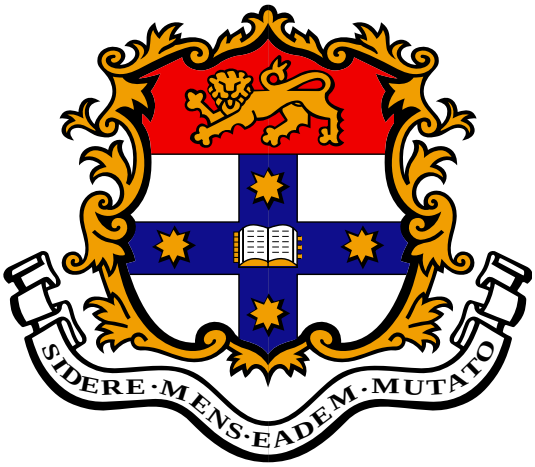
Training Parsers using Redundancy of Information

SUSAN HOWLETT
SID: 200409993



Supervisor: James R. Curran

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

School of Information Technologies
The University of Sydney
Australia

7 November 2008

Abstract

Supervised approaches to parsing natural language are state-of-the-art, but rely heavily on large amounts of annotated training data. The limitations of existing training data, in terms of both size and variety, has lead to a need for additional data. However, annotating the quantity and range of data required is prohibitively expensive.

We address this annotation bottleneck with an entirely automatic procedure for collecting and annotating text from the Web. We use the keywords of facts like *Mozart was born in 1756* to compile a corpus of related sentences. From this corpus we select simple sentences, which can be parsed with a high degree of accuracy, then use these syntactic analyses to place constraints on the analyses of all sentences collected. We extend a state-of-the-art parser to enforce these constraints and use the resulting parser output as training data.

Our research shows that it is indeed feasible to produce annotated training data through an entirely automatic procedure, and that we are now in a position to use raw Web text to create a new corpus that dwarfs the amount of data currently available. This will enable us to train robust and accurate parsers with which we can address a wide range of language processing tasks.

Acknowledgements

My deepest gratitude to my supervisor, James Curran, for introducing me to computational linguistics in my first year and for guiding me so well since.

Thank you also to the members of the Language Technology Research Group for providing advice and different perspectives at various times during the year, and to the University of Sydney for supporting me through an Honours scholarship.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

The amount of information stored in computer-readable form is vast and increasing, with at least a trillion words of English text on the Web alone (Brants and Franz, 2006). However, as more text is added, efficiently locating a particular piece of information becomes progressively more difficult. The process could be made considerably easier if the system storing and processing the information possessed some level of comprehension of the text that it indexes and the queries that it handles. Typically, this begins with the automatic analysis of the syntactic structure of the text, a process called *parsing*.

State-of-the-art approaches to parsing treat the process as a *supervised machine learning* task. That is, a large amount of annotated *training data*—in this case, text that has been marked up with its syntactic structure—is used to construct a statistical model that can then be used to predict the structure of previously unseen text. For parsing English, the main source of training data is the Wall Street Journal section of the Penn Treebank corpus, consisting of 1.1 million words of English newswire text that has been automatically annotated and then hand-corrected (Marcus *et al.*, 1993). This standard training corpus is conventionally referred to as the Penn Treebank, even though only a small section of the Penn Treebank proper is used.

The Penn Treebank has contributed substantially to the improvement of parsing and to the evaluation and comparison of parsing systems. However, there has been a growing realisation that to achieve further gains in performance across a variety of texts, more and larger resources are required. The development of the Penn Treebank took 8 years (Taylor *et al.*, 2003) and was extremely costly, chiefly due to the time-consuming and labour-intensive process of annotating the data. Although there is some work on the development of new resources (for example the LinGO Redwoods Treebank (Oepen *et al.*, 2002)), the high cost of manual annotation ultimately means that other methods of obtaining annotated data must be explored.

1.1 Contribution of this work

We address this data annotation problem by proposing a novel automatic procedure wherein a state-of-the-art parser is used to annotate its own additional training data. To do so, we rely on an observation that the information available on the Web is highly redundant, the intuition that simple sentences are easier to parse than complex sentences, and an assumption that certain information can be reliably transferred from one sentence to another.

We consider an observation in Brill *et al*. (2001) that the size of the Web increases the probability that a single piece of information can be found in several contexts, many of which require only simple linguistic processing to extract the information. We identify a number of different facts, for example *Mozart was born in 1756*, and retrieve sentences from the Web that contain all of the keywords of the fact, here *Mozart*, *born* and *1756*. We consider what makes a sentence simple and show that a Web search does indeed uncover numerous simple expressions of many facts, plus some more complex expressions.

Intuitively, it is easier to reliably determine the syntactic structure of a simple sentence, due to the limited ways in which the words can interact. A sentence such as (1) is short and is therefore likely to be reliably parsed, while a complex sentence, such as (2), is more difficult. However, these sentences are related by virtue of being collected using the same fact; we would like to transfer information from the simpler to the more complex sentence.

(1) Mozart was born in 1756.

(2) Wolfgang Amadeus Mozart (baptized Johannes Chrysostomus Wolfgangus Theophilus) was born in Salzburg in 1756, the second survivor out of six children.

Dependency grammars represent the syntactic structure of a sentence as relationships between pairs of words. The analysis of sentence (1) in one such grammar, described in more detail in Section 2.1.2, may be represented graphically by the diagram in Figure 1.1. The corresponding analysis for sentence (2) is shown in Figure 1.2. Comparing the analyses in Figures 1.1 and 1.2 we see that not only does the longer sentence contain the same words as the shorter sentence, because it is in part representing the same information, the structure of the shorter sentence is repeated within the structure of the larger sentence.

We make the novel assumption that if two sentences have some lexical overlap, by virtue of being collected using the same fact, then the analyses of the sentences will contain the same chain of relationships

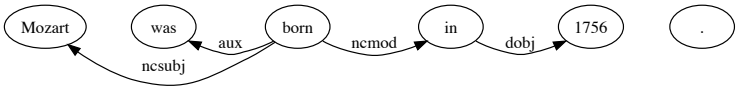


FIGURE 1.1: Analysis for sentence (1)

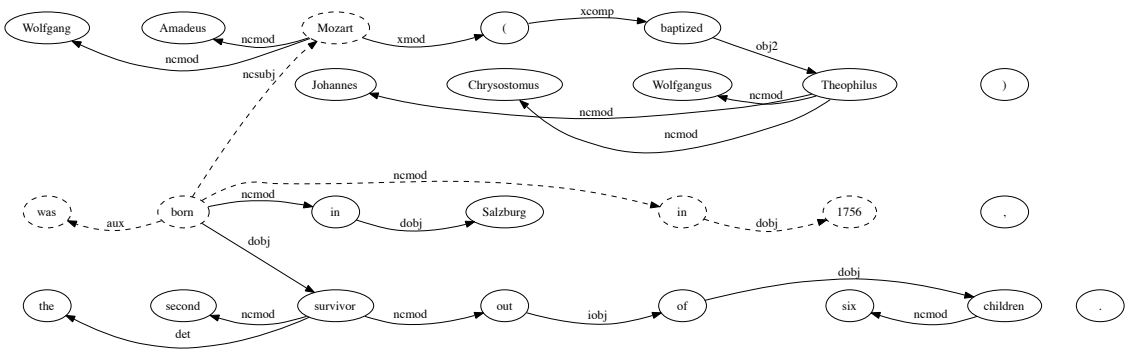


FIGURE 1.2: Analysis for sentence (2) with the analysis for sentence (1) indicated by dashed lines.

between the overlapping words. We present a new method for constraining a state-of-the-art parser to produce analyses consistent with a given chain of relationships.

Through the redundancy of information on the Web, we collect a number of simple sentences for some fact. Parsing each sentence furnishes us with a chain of relationships connecting the fact keywords. The fact that short sentences can be more reliably parsed implies that these chains are likely to be reliable, while the size of the sample of sentences collected is used to increase our confidence in a particular chain by evaluating the frequency with which the chain occurs.

Having established reliable chains of relationships for each fact and constrained the analyses of all associated sentences, we augment the parser's training data with the resulting constrained sentences, allowing the parser to incorporate information in its predictive model about the structures seen. By developing an automatic procedure for identifying constraint chains, we create a new, fully automatic procedure for increasing the amount of annotated training data available to the parser.

Finally, we evaluate the utility of this additional training data by comparing the performance of the parser, trained with and without the extra data, on the parser's standard evaluation corpus.

Our results show that it is feasible to automatically create annotated training data for statistical parsing, and this data can be created on an enormous scale for a range of domains. Using this data, we can train a robust and accurate parser for a range of natural language processing applications.

1.2 Outline

We begin in Chapter 2 with an overview of grammars and parsing, including a discussion of corpora and methods of evaluation. Chapter 3 presents our framework, describing in detail the particular grammar, corpus, parser and evaluation method we use. Chapter 4 explores the bottleneck that we face in parsing and introduces the way we plan to overcome it with a semi-supervised approach exploiting the redundancy of information on the Web. Chapters 5 to 7 describe the process by which we collect and annotate our new training data, which we evaluate in Chapter 8. Chapter 9 summarises our approach and describes the ways in which the work may be extended and applied to other areas.

Preliminary results of this work will appear at the *Australasian Language Technology Workshop 2008* under the title *Automatic Acquisition of Training Data for Statistical Parsers*.

CHAPTER 2

Grammars and Parsing

This chapter introduces the concept of the syntactic structure of a sentence and the ways this can be represented in a grammar. Section 2.2 discusses different ways in which a grammar may be characterised. In Section 2.3, we look at some of the important *corpora* (collections of text) used in parsing, then in Section 2.4 we consider one of the ways in which these corpora are used, to estimate probabilities about language. Section 2.5 describes the aim of parsing and one algorithm which may be used. Finally, Section 2.6 examines the ways in which a parsing system may be evaluated. In the following chapter, we will introduce the specific framework that we use, described in terms of the general information presented here.

2.1 Representing syntactic structure

Sentences are not simply unordered collections of words. For example, sentences (3) and (4) contain the same set of three words, yet describe two different events.

(3) Oswald shot Kennedy.

(4) Kennedy shot Oswald.

These two sentences differ in their meaning because the ways in which the words are related differ. *Bag of words* approaches to natural language processing treat sentences as unstructured and typically unordered collections of words, and as a result fail to recognise the difference between these two sentences, or the systematic way in which sentence (5) is related to sentence (3).

