexical database (Fellbaum, 1998), Google's Web1T n-gram counts corpus (Brants and Franz, 2006), and Wikipedia. A thorough exploration of these resources and the potential for new reranker features based on them has the potential to further improve parsing accuracy.

The core idea used in reranking is the notion of using the parser to generate the most probable alternatives according to its model, and then using a more flexible model to reorder them. This considers multiple *parses per sentence*. A natural extension of this idea is to consider *multiple sentences at once*. Parsers have historically considered all sentences to be independent one another for a variety of reasons, but this is a naïve assumption that ignores many of the structural and thematic similarities that abound in consecutive sentences. We propose the idea of *maximally compatible parsing*, where multiple sentences are parsed in parallel and the best analysis for each one is chosen given the wider context. This approach would allow parsers to more closely model the way in which people interpret and understand natural language, and allow more contextual information to be incorporated. Importantly, the way a *parse* is generated for each sentence would remain unchanged (this is the component of the parser most constrained by dynamic programming), but the *decoding and selection* of the best analysis would incorporate information from multiple sentences. In order for maximally compatible parsing to be successful, new decoding algorithms must be developed to do this.

8.2 Conclusions

Our core contributions have addressed numerous questions regarding parser evaluation and the benefits of reranking. We have developed a new evaluation system for the C&C parser based on an *extrinsic* textual entailment prediction task. The nature of this new evaluation means that it is fundamentally formalism-independent and can be used to fairly compare parsers based of different grammatical schemes. The basis on a natural human competence removes the reliance on linguistic annotation to produce evaluation data, and focusing the evaluation procedure on syntactic differences that are relevant to semantic interpretation, which is ultimately how most parsers are used. This methodology also avoids linguistic arguments about the merits of particular formalisms and their analysis of particular constructions. Our system improved on the upper bounds documented by previous attempts at cross-formalism comparison with the C&C parser, and placed second in an international evaluation. There is considerable scope for the task to develop into a new standard for fairer, more representative parser evaluation.

We have taken a state-of-the-art reranking system and reimplemented it for the C&C parser. Our detailed analysis of parsing errors prompted the development of novel features tailored for CCG reranking and targeting the errors that the parser actually makes. In the process, we have comprehensively described the features used by our reranker and provided a blueprint for future implementations.

An open question from the literature was whether reranking would improve accuracy for parsers that differed substantially in design from the Collins and Charniak systems. Johnson and Ural (2010) attempted to use the Charniak reranker with the Berkeley parser without modification, and reported negligible differences in performance. They speculated that the features of the reranker may have been implicitly adapted to the Charniak parser. Until now, there had been no attempt to implement a new reranker for a parser using a different formalism but the same underlying training data.

Our results conclusively show that features useful for reranking the Charniak parser are detrimental to CCG parsing accuracy, and that it is necessary to design reranker features specifically to complement a parser and the mistakes that it makes. Reranking does not guarantee performance improvements in and upon itself: it is possible to implement a reranker with state-of-the-art features and observe a performance deficit. Instead, respectable performance gains come through features tailored to the parser, and these gains will lead to improvements in many of the systems that utilise parsing for more sophisticated information management.

Bibliography

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

Yehoshua Bar-Hillel. 1953. A Quasi-Arithmetical Notation for Syntactic Description. *Language*, 29(1):47–58.

Ezra W. Black, Steven P. Abney, Daniel P. Flickenger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert J. P. Ingria, Frederick Jelinek, Judith L. Klavans, Mark Y. Liberman, Mitchell P. Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*, pages 306–311. Pacific Grove, California, USA.

Ezra W. Black, Roger Garside, and Geoffrey N. Leech, editors. 1993. *Statistically-driven computer grammars of English: The IBM/Lancaster approach*. Number 8 in Language and Computers. Amsterdam.

Thorsten Brants and Alex Franz. 2006. Web 1T 5-gram, version 1.

Forrest Brennan. 2008. *k-best Parsing Algorithms for a Natural Language Parser*. Master's thesis, University of Oxford.

Ted Briscoe. 2006. An Introduction to Tag Sequence Grammars and the RASP System Parser. Technical Report 662, University of Cambridge, Cambridge, United Kingdom.

Ted Briscoe and John Carroll. 2006. Evaluating the Accuracy of an Unlexicalized Statistical Parser on the PARC DepBank. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 41–48. Sydney, Australia.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The Second Release of the RASP System. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 77–80. Sydney, Australia.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-Coverage Deep Statistical Parsing using Automatic Dependency Structure Annotation. *Computational Linguistics*, 34(1):81–124.

John Carroll, Anette Frank, Dekang Lin, Detleft Prescher, and Hans Uszokoreit. 2002. Proceedings of the Workshop 'Beyond PARSEVAL - Towards improved evaluation measures for parsing systems'. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-02)*. Las Palmas, Canary Islands, Spain.