(5) Kennedy was shot by Oswald.

Syntax refers to the way in which words are combined in a sentence and the relationships that are produced as a result. A *grammar* is a description of the syntax of a particular language. The two main types of grammar are based on *constituent structure* and *dependency relations*.

2.1.1 Constituent structure

Constituents, also referred to as *phrases*, are groups of adjacent words that operate as a single unit. The *constituent structure* or *phrase structure* of a sentence is the way in which the words and constituents combine to form larger constituents.

This approach to representing syntactic structure begins with the notion that words fall into different classes, called *parts of speech* or *word classes*, with different distributional properties. Examples of these classes are *noun*, *verb*, *adjective*, and *determiner*.

There are likewise different kinds of phrase-level constituent, such as *noun phrase* and *verb phrase*. The labels noun phrase, verb phrase, etc. may be referred to as *phrasal categories*. For example, a noun phrase might consist of a determiner, followed by zero or more adjectives, followed by a noun, while a verb phrase might consist of a verb followed by a noun phrase.

In each constituent, we can identify one element as the *head* of the phrase, which is in some sense the core word of the phrase. Roughly, it determines the syntactic properties of the phrase and is the element that is modified by all other phrasal elements. In the case of a noun phrase, the head is the noun; in a verb phrase, it is the verb.

The construction of constituents is usually represented through *phrase structure rules*, for example

$$NP \rightarrow \left\{ \begin{array}{c} Det\ Adj^*\ N \\ PN \end{array} \right\}$$

$$VP \rightarrow V\ NP$$

where NP = noun phrase, VP = verb phrase, Det = determiner, Adj = adjective, N = noun, PN = proper noun. The asterisk represents 0 or more repetitions of the preceding element, and the braces indicate a choice between several options. Parentheses would be used to indicate optional elements.

These phrase structure rules are typically *context free*; that is, the left-hand side consists of only a single non-terminal symbol. Such grammars can be converted into *Chomsky normal form* (*CNF*) to simplify

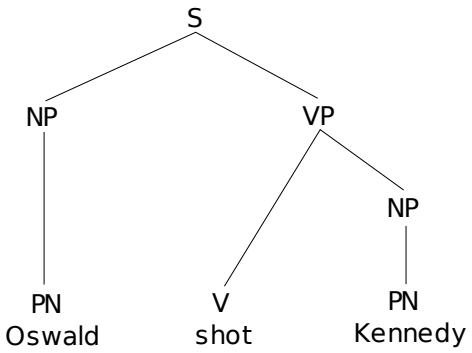


FIGURE 2.1: Phrase structure tree for sentence (3), where S = sentence, NP = noun phrase, VP = verb phrase, PN = proper noun, V = verb.

processing. In CNF, the right-hand side of each rule consists of either two non-terminals or a single terminal symbol.

Phrase structure rules describe how constituents may be formed in the language in general. The structure of a specific sentence is then given as a tree structure such as that depicted in Figure 2.1, where each node represents a constituent or word, and the children of a node X must be licensed by the right-hand side of a phrase structure rule with X on the left-hand side.

For computational purposes, the constituent structure of a sentence may be represented in a string format by the use of labelled bracketing. For example, the tree in Figure 2.1 may be represented by the string `(S(NP(PN Oswald))(VP(V shot)(NP(PN Kennedy)))).`

2.1.2 Dependency relations and the RASP GR scheme

An alternative description of the syntactic structure of a sentence is a list of relationships between pairs of words. Such relationships include *subject*, *object* and *modifier* relationships, and are variously known as *predicate-argument structures*, *dependency relations*, *dependencies* and *grammatical relations*.

The pair of words related by a dependency relation do not share an equal status; one word, called the *head*, creates the dependency, while the other, the *dependent*, simply fills in the slot in the dependency. The overlap in terminology with constituent structure comes about from the notion that all elements in a constituent are related to the head of the constituent by dependency relations. In sentence (3), the word *shot* is the head of two dependency relations, `subj` and `obj`, with *Oswald* and *Kennedy* respectively acting as the dependents. This may be represented with a diagram such as Figure 2.2.

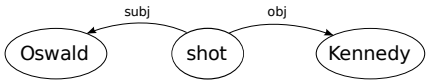


FIGURE 2.2: Grammatical dependencies for sentence (3).

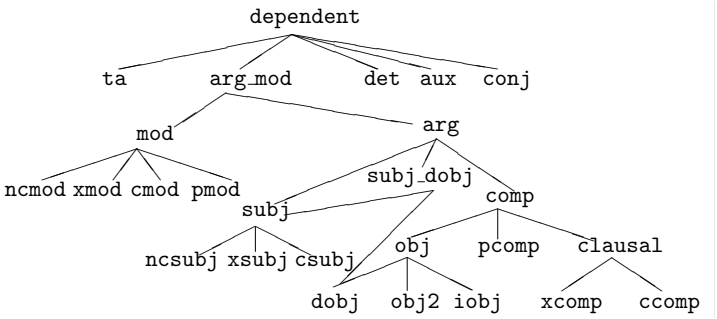


FIGURE 2.3: The grammatical relations hierarchy used in the RASP parser, taken from Briscoe (2006)

Like with constituent structure, it is convenient to be able to represent dependency relations in string form. Here, we represent them as 3-tuples of the form `(type head dependent)`, for example `(subj shot Oswald)` and `(obj shot Kennedy)`.

The RASP system parser (Briscoe *et al*., 2006) makes use of a particular scheme of dependency relations, described in detail in Briscoe (2006). In the RASP system, the relations are referred to as grammatical relations or GRs. Although this term is a generic label equivalent to *dependency relations*, to avoid confusion we shall henceforth use the terms grammatical relations and GRs to refer exclusively to this particular dependency scheme, and not as generic terms.

The RASP GR scheme arranges grammatical relations in a hierarchy, as represented by a tree-like structure, reproduced in Figure 2.3. The hierarchy enables a system to be more or less specific about a particular relation, for example the relation `subj` encompasses three more specific subject relations: `ncsubj`, for a non-clausal subject such as in *He matters*; `xsubj`, a subject consisting of a predicate that has no subject of its own such as in *Leaving matters*; and `csubj`, a subject consisting of a predicate that has a subject expressed such as in *That he left matters*.

Note that the RASP GR hierarchy is not strictly a tree because of the `subj_dobj` node, which allows a system using this scheme to avoid disambiguating between the subject and direct object of a verb. Briscoe (2006) mentions that this is included for the purposes of cross parser evaluation.

An example of a sentence analysed according to the RASP GR scheme will be given in Section 2.3.2.

2.2 Characterising grammars

It can be useful to categorise grammars according to some of their properties. Two properties that we might consider are the *depth* of the grammar and the method by which the grammar was developed.

2.2.1 Depth

Cahill *et al*. (2008) use the notion of *depth* to describe whether a grammar associates sentences with any semantic information in addition to its syntactic information. For this purpose, dependency relations are considered semantic as well as syntactic because they more explicitly encode the idea of *who did what to whom*. The RASP system parser is an example of a *deep* grammar.

Under this analysis, grammars based purely on constituent structure, for example Collins (1999) and Charniak (2000), are *shallow*. Constituent structures may, however, be the basis of deep grammars as well. For example, Lexical Functional Grammar (LFG), used in the XLE parser discussed in Cahill *et al*. (2008), augments constituent structure trees with *functional structures* which capture dependency relations. For example, a simplified functional structure that might be associated with the top node of the tree in Figure 2.1 is

$$\begin{bmatrix} \text{PRED} & \text{‘shot’} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{‘Oswald’} \end{bmatrix} \\ \text{OBJ} & \begin{bmatrix} \text{PRED} & \text{‘Kennedy’} \end{bmatrix} \end{bmatrix}$$

2.2.2 Method of development

We may also distinguish between *hand-crafted* grammars, which have been developed manually, and *automatically-acquired* grammars, where the components of the grammar are automatically extracted from raw or annotated text such as the corpora described in Section 2.3 below.

In hand-crafted grammars, the rules describing the composition of constituents or the identification of dependency relations are written by hand. Examples of such grammars are used in Riezler *et al*. (2002) and the RASP system (Briscoe *et al*., 2006). While these grammars are typically deep, few

can accommodate a wide variety of sentences and attempts at increasing the grammar's coverage incur significant development costs (Cahill *et al.*, 2008).

In contrast, automatically-acquired grammars are developed by extracting possible word classes, phrase structure rules or dependency identification rules from a corpus of text. This corpus may contain raw text, or it may contain sentences that have been annotated with syntactic information. Automatically-acquired grammars are frequently wide-coverage and cheaper to produce than hand-crafted grammars. However, they are dependent on the corpus used, in terms of the range of structures extracted and the depth of the analysis. Automatically-acquired grammars are used in, for example, Collins (1999), Charniak (2000), and Bod (2006).

2.3 Corpora

A *corpus* may be defined as any collection of texts, typically for the purposes of linguistic study. It may consist of raw text, or may contain additional information in the form of annotations, from part of speech tags to constituent structures or dependency information.

In parsing, corpora are used for a number of different purposes. Firstly, they may be used for grammar development, as described in Section 2.2.2 above. Secondly, they can provide statistical information, to be discussed in Section 2.4. Annotated corpora may also be used for evaluation, as described in Section 2.6.

In this section, we describe the main corpus used in English parsing, the Penn Treebank, plus one of its derivative corpora, the PARC 700 Dependency Bank. The development of both corpora involved some manual annotation efforts, which is a costly process despite being supported by automatic procedures.

2.3.1 The Penn Treebank

The Penn Treebank project was an 8-year endeavour (1989–1996) to create a large, annotated corpus to aid research into both computational and theoretical linguistics (Taylor *et al.*, 2003). The corpus was developed in two stages; the first stage completed in 1992 with the first release of the corpus.

The first release of the Penn Treebank contained approximately 2.9 million words of text annotated with constituent structure trees, including the 1.2 million tokens of the Brown corpus and 1.1 million words of newswire text from the Wall Street Journal. An example of such an annotated sentence is reproduced

```
( (S
   (NP Battle-tested industrial managers
       here)
    always
   (VP buck
       up
       (NP nervous newcomers)
       (PP with
           (NP the tale
               (PP of
                   (NP (NP the
                           (ADJP first
                                 (PP of
                                     (NP their countrymen)))
                           (S (NP *)
                              to
                              (VP visit
                                  (NP Mexico))))
                       ,
                       (NP (NP a boatload
                               (PP of
                                   (NP (NP warriors)
                                       (VP-1 blown
                                           ashore
                                           (ADVP (NP 375 years)
                                                 ago)))))
                        (VP-1 *pseudo-attach*)))))))))
  .)
```

FIGURE 2.4: An example illustrating Penn Treebank annotation, taken from Marcus *et al*. (1993)

in Figure 2.4. A further 2 million words of text from the Wall Street Journal were annotated with part of speech (POS) tags but not constituent structure. Both POS tags and constituent structures were added to sentences by an automatic annotation process followed by manual correction. These processes are described in detail in Marcus *et al*. (1993). In the second phase, this annotation was extended to contain some grammatical relation information (Marcus *et al*., 1994).

This corpus, or more specifically the 1.1 million words of text from the Wall Street Journal that have been annotated with constituent structure trees, has become become highly influential in English parsing. Although the Wall Street Journal text is only a subset of the corpus, in parsing the term *Penn Treebank* now conventionally refers to this smaller collection. When developing parsers on this text, sections 02–21 are typically used for training data, section 00 for development and section 23 for testing.

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 2.5: The DepBank annotations for sentence (6) with both the original (top) and Briscoe and Carroll (2006) (bottom) annotations. This figure is taken from Briscoe and Carroll (2006).

The emergence of the Penn Treebank as a field standard during its first phase is a significant factor in the large number of shallow, automatically-acquired grammars. Cahill *et al*. (2008) note that automatic acquisition of deep grammars generally involves either the conversion of the Penn Treebank phrase structure trees into a new formalism or the annotation of these trees with extra dependency information, producing a variety of related resources.

2.3.2 The PARC 700 Dependency Bank (DepBank)

One corpus derived from the Penn Treebank is the PARC 700 Dependency Bank, or DepBank. DepBank consists of 700 sentences randomly selected from the Penn Treebank Wall Street Journal section 23. In creating the corpus, the sentences were parsed with an LFG parser and the functional structures produced (see Section 2.2.1) were converted into another format and then manually corrected (King *et al*., 2003). An example of a sentence annotated in the DepBank format, sentence (6), is given in the top half of Figure 2.5.

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

Although the DepBank annotations are similar to the RASP grammatical relation scheme, Briscoe and Carroll (2006) adjusted the annotations to more reflect their scheme more closely, in order to evaluate the RASP parser. This new version of DepBank has been made publicly available (Briscoe and Carroll, 2006). The bottom half of Figure 2.5 gives sentence (6) after reannotation, taken from Briscoe and Carroll (2006).

2.4 Language models and training parsers

Due to ambiguities in natural language, sentences typically have several possible syntactic analyses. In order to choose between them, a parser must be able to determine the probability of each candidate analysis. These probabilities form the parser's *language model*. Since these probabilities cannot be determined exactly, they must be estimated from a sample of the language, a corpus. This process is called *estimating the language model* or *training the parser*, and is accomplished with one of a number of *learning algorithms*.

Parsing systems may be characterised by whether the learning algorithm used is *supervised* or *unsupervised*. Although hand-crafted grammars may theoretically be combined with unsupervised algorithms, in practice unsupervised algorithms only accompany automatically-acquired grammars.

A supervised learning algorithm requires training data that consists of a large number of sentences which have been annotated with the structures that the language model is expected to evaluate. That is, supervised approaches require large, annotated corpora such as those described in Section 2.3. Examples of supervised parsers include Collins (1999) and Charniak (2000).

In contrast, an unsupervised learning algorithm estimates probabilities directly from raw text. To do this, the system must discover patterns in the text; the patterns it discovers therefore do not necessarily conform to any pre-determined set of structures. Unsupervised approaches to parsing are quite rare, often confined to a separate area of research known as grammar induction. However, some work on unsupervised parsing exists.

Both Bod (2006) and Seginer (2007) present unsupervised parsing systems that appear to produce results comparable to those achieved with supervised techniques. Indeed, Bod (2006) claims that his results herald an end to purely supervised approaches. It is not clear, however, that the results presented

are as impressive as they seem. Sentences in newswire text are typically long; the Penn Treebank has an average sentence length of approximately 20 words and the field standard is to evaluate parser performance on sentences up to 40 words. However, in both Bod (2006) and Seginer (2007), the main part of the evaluation takes place on sentences up to 10 words long. Further, Bod (2006) explicitly says that the supervised system which is used as a comparison is not state-of-the-art, and has been binarised, causing a further decrease in its performance. This suggests that unsupervised parsing is still far behind supervised parsing in terms of performance.

2.5 Parsing

In brief, the task of parsing is to identify the possible structures of a sentence, as allowed by a particular grammar, and then to assign probabilities to these structures based on a language model and return the most likely analysis. To illustrate how this can be done, this section describes one important parsing algorithm, known as the Cocke-Kasami-Younger (CKY) algorithm, for parsing a context-free constituent structure grammar in Chomsky normal form (see Section 2.1.1).

The CKY algorithm is called a *chart parsing algorithm* because it uses a triangular chart to store possible nodes in the parse tree. Each cell in the chart represents a different *span* or sequence of words in the sentence, and contains all the roots of all possible parse trees that could generate that span. An example chart is given in Figure 2.6. The numbers beside each word and along the left side of the chart are used to identify cells in the chart and thus particular spans. For example, the cell containing PP is cell $(3, 2)$ and represents the span with length 2, beginning at position 3 in the sentence; that is, the span *in 1756*. The fact that this cell contains PP represents the fact that *in 1756* is a prepositional phrase.

The CKY algorithm is a dynamic programming algorithm that fills this chart from the bottom up according to a given grammar. In the example in Figure 2.6 we use sentence (7) and the following grammar, which does not correspond to any particular linguistic formalism and is intended purely for illustration.

(7) Mozart was born in 1756.

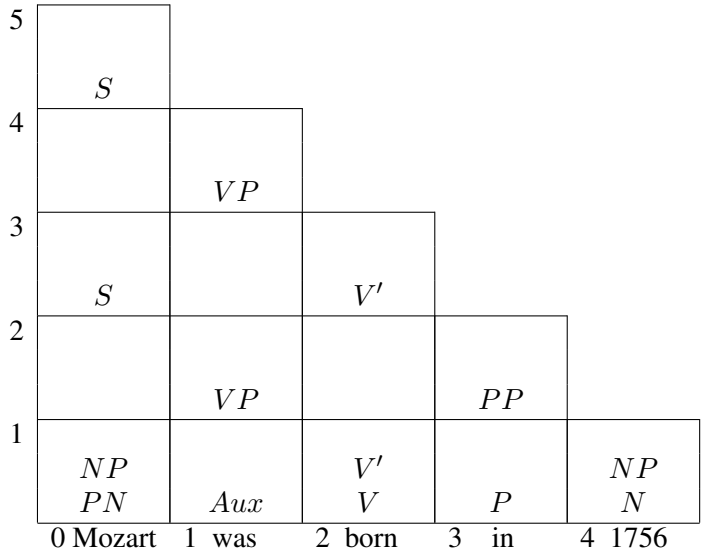


FIGURE 2.6: Chart representing the analyses of a simple sentence

$$S \rightarrow NP\ VP \qquad PP \rightarrow P\ NP$$

$$VP \rightarrow Aux\ V' \qquad NP \rightarrow \left\{ \begin{array}{c} PN \\ (Det)\ N \end{array} \right\}$$

$$V' \rightarrow V\ (PP)$$

First, note that this grammar is not in Chomsky normal form, since it contains *unary* (non-branching) rules: $V' \rightarrow V$, $NP \rightarrow PN$ and $NP \rightarrow N$ (recall that parentheses indicate optional elements). This entails an adaptation to the CKY algorithm, namely the inclusion of a step when unary rules may apply. This is a small adjustment to the algorithm only.

The algorithm proceeds as follows. First, the bottom row of cells, representing the individual words, are filled with the words' possible parts of speech. In Figure 2.6, these are the entries PN, Aux, V, P and N in the bottom row. Next, for grammars that contain unary rules, for example $NP \rightarrow PN$, any cells containing a node of the same type as the right-hand side of a rule (PN) will have an additional node added that represents the left-hand side of the rule (NP). This new node will have the first node as its only child. The action of unary rules add the NP and V' options to the bottom row of the chart in Figure 2.6.

Each row of the chart is then filled in in turn. The cell (i, j) is filled by considering some pair of cells (i, k) and $(i + k, j - k)$ in the chart. That is, we divide the span beginning at position i of length j into two parts, the first of length k and the second of length $j - k$. If the cell (i, k) contains a node Y and cell

$(i + k, j - k)$ contains a node Z and there exists a rule in the grammar $X \rightarrow Y\ Z$, then a node X, with the nodes Y and Z as children, will be added to cell (i, j). This repeats for all possible choices of k. In this example, the node $(0, 1)$ contains an NP node for the span *Mozart*, and the node $(1, 2)$ contains a VP node for the span *was born*. These can combine according to the rule $S \rightarrow NP\ VP$, so a node S is added the cell $(0, 3)$, representing the fact that *Mozart was born* is a valid sentence under this grammar.

At the completion of each row, if the grammar contains any unary rules, then these are applied as before, and the process repeats with the next row in the chart. In this example, the unary rules only apply on the first row of the chart.

When all of the cells in the chart are filled in, the possible analyses of the sentence are represented by the nodes in the top cell of the chart, here $(0, 5)$, which represents the span corresponding to the complete sentence. In this example, because this cell contains an S node, the grammar contains an analysis for the complete sentence. What is not evident from Figure 2.6 is that each node in the chart contains other nodes as its children; for example the S in cell $(0, 5)$ has as children the NP node in cell $(0, 1)$ and the VP node in cell $(1, 4)$. These child links allow us to retrieve the full parse tree from the chart once it has been filled.

2.6 Parser evaluation

Once a parsing system has been developed, it is necessary to evaluate the quality of its output. This may be done in one of two ways: by measuring the parser's performance in some application-independent evaluation, or by measuring the impact of the parser on the performance of a larger language-processing system engaged in another task. Borrowing terminology from biology, Ide and Véronis (1998) describe these two styles of evaluation as *in vitro* and *in vivo* evaluation respectively.