Eugene Charniak. 1997. Statistical Parsing with a Context-free Grammar and Word Statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 598–603. Providence, Rhode Island, USA.

Eugene Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-00)*, pages 132–139. Seattle, Washington, USA.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 173–180. Ann Arbor, Michigan, USA.

Nancy Chinchor. 1992. The statistical significance of the MUC-4 results. In *Proceedings of the Fourth Message Understanding Conference (MUC-4)*, pages 30–50. McLean, Virginia.

Kenneth W. Church and Ramesh S. Patil. 1982. Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table. *American Journal of Computational Linguistics*, 8(3–4):139–149.

Stephen Clark and James R. Curran. 2004a. Parsing the WSJ Using CCG and Log-Linear Models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 103–110. Barcelona, Spain.

Stephen Clark and James R. Curran. 2004b. The Importance of Supertagging for Wide-Coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 282–288. Geneva, Switzerland.

Stephen Clark and James R. Curran. 2007a. Formalism-Independent Parser Evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, pages 248–255. Prague, Czech Republic.

Stephen Clark and James R. Curran. 2007b. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552.

Stephen Clark and James R. Curran. 2009. Comparing the Accuracy of CCG and Penn Treebank Parsers. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers (ACL-IJCNLP '09)*, pages 53–56. Singapore.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building Deep Dependency Structures using a Wide-Coverage CCG Parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 327–334. Philadelphia, Pennsylvania, USA.

Michael Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 184–191. Santa Cruz, California, USA.

Michael Collins. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, pages 16–23. Madrid, Spain.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, Pennsylvania, USA.

Michael Collins. 2000. Discriminative Reranking for Natural Language Parsing. In *Proceedings of the*

17th International Conference on Machine Learning (ICML-00), pages 175–182. Palo Alto, California, USA.

Michael Collins and Terry Koo. 2005. Discriminative Reranking for Natural Language Parsing. *Computational Linguistics*, 31(1):25–70.

James R. Curran, Stephen Clark, and David Vadas. 2006. Multi-Tagging for Lexicalized-Grammar Parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pages 697–704. Sydney, Australia.

Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. 2009. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering*, 15(4):i–xvii.

Steve DeNeefe and Kevin Knight. 2009. Synchronous Tree Adjoining Machine Translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP-09)*, pages 727–736. Singapore.

Bojan Djordjevic. 2006. Efficient Combinatory Categorial Grammar Parsing. In *Proceedings of the 2006 Australasian Language Technology Workshop (ALTW-06)*, pages 3–10. Sydney, Australia.

Abdessamad Echihabi and Daniel Marcu. 2003. A Noisy-Channel Approach to Question Answering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, pages 16–23. Sapporo, Japan.

Jason Eisner. 1996. Efficient Normal-Form Parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 79–86. Santa Cruz, California, USA.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, Massachusetts, USA.

Daniel Gildea. 2001. Corpus Variation and Parser Performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP-01)*, pages 167–202. Pittsburgh, Pennsylvania, USA.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, University of Edinburgh, Edinburgh, United Kingdom.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Matthew Honnibal, Joel Nothman, and James R. Curran. 2009. Evaluating a Statistical CCG Parser on Wikipedia. In *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, pages 38–41. Singapore.

Liang Huang and David Chiang. 2005. Better k-best Parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT-05)*, pages 53–64. Vancouver, British Columbia, Canada.

Mark Johnson and Ahmet Engin Ural. 2010. Reranking the Berkeley and Brown Parsers. In *Proceedings of Human Language Technologies: the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-10)*, pages 665–668. Los Angeles, California, USA.

Tadao Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, pages 1–8. Budapest, Hungary.

Dan Klein and Christopher D. Manning. 2001. Parsing and Hypergraphs. In *Proceedings of the 7th International Workshop on Parsing Technologies (IWPT-01)*. Beijing, China.

Jonathan K. Kummerfeld, Jessika Roesner, Tim Dawborn, James Haggerty, James R. Curran, and Stephen Clark. 2010. Faster Parsing by Supertagger Adaptation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*, pages 345–355. Uppsala, Sweden.

Dekang Lin. 1995. A Dependency-based Method for Evaluating Broad-Coverage Parsers. In *Proceedings of the 14th International Joint Conference on AI (IJCAI-95)*, pages 1420–1425. Montreal, Canada.

Dekang Lin. 1998. A Dependency-based Method for Evaluating Broad-Coverage Parsers. *Natural Language Engineering*, 4(2):97–114.

David Magerman. 1995. Statistical Decision Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pages 276–283. Cambridge, Massachusetts, USA.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective Self-Training for Parsing. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT/NAACL-06)*, pages 152–159. New York City, New York, USA.

Guido Minnen, John Carroll, and Darren Pearce. 2000. Robust, applied morphological generation. In *Proceedings of the First International Conference on Natural Language Generation (INLG-00)*, pages 201–208. Mitzpe Ramon, Israel.