Ide and Véronis (1998) describe how the development of methods for in vitro evaluation requires a close examination of the features of the problem and thus leads to a deeper understanding of the task, while in vivo evaluation does not require agreement on possibly elusive concepts in order to provide a quantitative comparison of systems, but does not further understanding of the task nor help to demonstrate the transferability of methods to other applications. Ide and Véronis (1998) discuss these styles of evaluation in the context of another language processing task, *word sense disambiguation*, where the aim is to identify which meaning of a polysemous word is intended in a particular context. Although semantics

may be more difficult to make precise than syntax, their comments regarding the two styles of evaluation may be applied here also.

Bod (2006) argues that parsers should be evaluated in terms of their contribution to a concrete task (in vivo), since evaluation with respect to any annotated corpus will favour parsers trained on that form of annotation. It is not clear whether the author intends to include annotation schemes based on dependency relations in this statement. However, the majority of evaluation methods for parsing are in vitro.

2.6.1 Field standard evaluation

The current standard method of parser evaluation is the calculation of the PARSEVAL metrics of *precision*, *recall* and *F-score*. These metrics evaluate the similarity between the phrase structure trees in a given evaluation corpus, called the *gold standard*, and the phrase structure trees produced by the parser when analysing the text of the corpus. This similarity is calculated through a comparison of the bracketed string representation of the trees, determining the extent to which the bracketing in the two analyses match.

Originally used to evaluate the performance of classification systems, in this context precision (P) and recall (R) are defined as

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

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

F-score is defined as the harmonic mean of precision and recall, and is given by

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

Two additional metrics, labelled precision and labelled recall, may also be used. These differ from the unlabelled metrics in that they require the label for the phrasal grouping, as well as the locations of the delimiting brackets, to match the gold standard.

The field standard evaluation corpus is the Penn Treebank Wall Street Journal section 23. In this setup, state-of-the-art parsing achieves results of approximately 85–90% labelled precision and recall (Collins, 2003).

2.6.2 Alternative evaluations

There are a number of disadvantages to this standard evaluation scheme. Cahill *et al*. (2008) argue that tree representations of sentences do not necessarily provide enough information for language processing tasks because of their shallow nature. Secondly, they argue that there are a number of different but equally valid tree representations for a single sentence, and the gold standard used for evaluation must adopt one particular representation, placing parsers trained on other representations at a disadvantage.

In the introduction to the Language Resources and Evaluation Conference (LREC) workshop *Beyond PARSEVAL*, Carroll *et al*. (2002) describe how many researchers have found that the standard combination of Penn Treebank and PARSEVAL metrics is inappropriate for their work and therefore make use of a variety of other evaluation schemes. Carroll *et al*. (2002) point out that there is little agreement in the field regarding these alternative methods of evaluation, thus motivating the workshop, in which a number of different evaluation schemes are discussed. Among these are a scheme based on edit distance (Roark, 2002) and the leaf-ancestor metric (Sampson and Babarczy, 2002), although neither has gained much currency. Both apply to tree-based representations of syntactic structure.

Roark (2002) advocate comparing string representations of the parser output and gold standard parse using *edit distance*. Edit distance is defined as the number of operations on a string required to make it identical to a second string, with flexibility provided by the flexibility in defining the possible operations. Typically operations include the deletion or insertion of a unit such as a character, or the replacement of one unit with another. Roark (2002) argues that this is the method of evaluation favoured in speech recognition research, which has a closer relationship to parsing than does classification.

The leaf-ancestor metric proposed by Sampson and Babarczy (2002) focuses on the paths in the parse tree from the root to the leaf nodes. In phrase structure trees, leaf nodes generally have a one-to-one correspondence with words in the sentence. As such, for each word a path can be calculated from the root of the tree for both the gold standard tree and the parser output. Sampson and Babarczy (2002) propose evaluating the parse by calculating the similarity of each pair of such paths. Interestingly, they also make use of the concept of edit distance for the similarity measure.

2.6.3 Dependency-based evaluation

One final alternative to the Penn Treebank plus PARSEVAL evaluation makes use of grammatical relations, a strategy which appears to have its origins in work by Lin (1998). Instead of calculating precision

and recall over tree bracketings, Briscoe *et al*. (2002) propose the calculation of precision and recall over lists of dependencies between words. In this case, precision and recall would be defined as

$$P = \frac{\text{\# dependencies correct}}{\text{\# dependencies in the parser output}}$$

$$R = \frac{\text{\# dependencies correct}}{\text{\# dependencies in the gold standard}}$$

F-score is still defined as the harmonic mean of these two quantities.

Briscoe *et al*. (2002) advocate using the RASP GR scheme for this evaluation, as the hierarchy supports evaluation at various levels of granularity. Note that the hierarchy presented in Briscoe *et al*. (2002) is slightly different from the hierarchy presented in the later work Briscoe (2006), which we discussed in 2.1.2.

Dependency-based methods such as this have been proposed as theoretically allowing more meaningful cross-formalism comparison (Cahill *et al*., 2008) as they are not subject to the same disadvantages as tree-based methods. Their application has shown that they are not entirely free from these drawbacks (Cahill *et al*., 2008; Clark and Curran, 2007); nevertheless, the scheme has gained in popularity and appears to be emerging as a new field standard alongside the PARSEVAL metrics, and are what we use here.

2.7 Summary

In this chapter, we have given an overview of different kinds of grammars and how they are used to provide syntactic analyses of sentences. We have also discussed some important corpora, and the ways in which parsers can be evaluated. In the next chapter, we will describe the experimental framework that underlies the remainder of this thesis, and how it relates to the general framework presented here.

CHAPTER 3

Parsing with Combinatory Categorial Grammar

This chapter describes the framework within which this research operates, placing it within the context of the overview given in Chapter 2. Section 3.1 describes the grammar formalism used, called *Combinatory Categorial Grammar* (CCG). Section 3.2 describes the corpus used to train a CCG parser, which is then described in detail in Section 3.3. Section 3.4 discusses the methods by which this parser is evaluated. Finally, in Section 3.5 we outline some previous extensions to the parser.

3.1 Representing syntactic structure

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a deep, *mildly context-sensitive*, *lexicalized* grammar formalism that incorporates both constituent structure and dependency information in its analyses. This section describes CCG in relation to the general description of grammars given in Section 2.1. First, we describe the labels that can be assigned to constituents, corresponding to the word- and phrase-level categories in Section 2.1.1. Next, we describe the rules by which constituents in CCG can be combined, corresponding to the phrase structure rules also discussed in that section. We then describe how constituent labels are marked up to enable us to generate dependency information, and how these dependencies are formed.

3.1.1 Categories

The labels that represent the possible constituents in CCG (roughly corresponding to noun phrase, verb phrase, etc.) are called *categories*. These categories may be either *atomic*, representing constituents that can stand alone, or *complex*, which are functions that require the presence of particular arguments in order to produce a grammatical construction.

The possible atomic categories are S, N, NP and PP, representing clauses, nouns, noun phrases and prepositional phrases respectively. These may carry additional features, for example a noun phrase beginning with a determiner is marked $NP[nb]$, indicating a *non-bare* noun phrase, while *declarative* clauses are marked $S[dcl]$ to distinguish them from, for example, *interrogative* clauses ($S[q]$) or *infinitival* clauses marked with *to* ($S[to]$).

Complex categories are represented by binary structures of the form R/A or $R\backslash A$, where A is an argument (a constituent with either an atomic or a complex category) and R is the result category produced when the category is combined with its argument A. The forward and backward slashes specify that A is expected to the right and left of the category, respectively. Multiple arguments are specified through nesting; for example, $(S[dcl]\backslash NP)/PP$ represents a word expecting a prepositional phrase to the right then a noun phrase to the left and which, when both arguments have been found, forms a declarative clause. This would be the category of a verb such as *alluded* in sentence (8).

(8) Mobil alluded to the work-force cuts.

Many grammars make a distinction between labels applied to word-level categories (parts of speech) and those applied to phrase-level categories. CCG, however, makes no such distinction. For example, the category $S\backslash NP$ may apply to either an *intransitive verb* (a verb that takes no arguments) or to a complete verb phrase, containing a verb plus all of its arguments. This CCG representation captures the fact that in the context of the rest of the sentence, an intransitive verb acts in much the same way as a complete verb phrase.

3.1.2 Combinatory rules

CCG is termed a *lexicalized* grammar formalism because it aims to include as much grammatical information as possible in the lexical items, and combine constituents using only a few, generic rules. The basic combinatory rules in CCG, which come from the original context-free Categorial Grammar (Bar-Hillel, 1953), are two *functional application* rules that specify that a constituent with a complex category can be combined with the outermost of its nested arguments to produce a constituent with the category of the complex category's result. These two rules are called *forward* ($>$) and *backward* ($<$) *application*

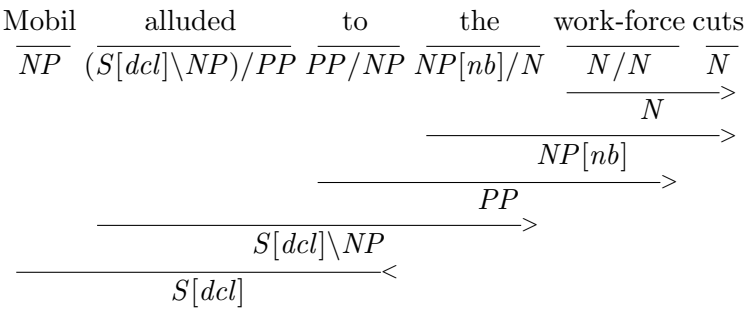


FIGURE 3.1: CCG derivation for sentence (8).

and are respectively represented

$$X/Y \;\; Y \Rightarrow X \quad (>)$$

$$Y \;\; X\backslash Y \Rightarrow X \quad (<)$$

An analysis or *derivation* illustrating these rules, using sentence (8), is given in Figure 3.1. Although they are presented differently, CCG analyses are in fact quite similar to phrase structure trees like that shown in Figure 2.1. Each category in the CCG analysis represents a constituent and thus corresponds to a node in the tree. Unlike phrase structure trees, CCG analyses are conventionally written with the words at the top.

In addition to these functional application rules, CCG contains a number of other combinatory rules that increase the grammar's expressive power from context-free to mildly context-sensitive (Vijay-Shanker and Weir, 1994). These rules are:

Forward composition

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

Backward composition

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

Backward crossed composition

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

Type-raising

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

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

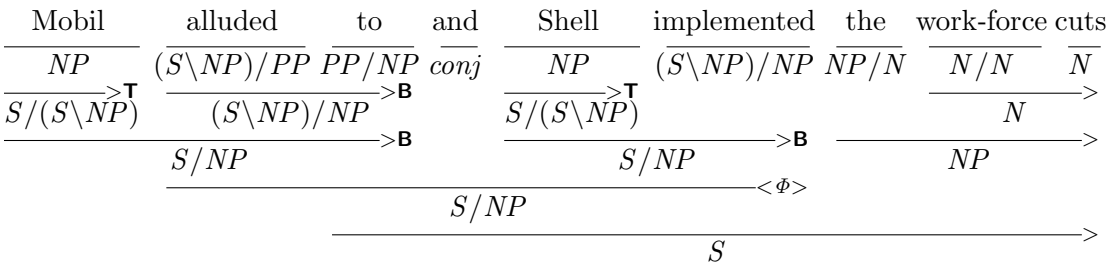


FIGURE 3.2: CCG derivation illustrating type-raising, forward composition, and coordination.

The T in these last two rules may be any CCG category, although there may be language-dependent restrictions. Finally, CCG contains a single *ternary* rule for *coordination*, expressed

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

An analysis incorporating some of these rules is given in Figure 3.2, using sentence (9).

(9) Mobil alluded to and Shell implemented the work-force cuts.[1]

The extra expressiveness afforded by these additional rules leads to a number of distinctive features of CCG. First, CCG provides a natural analysis for a phenomenon known as *non-constituent coordination*, where two sections of a sentence that are not typically recognised as constituents are coordinated. An example of this is illustrated in Figure 3.3. Here, *a teacher an apple* is not generally considered to form a single constituent. Coordination is usually restricted to constituents, so other grammars are only able to explain this phenomenon by positing the deletion of a second instance of the word *give* in the sentence.

Secondly, these additional rules lead to what is known as *spurious ambiguity*, where even a simple sentence such as that in Figure 3.1 has alternative analyses, one of which is given in Figure 3.4. Although constituents have been combined in a different order, the analyses are equally valid because the dependencies that are created (discussed in the next section) remain the same.

To reduce the level of ambiguity in the grammar and avoid such additional analyses, we can define the concept of a *normal-form derivation* where restrictions known as *Eisner constraints* aim to ensure that the type-raising and composition rules are used only when necessary. Specifically, these constraints

[1] Although sentence (8) is adapted from a Penn Treebank sentence, the reference to Shell has been invented for the purposes of illustration only.

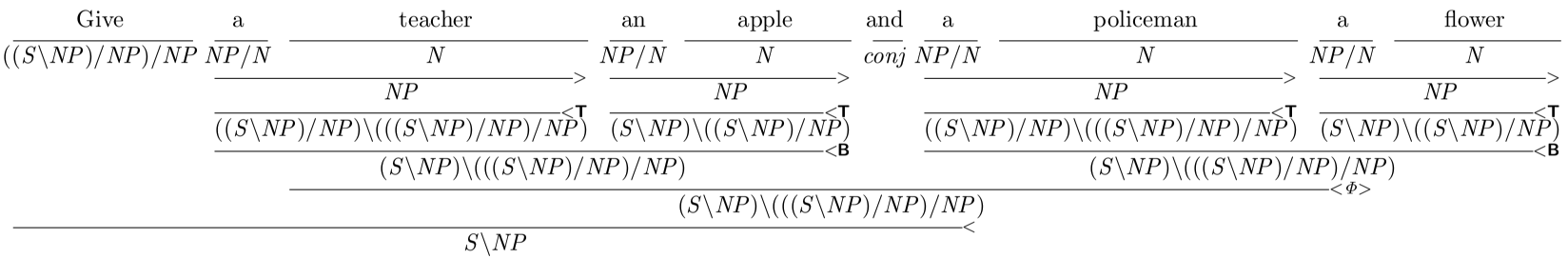


FIGURE 3.3: Analysis illustrating non-constituent coordination.

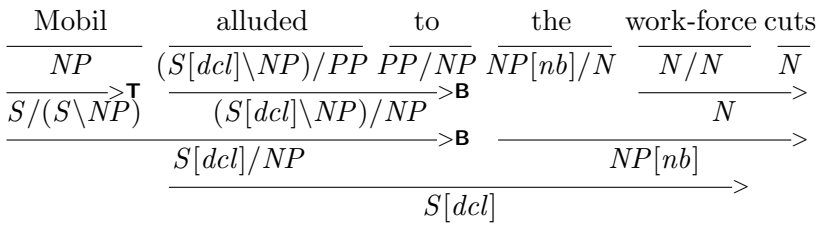


FIGURE 3.4: The expressive power of CCG leads to multiple analyses for even simple sentences.

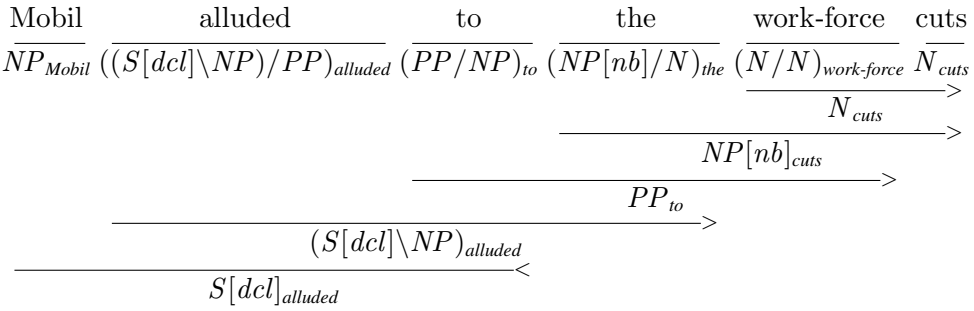


FIGURE 3.5: CCG analysis of sentence (8) showing constituent heads as a subscript on the corresponding categories.

stipulate that the result of a forward composition cannot be the left-hand category involved in another forward composition or forward application, and likewise the result of a backward composition cannot be the right-hand category in another backward composition or backward application. In a grammar without type-raising, these constraints eliminate the spurious ambiguity altogether (Eisner, 1996), and although the implementation of CCG in the parser we use contains type-raising rules, the inclusion of Eisner constraints substantially reduces the parser search space (Clark and Curran, 2004).

3.1.3 Dependencies in CCG

In addition to constituent structure, CCG analyses incorporate dependency information. This is accomplished through the use of markup on CCG categories. This section will describe the two kinds of category markup, representing head and dependency information, using sentence (8) as a running example.

As discussed in Section 2.1, each constituent or phrase contains a single word which can be identified as the head of the phrase. For example, Figure 3.5 shows the CCG derivation of sentence (8) with the head of each constituent marked in subscript on the category.

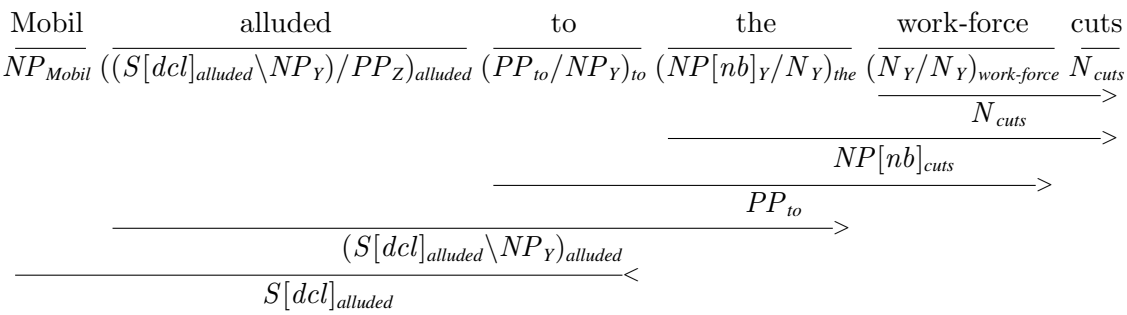


FIGURE 3.6: CCG analysis of sentence (8) showing all variables.

At the word level, each word must be its own head. When combining constituents through the application of rules, however, a choice must be made as to which word becomes the head of the new constituent. Usually this choice is between the heads of the constituents combined by the rule. To indicate how this choice should be made, CCG categories include markup in the form of variables on each subpart of the category. In this markup, the variable X is used to refer to the word that bears the CCG category, while other variables refer to the heads of the arguments that it collects in the derivation.

For example, the category $NP[nb]/N$ on the word *the* is annotated $(NP[nb]_Y/N_Y)_X$. This indicates that *the* combines with a constituent to the right with category N and head Y to produce a constituent with category $NP[nb]$ and head Y.

Similarly, the category of the word *alluded* is annotated $((S[dcl]_X\backslash NP_Y)_X/PP_Z)_X$, which indicates that *alluded* is a constituent with head X (=*alluded*) that combines with a constituent to the right with category PP and head Z to form a constituent with category $S[dcl]\backslash NP$ and head *alluded*, which then combines with a constituent to the left with category NP and head Y to produce a constituent with category $S[dcl]$ and head *alluded*.

Figure 3.6 shows the derivation in Figure 3.5 with all the variables on the categories included. Consider the step when *the* with category $(NP[nb]_Y/N_Y)_{the}$ combines with *work-force cuts* with category N_{cuts}. When these two categories are combined by forward application, the constituent N_{cuts} is identified as being the N_Y argument of *the*, and the variable Y on *the* is unified with the variable X (=*cuts*) on the constituent *work-force cuts*, setting the value of Y on *the* to *cuts*. Since the resulting category $NP[nb]$ is also marked as having head Y, the resulting constituent *the work-force cuts* has the head *cuts*.

The second form of markup on CCG categories indicate the dependencies that are to be formed. Every word that creates dependencies will bear a category with a number of *slots*, each associated with a variable on the category. These slots are indicated by subscript numbers paired with its associated variable. For example, *alluded* has two argument slots, one for its subject noun phrase and one for its prepositional phrase complement. This is represented in the category markup as $((S[dcl]_X\backslash NP_{Y,1})_X/PP_{Z,2})_X$. The dependencies are created during the derivation as argument slots become filled (in this case, when the NP and PP arguments combine with *alluded*).