Mark-Jan Nederhof, Gosse Bouma, Rob Koeling, and Gertjan van Noord. 1997. Grammatical analysis in the OVIS spoken-dialogue system. In *Proceedings of the Interactive Spoken Dialog Systems Workshop*. Madrid, Spain.

Dominick Ng, James W.D. Constable, Matthew Honnibal, and James R. Curran. 2010. SCHWA: PETE Using CCG Dependencies with the C&C Parser. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval-2010)*, pages 313–316. Uppsala, Sweden.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pages 433–440. Sydney, Australia.

Adwait Ratnaparkhi. 1997. A Linear Observed Time Statistical Parser Based on Maximum Entropy

Models. In *Proceedings of the 1997 Conference on Empirical Methods in Natural Language Processing (EMNLP-97)*, pages 1–10. Providence, Rhode Island, USA.

Laura Rimell and Stephen Clark. 2010. Cambridge: Parser Evaluation Using Textual Entailment by Grammatical Relation Comparison. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval-2010)*, pages 268–271. Uppsala, Sweden.

Laura Rimell, Stephen Clark, and Mark Steedman. 2009. Unbounded Dependency Recovery for Parser Evaluation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP-09)*, pages 813–821. Singapore.

Federico Sangati, Willem Zuidema, and Rens Bod. 2009. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT-09)*, pages 238–241. Paris, France.

Satoshi Sekine. 1997. The Domain Dependence of Parsing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, pages 96–102. Washington, D.C., USA.

Libin Shen, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative Reranking for Machine Translation. In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-04)*, pages 177–184. Boston, Massachusetts, USA.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, Massachusetts, USA.

Ann Taylor, Mitchell P. Marcus, and Beatrice Santorini. 2003. The Penn Treebank: An Overview. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, pages 5–22. Kluwer Academic Publishers.

Daniel Tse and James R. Curran. 2010. Chinese CCGbank: extracting CCG derivations from the Penn Chinese Treebank. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING-2010)*, pages 1083–1091. Beijing, China.

David Vadas and James R. Curran. 2007. Adding Noun Phrase Structure to the Penn Treebank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, pages 240–247. Prague, Czech Republic.

David Vadas and James R. Curran. 2008. Parsing Noun Phrase Structure with CCG. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08:HLT)*, pages 335–343. Columbus, Ohio, USA.

K Vijay-Shanker and David J. Weir. 1994. The Equivalence of Four Extensions of Context-Free Grammars. *Mathematical Systems Theory*, 27(6):511–546.

Rui Wang and Yi Zhang. 2010. MARS: A Specialized RTE System for Parser Evaluation. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval-2010)*, pages 272–275. Uppsala, Sweden.

Michael White and Rajakrishnan Rajkumar. 2009. Perceptron Reranking for CCG Realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP-09)*, pages 410–419. Singapore.

Kent Wittenburg. 1986. *Natural Language Parsing with Combinatory Categorial Grammars in a Graph-Unification-Based Formalism*. Ph.D. thesis, University of Texas at Austin.

Kent Wittenburg. 1987. Predictive Combinators: A Method for Efficient Processing of Combinatory Categorial Grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL-87)*, pages 73–80. Stanford, California, USA.

Daniel H. Younger. 1967. Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10(2):189–208.

Deniz Yuret, Aydın Han, and Zehra Turgut. 2010. SemEval-2010 Task 12: Parser Evaluation using Textual Entailments. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval-2010)*, pages 51–56. Uppsala, Sweden.

APPENDIX A

The Penn Treebank Part-of-Speech Tag Set

Tag	Description	Tag	Description
``	Opening quotation mark	NNPS	Proper noun, plural
''	Closing quotation mark	PDT	Predeterminer
(	Opening parenthesis	POS	Possessive ending
)	Closing parenthesis	PRP	Personal pronoun
,	Comma	PRP$	Possessive pronoun
--	Dash	RB	Adverb
.	Sentence terminator	RBR	Adverb, comparative
:	Colon or ellipsis	RBS	Adverb, superlative
CC	Coordinating conjunction	RP	Particle
CD	Cardinal number	SYM	Symbol
DT	Determiner	TO	to
EX	Existential there	UH	Interjection
FW	Foreign word	VB	Verb, base form
IN	Preposition or subordinating conjunction	VBD	Verb, past tense
JJ	Adjective	VBG	Verb, gerund or present participle
JJR	Adjective, comparative	VBN	Verb, past participle
JJS	Adjective, superlative	VBP	Verb, non-3rd person singular present
LS	List item marker	VBZ	Verb, 3rd person singular present
MD	Modal	WDT	Wh-determiner
NN	Noun, singular or mass	WP	Wh-pronoun
NNS	Noun, plural	WP$	Possessive wh-pronoun
NNP	Proper noun, singular	WRB	Wh-adverb