The dependencies themselves are 5-tuples $\langle h_f, f, s, h_a, l\rangle$. Here, f is the CCG category which creates the dependency, in this example $(S[dcl]\backslash NP)/PP$. h_f is the word that bears that category; it is the value that fills the variable X on the category f, in this example *alluded*. s specifies which slot is being filled and is one of the numbers that appears on f as a subscript, for example 1. h_a is the head word of the constituent filling slot s; it will be the value of the variable associated with the slot s, in this example *Mobil*. 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 some distance part, caused for example by *wh- extraction* in the formation of *relative clauses*. If the dependency is long-range, then l contains the CCG category via which the dependency was formed. For sentence (8), *alluded* yields two dependencies, both local:

$$\langle \textit{alluded},\ (S[dcl]\backslash NP_1)/PP_2,\ 1,\ \textit{Mobil},\ -\rangle$$

$$\langle \textit{alluded},\ (S[dcl]\backslash NP_1)/PP_2,\ 2,\ \textit{to},\ -\rangle$$

An example of a long-range dependency can be seen in the analysis of (10), pictured in Figure 3.7.

(10) The work-force cuts that Mobil alluded to

In this sentence, a long-range dependency forms between *to* and *cuts*, mediated by the word *that*. The dependency is represented

$$\langle \textit{to},\ PP/NP_1,\ 1,\ \textit{cuts},\ (NP\backslash NP)/(S[dcl]/NP)\rangle$$

Finally, variables may contain multiple values, in order to represent coordination. For example, in sentence (11) (Figure 3.8), the coordination of the noun phrases *stocks* and *bonds* results in the creation of a constituent *stocks and bonds* with two values for its head variable.

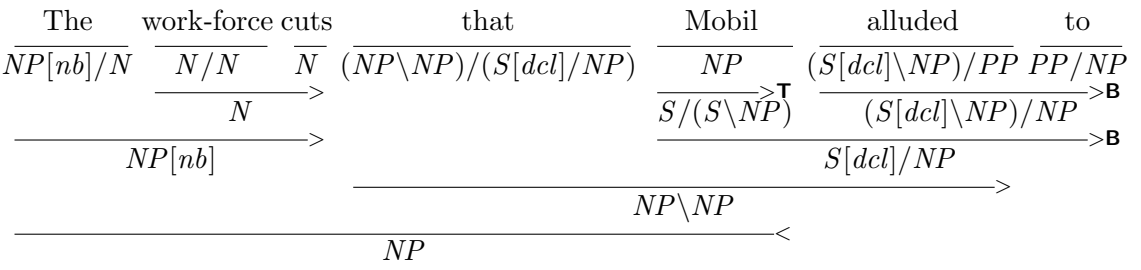


FIGURE 3.7: CCG derivation of sentence (10), which creates a long-range dependency.

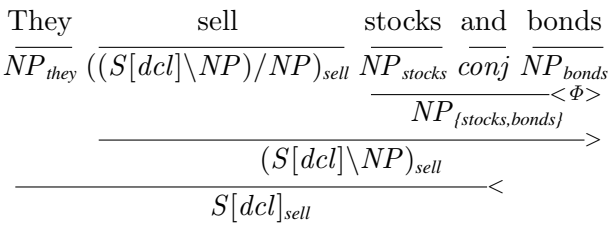


FIGURE 3.8: CCG derivation illustrating multiple variable values in coordination.

(11) They sell stocks and bonds.

3.2 CCGbank

The main resource for English parsing with CCG is called CCGbank, a corpus derived from the Wall Street Journal section of the Penn Treebank via a semi-automated process (Hockenmaier, 2003). Almost all sentences were in the Penn Treebank were successfully converted; a few dozen sentences in the Penn Treebank are absent from CCGbank. Some additional issues arise as a result of the conversion process.

Firstly, all quotation marks were removed from Penn Treebank sentences prior to conversion and as such any parser trained on CCGbank has no data on which to base decisions about quotation marks encountered during parsing. There has been some work on reintroducing quotation marks to CCGbank (Tse and Curran, 2007).

Secondly, the Penn Treebank annotations avoid marking any internal structure in noun phrases, since it is difficult to give a structure that is acceptable to all linguistic theories. As a consequence of this, there was no evidence available to the conversion process to determine how the structure of each noun phrase should be annotated. All noun phrases in CCGbank are therefore uniformly structured as strictly right-branching trees, regardless of the actual construction of the phrase. As a result, parsers trained on CCGbank are unable to effectively analyse noun phrase structures. Vadas and Curran (2007) looked at

reintroducing noun phrase structure to the Penn Treebank; until this is transferred to CCGbank, noun phrase structure will continue to be a weakness of this resource.

In addition to these issues, it is important to note that the analyses in CCGbank do not always conform strictly to the CCG formalism. These include changes to the representation of the coordination rule and the addition of a number of non-CCG rules.

As described in Section 3.1.2, the coordination rule in the only ternary rule in CCG, combining two elements with category X plus a $conj$ element to form a single constituent with category X. In CCGbank, coordination is instead represented as the following pair of binary rules.

$$conj \quad X \Rightarrow X[conj]$$

$$X \quad X[conj] \Rightarrow X$$

Finally, CCGbank includes a number of non-CCG rules. In pure CCG, the only unary rules are the two type-raising rules; all other rules combine two (or three, in the case of coordination) constituents. In CCGbank, however, there exist a number of additional unary rules, typically to allow some constituent to act as a modifier of another constituent. One example, taken from Hockenmaier (2003), is (12), where *advertising imported cigarettes* is a non-finite verb phrase with category $S[ng]\backslash NP$, acting as a modifier of the noun phrase *signboards*.

(12) signboards advertising imported cigarettes

To act as a noun phrase modifier in this way, the constituent's category $S[ng]\backslash NP$ would need to become $NP\backslash NP$. In pure CCG, this would be accomplished by the word *advertising* having $(NP\backslash NP)/NP$ as a possible category in addition to $(S[ng]\backslash NP)/NP$. Since essentially every constituent $S[ng]\backslash NP$ may similarly function as a noun phrase modifier, every category based on $S[ng]\backslash NP$ would likewise need an alternative based on $NP\backslash NP$. This would lead to a massive increase in the number of CCG categories, so instead CCGbank uses a unary rule to allow a constituent with category $S[ng]\backslash NP$ to change to the category $NP\backslash NP$. Other non-CCG rules in CCGbank include rules to handle punctuation and a rule that allows a noun phrase to consist of a single noun with no determiner or modifiers.

3.3 C&C Parser

The C&C parser (Clark and Curran, 2007) is a state-of-the-art parser based on the CCG formalism, with language models estimated using a supervised algorithm, trained on CCGbank. In addition to producing state-of-the-art results, the implementation is highly efficient, showing that parsing with CCG is practical, even with its spurious ambiguity (Clark and Curran, 2007). This section will explain how the C&C parser's implementation of CCG differs from pure CCG, the process involved in training the parser, and finally the parsing process itself, consisting of three main phases: supertagging, parsing and output.

3.3.1 Implementation of CCG

The grammar used in the C&C parser is automatically acquired from CCGbank. As a result, it does not strictly conform to CCG as it was described in Section 3.1, primarily due to the inclusion of non-CCG rules mentioned above. However, it also does not entirely match the annotations in CCGbank; some analyses in CCGbank cannot be produced by the C&C parser.

In Section 3.1, we described the ternary coordination rule in CCG according to Steedman (2000). In Section 3.2, we described how this rule had been replaced by a pair of binary rules in CCGbank. The implementation of coordination in the C&C parser is different again. The first of the CCGbank coordination rules was

$$conj \;\; X \Rightarrow X[conj]$$

The C&C parser instead uses

$$conj \;\; X \Rightarrow X \backslash X$$

which allows the second step of the coordination to proceed as a standard backward application, with one slight difference—the $X \backslash X$ category produced is marked with a special *conj* marker.

In Section 3.1, we also mentioned that the type-raising rules may operate with any category in place of T, although languages may place certain restrictions on what categories may be used. The C&C parser differs from pure CCG in that it does not allow this rule to operate arbitrarily; instead, the allowed type-raising options are hard-coded, generating only the instances of type-raising seen in CCGbank.

Finally, the C&C parser differs from CCGbank because not all of the non-CCG rules included in CCGbank have been implemented. This reflects the fact that some of these rules are noisy, a result of unusual structures in the Penn Treebank, or difficulties with the semi-automatic conversion to CCG analyses.

3.3.2 Estimating the language model

Section 2.4 introduced the notion of a parser's *language model* and explained that the function of the language model is to provide estimates of the probability of the analyses that the parser suggests for a given sentence. Clark and Curran (2007) describe three different models for the C&C parser; one of these involves calculating probabilities over the constituent structure component of the analysis, while the other two operate on the dependency structure.

All three of the C&C parsing models are *maximum entropy* or *log-linear* models. Maximum entropy models assign probability mass to events seen in the training data in order to match the observed frequencies as closely as possible, and distribute probability mass uniformly for unseen events. This is accomplished by selecting a number of *features* in the training data and setting the probabilities of these features in the model to be equal to the relative frequency with which they have been observed. These observed frequencies are constraints on the parameters of the model, and may be satisfied by a number of different models; the maximum entropy model is the one out of these possibilities that assigns probability mass uniformly to all unseen events (that is, the one that maximises the entropy of the model given the constraints).

The first of the C&C parsing models uses features from the constituent structure of the analyses including the words or parts of speech of words combined by particular rules. The probabilities of these features are calculated based on their observed frequencies in CCGbank sections 02–21, considering only *normal-form* derivations (see Section 3.1.2). This model is referred to as the *normal-form model*.

The two dependency-based models use features from the sets of dependencies produced by the analyses in CCGbank sections 02–21. In the first dependency model, the probabilities are calculated across all constituent structures leading to the set of dependencies; in the second dependency model, known as the *hybrid model*, the constituent structures used for calculating the probabilities are restricted to normal-form derivations in order to exploit some of the computational benefits of the normal-form model. Clark and Curran (2007) describe the training process of the parser and the features used in more detail.

Clark and Curran (2007) outline results for all three models according to the primary method of evaluation for the parser, to be discussed in Section 3.4. In this evaluation, the dependency-based models perform slightly better than the normal-form model. However, they are also more computationally expensive to train, so in the work that follows we use only the normal-form model.

3.3.3 The supertagger

The first component of the C&C parsing system is called the *supertagger*. The function of the supertagger is to assign the most likely CCG categories to each word in a given sentence, based partly on their part of speech tags and partly on the CCG categories observed for the word in CCGbank.

The supertagger takes two parameters, β and k, which respectively govern the number of tags returned for each word and the cut-off point at which a word is considered rare. The supertagger is tightly integrated with the parser so that if the parser cannot find an analysis using the categories assigned, or if the categories lead to so many possibilities that the parser runs out of space in the chart, then these two parameters can be adjusted and the supertagger recalculates the set of CCG categories for the sentence.

For each word, the supertagger will first determine how often the word has been seen before. Any word that was seen fewer than k times in the training data (CCGbank sections 02–21) is considered rare and is assigned CCG categories based purely on its part of speech. By default, the C&C parser uses a value of $k = 20$, but increases this to $k = 100$ if the parser cannot find a spanning analysis for the sentence using the lower cut-off.

If the word is seen k or more times, the supertagger returns the most likely category from that word, according to the training data. In addition, the supertagger returns all CCG categories that have been seen with the word with a relative frequency greater than βp_1, where p_1 is the probability of the most likely category. β will typically be given values between 0.0001 and 0.075; a higher value for β means a smaller number of categories returned.

3.3.4 The chart parsing algorithm

Once CCG categories have been assigned to the words in the sentence, control passes to the chart parsing algorithm. First, the chart is initialised by filling the first row of cells with the CCG categories assigned to each word by the supertagger. Because CCG is a binary branching grammar, the CKY chart parsing

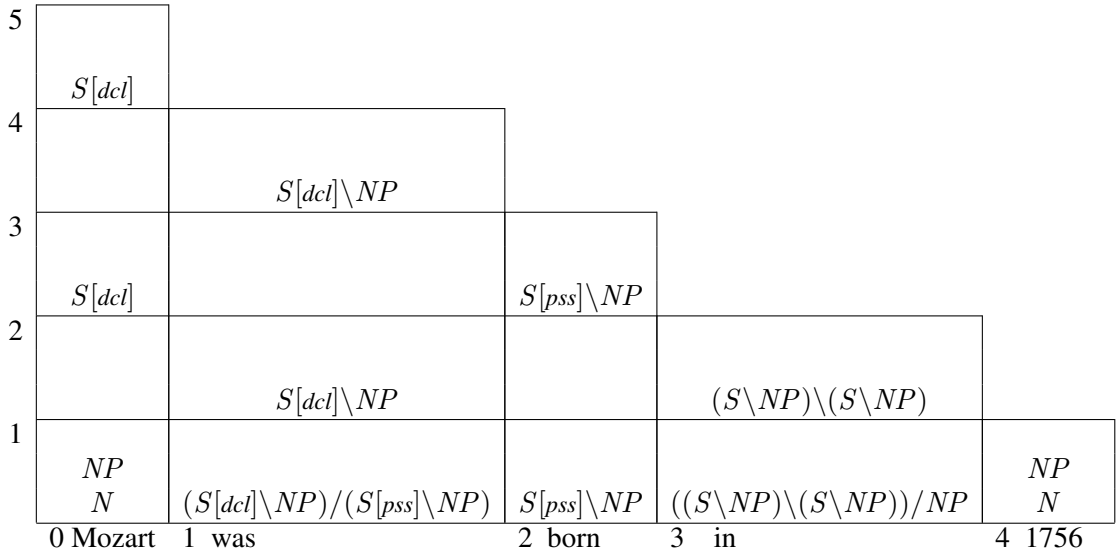


FIGURE 3.9: CCG equivalent of the chart in Figure 2.6.

algorithm described in Section 2.5 can be applied naturally. Steedman (2000) describes how the CKY algorithm applies to CCG.

The CCG equivalent of the chart given in Figure 2.6 is given in Figure 3.9. To make the diagram simpler, many of the values in the cells have been left out. For example, due to type-raising rules, any cell containing the category NP is augmented with seven additional categories, ranging in complexity from $S/(S\backslash NP)$ and $S/(S/NP)$ to $((S\backslash NP)/(S[adj]\backslash NP))\backslash(((S\backslash NP)/(S[adj]\backslash NP))/NP)$.

At the conclusion of the chart parsing algorithm, if the topmost cell in the chart contains no nodes, the parser has found no analyses for the sentence. In this case, the system can return to the supertagging phase with a lower value for β or a higher value for k and begin again. Conversely, if there are too many possible analyses and the chart becomes too full, the system can return to the supertagger with a higher value for β or a lower k. The possible values for β and k that the system tries are set as parameters; if no analysis is found using any of these values, the parser is unable to find an analysis for the sentence.

3.3.5 Parser output

Having filled in all the cells in the parse chart, the next stage of the parsing process is to calculate which analysis in the chart is the most probable. The *decoder* examines each node in the top-most cell of the chart (in Figure 3.9, this is cell $(0, 5)$) and calculates for each node the probability of the corresponding analysis according to one of the language models estimated in Section 3.3.2. These probabilities can be

used to compare analyses of a single sentence, but are not meaningful in the comparison of two different sentences.

After the decoder selects the most likely analysis from all of the analyses found by the parser, the *printer* outputs the analysis to disk, in one of a number of possible formats. First, the printer may output the CCG constituent structure found for the sentence. This can be done in as a Prolog-style bracketed string, or in a style matching the layout of the original CCGbank files. Alternatively, the printer may list the CCG dependencies constructed by the CCG analysis. Finally, these CCG dependencies can be converted to grammatical relations under the RASP GR scheme, for the purposes of comparison with other parsers, as described in the next section. Example output of the system is given in Figures 3.10 to 3.13, for sentence (13).

(13) Mozart was born in 1756.

3.4 Evaluating the C&C parser

The field standard evaluation of PARSEVAL metrics over Penn Treebank bracketings is problematic for CCG for two reasons. Firstly, CCG parse trees are frequently quite a different shape from those in the Penn Treebank, most notably due to the fact that CCG trees are binary while Penn Treebank analyses are typically very flat. Secondly, CCG's spurious ambiguity can lead to derivations which have drastically different tree shapes and will therefore score poorly. However, these analyses generate the same dependencies between words and as such are equally valid, so they should not be penalised.

For these reasons, the C&C parser is evaluated using a dependency-based evaluation. That is, precision and recall are calculated over dependencies rather than tree structures. These values are defined as in Section 2.6.3. Clark and Curran (2007) make use of two separate gold standards in their evaluation, CCGbank, described in Section 3.2 and Briscoe and Carroll (2006) reannotation of the PARC Dependency Bank (DepBank), discussed in Section 2.3.2.

The main evaluation of the C&C parser involves comparing the parser output against the CCG dependencies in CCGbank. There are some differences between the dependencies created by the parser and those in CCGbank. Some additional dependencies output by the parser can simply be ignored; other differences mean that the C&C parser faces an upper bound of 99.80% precision and 99.18% recall in this evaluation (Clark and Curran, 2007).

```
:- multifile w/8, ccg/2, id/2.
:- discontiguous w/8, ccg/2, id/2.
:- dynamic w/8, ccg/2, id/2.

1 parsed at B=0.075, K=20
1 coverage 100%
ccg(1,
 rp('S[dcl]',
  ba('S[dcl]',
   lex('N','NP',
    lf(1,1,'N')),
   fa('S[dcl]\NP',
    lf(1,2,'(S[dcl]\NP)/(S[pss]\NP)'),
    ba('S[pss]\NP',
     lf(1,3,'S[pss]\NP'),
     fa('(S[X]\NP)\(S[X]\NP)',
      lf(1,4,'((S[X]\NP)\(S[X]\NP))/NP'),
      lex('N','NP',
       lf(1,5,'N')))))),
  lf(1,6,'.'))).

w(1, 1, 'Mozart', 'Mozart', 'NNP', 'I-NP', 'I-PER', 'N').
w(1, 2, 'was', 'be', 'VBD', 'I-VP', 'O', '(S[dcl]\NP)/(S[pss]\NP)').
w(1, 3, 'born', 'bear', 'VBN', 'I-VP', 'O', 'S[pss]\NP').
w(1, 4, 'in', 'in', 'IN', 'I-PP', 'O', '((S[X]\NP)\(S[X]\NP))/NP').
w(1, 5, '1756', '1756', 'CD', 'I-NP', 'I-DAT', 'N').
w(1, 6, '.', '.', '.', 'O', 'O', '.').
```

FIGURE 3.10: C&C parser output for sentence (13), Prolog format.

```
ID=1 PARSER=GOLD NUMPARSE=1
(<T S[dcl] 1 2>
  (<T S[dcl] 0 2>
    (<T NP 0 1>
      (<L N NNP NNP Mozart N>) )
    (<T S[dcl]\NP 1 2>
      (<L (S[dcl]\NP)/(S[pss]\NP) VBD VBD was (S[dcl]\NP)/(S[pss]\NP)>)
      (<T S[pss]\NP 1 2>
        (<L S[pss]\NP VBN VBN born S[pss]\NP>)
        (<T (S\NP)\(S\NP) 1 2>
          (<L ((S\NP)\(S\NP))/NP IN IN in ((S\NP)\(S\NP))/NP>)
          (<T NP 1 1>
            (<L N CD CD 1756 N>) ) ) ) ) )
  (<L . . . . .>) )
```

FIGURE 3.11: C&C parser output for sentence (13), CCGbank format. Line breaks and indentation have been added for clarity.

```
in_4 (((S[X]{Y}\NP{Z}){Y}\(S[X]{Y}<1>\NP{Z}){Y}){_}/NP{W}<2>){_}
     2 1756_5 0
in_4 (((S[X]{Y}\NP{Z}){Y}\(S[X]{Y}<1>\NP{Z}){Y}){_}/NP{W}<2>){_}
     1 born_3 0
was_2 ((S[dcl]{_}\NP{Y}<1>){_}/(S[pss]{Z}<2>\NP{Y*}){Z}){_}
     2 born_3 0
was_2 ((S[dcl]{_}\NP{Y}<1>){_}/(S[pss]{Z}<2>\NP{Y*}){Z}){_}
     1 Mozart_1 0
born_3 (S[pss]{_}\NP{Y}<1>){_} 1 Mozart_1 0
     ((S[dcl]{X}\NP{Y}<37>){X}/(S[pss]{Z}<38>\NP{Y*}){Z}){X}
<c> Mozart|Mozart|NNP|N was|be|VBD|(S[dcl]\NP)/(S[pss]\NP)
    born|bear|VBN|S[pss]\NP in|in|IN|((S\NP)\(S\NP))/NP
    1756|1756|CD|N .|.|.|.
```

FIGURE 3.12: C&C parser output for sentence (13), CCG dependencies format. Line breaks and indentation have been added for clarity.

```
(dobj in_3 1756_4)
(ncmod _ born_2 in_3)
(aux born_2 was_1)
(ncsubj born_2 Mozart_0 obj)
<c> Mozart|Mozart|NNP|N was|be|VBD|(S[dcl]\NP)/(S[pss]\NP)
    born|bear|VBN|S[pss]\NP in|in|IN|((S\NP)\(S\NP))/NP
    1756|1756|CD|N .|.|.|.
```

FIGURE 3.13: C&C parser output for sentence (13), RASP GR format. Line breaks and indentation have been added for clarity.

The second evaluation is primarily aimed at enabling a comparison of the C&C parser with other work. DepBank is used for this as it is a publicly available gold standard and the GR scheme involved has some similarities with the parser's CCG dependency output. In order to perform this evaluation, the C&C parser's dependency output must be converted into the RASP GR scheme. The mapping is a many-to-many mapping and was developed by comparing the gold standard dependencies in CCGbank section 23 with the grammatical relations in DepBank.

Firstly, one grammatical relation may map to several CCG dependencies. For example, the in sentences (14) and (15), the word *use* will be assigned two distinct CCG categories. As a result, the dependency formed between *use* and *resources* will differ between the two sentences. Under the grammatical relations scheme, the relationship between both sentences is the same.

(14) They used the available resources.

(15) They used the available resources as a starting point.

$$\langle \textit{used}, (S[\textit{dcl}]\backslash NP_1)/NP_2, 2, \textit{resources}, - \rangle$$

$$\langle \textit{used}, ((S[\textit{dcl}]\backslash NP_1)/PP_2)/NP_3, 3, \textit{resources}, - \rangle$$

```
(dobj used resources)
```

This part of the mapping shows that grammatical relations generalise better across sentences, a fact that we will make use of in our method later.

It is also the case that one CCG dependency can be mapped to two grammatical relations. For example, the second dependency on category $(S[\textit{dcl}]\backslash NP_1)/NP_2$ ordinarily becomes the `dobj` (direct object) relation; if, however, the word bearing this category is a form of the verb *be*, the grammatical relation produced will be `xcomp` (predicate complement).

3.5 Previous extensions to the C&C parser

In Chapter 7, we will be describing our new method for placing constraints on the analyses provided by the C&C parser, as part of a larger process aimed at eliminating the need for manual annotation of training data. Before discussing this process, it is interesting to consider two previous extensions to the parser that are related to our aim in different ways. The first extension, by Clark and Curran (2006), explores a way of reducing the annotation cost by training the parser on incomplete data, an approach called *partial training*. The second extension, by Djordjevic *et al*. (2007), is primarily aimed at increasing the efficiency of the C&C parser. Of interest here is one of the methods used—restricting the space of possible analyses based on automatically-identified constraints.

Since CCG is a lexicalized grammar, much of the grammatical information in a sentence is expressed in the categories assigned to individual words. Clark and Curran (2006) therefore investigate how these word-level CCG categories can be used to extract information about the dependencies in the sentence and thus infer the full parse tree. They hypothesise that many dependencies in the sentence can be reliably determined from the word-level categories alone, and that the parser can be trained using just these inferred dependencies.

The Clark and Curran (2006) procedure is as follows. Instead of passing fully parsed sentences into the system as training data, sentences passed in are annotated only with their word-level CCG categories. These categories create a *parse forest*, a large collection of possible parse trees. The system then selects the CCG dependencies that appear in $k\%$ of the trees in this forest, where k is some parameter that can

be adjusted to control the trade-off between the variety of dependencies selected and the probability of their accuracy. The reasoning is that, if $k = 100$, every dependency collected appears in every possible analysis using these word-level categories, and therefore must appear in the correct analysis, depending on the coverage of the grammar. If k is high but not quite 100, the dependencies are highly likely to be correct. The parser is then trained using the dependencies selected in this way, rather than the complete set of dependencies as would ordinarily be the case.

Using this approach, Clark and Curran (2006) train a model for the C&C parser and evaluate its performance using the primary method of evaluation for the C&C parser, comparing against dependencies in CCGbank (Section 3.4). They report their best results at $k = 85$, achieving an F-score only 1.3% lower than that achieved using fully-parsed training data. They observe that annotation of text with only the word-level categories is fast, and combined with the parser's remarkable performance, this annotation speed suggests that more data can be acquired relatively cheaply, with minimal reduction in performance.

In the second extension, Djordjevic *et al*. (2007) implement three enhancements to the C&C parser aimed at increasing its efficiency and therefore speed. Of these three enhancements, the one of interest here is the addition of constraints to the parse chart to reduce the size of the set of possible analyses for a sentence.

The constraints that Djordjevic *et al*. (2007) add target the constituent structure component of CCG and specify that certain sequences of words in the sentence must be grouped together to form a constituent. For example, in sentence (8) in Section 3.1, a constraint could be added that the span *to the work-force cuts* must form a constituent, therefore preventing the analysis in Figure 3.4. The constraint that a particular sequence of words must form a constituent is equivalent to requiring that a particular cell in the parse chart appears as a node in the final analysis. Since constituents cannot partially overlap each other, this requirement then affects the validity of other cells, as illustrated by the diagram in Figure 3.14, taken from Djordjevic *et al*. (2007).

Djordjevic *et al*. (2007) investigate whether useful constraints can be automatically identified from features of the sentence. The constraints they use are based on punctuation and on noun phrase *chunks* identified by an earlier component of the parser. They found that constraints based on noun phrase chunks were of limited usefulness, while those based on punctuation provided good results, suggesting that punctuation can be useful in indicating how a sentence should be analysed.

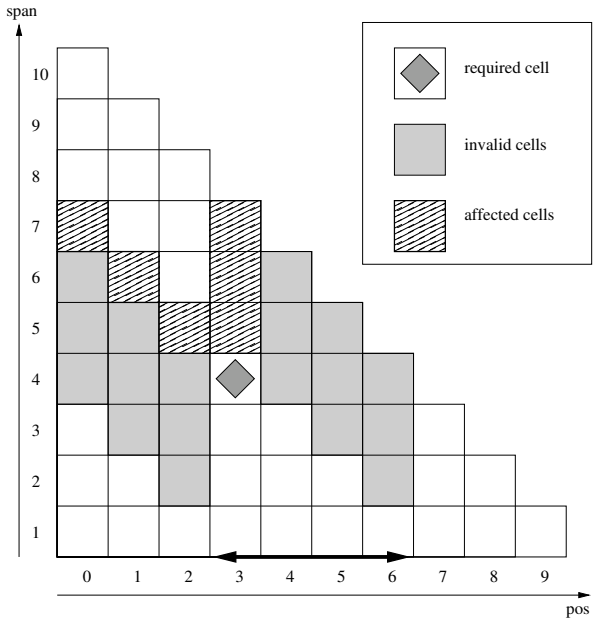


FIGURE 3.14: Specifying that a particular sequence of words must form a constituent affects which cells of the parse chart may be used in the analysis of the sentence. This diagram is taken from Djordjevic *et al*. (2007); *pos* indicates the starting position and *span* the length of the sequence of words.

3.6 Summary

In this chapter, we described Combinatory Categorial Grammar, a grammar formalism incorporating both constituent structure and dependency information in its analyses of sentences. We also described CCGbank, a corpus containing CCG analyses, derived from the Penn Treebank, and the operation and evaluation of a state-of-the-art parser (C&C) that analyses sentences according to a slightly modified form of CCG, extracted from CCGbank. Finally, we introduced two extensions to the C&C parser, which we will return to when we describe our new extension. In the next chapter, we will consider how characteristics of the Penn Treebank limit the performance of any parser trained on it or one if its derivative corpora, including the C&C parser. We then present our novel solution to this problem.

CHAPTER 4

The Annotation Bottleneck

As described in Section 2.4, supervised approaches to parsing, where the language model is estimated from an annotated corpus, still outperform unsupervised approaches, which are estimated from raw text alone. However, the requirement for annotated data is a major bottleneck; the cost of annotation limits the amount and variety of data available. In this chapter, we examine the disadvantages of the Penn Treebank, and the cost of annotating new corpora. We then introduce the concept of semi-supervised learning. Finally, we look at how the Web has been used in language processing tasks, before outlining how we use the Web to implement a semi-supervised method for addressing this annotation bottleneck.

4.1 Drawbacks of the Penn Treebank

Although the emergence of the Penn Treebank as a field standard has been beneficial in the development, evaluation and comparison of English parsing systems, the corpus is limited in genre and size. These limitations then significantly affect the performance of any parser trained solely on the Penn Treebank or one of its derived resources, including the C&C parser.

Biber (1993) investigates grammatical and lexical variation between genres and shows that different genres may have quite different probabilities for the same structures. This result suggests that a parser trained on one genre will underperform when analysing text from other genres. Empirical studies by Gildea (2001) and Sekine (1997) have borne out this prediction.

Gildea (2001) investigates the differences in performance between parsers trained on the Penn Treebank and parsers trained on the Brown corpus, a *balanced* corpus containing written texts from a range of fiction and non-fiction genres. He finds that training a parser on a small amount of data from the same genre as the test set is more useful than training on a large amount of data from a different genre, and that adding such unmatched training data neither helps nor hinders performance.

Similar findings are reported by Sekine (1997), who explores the difference in parsing performance across different domains from within the Brown corpus. Sekine finds that the best parsing accuracy is achieved using a grammar trained on the same domain as the test set, followed by the grammar trained on an equivalent size corpus containing texts from the same class (fiction/non-fiction), with grammars trained on a different class or different domain performing noticeably worse.

On the basis of the syntactic variation he observed between genres, Biber (1993) argues that general language studies, as opposed to studies of specific language varieties, must be based on corpora consisting of a number of genres. Since our aim is to build a wide-coverage parser capable of analysing a wide variety of texts, Biber's argument, supported by the empirical results, implies that training the parser on only the newswire text in the Penn Treebank will produce inadequate results.

In addition to restricted genre, the Penn Treebank is limited in size. Banko and Brill (2001) investigate how performance in another language processing task, *confusion set disambiguation*, is affected by training corpus size. Although they do not discuss parsing, their findings may have implications for the size of training corpora for parsing.

Confusion set disambiguation may be formulated as follows: given a text and a set of words (for example {*to*, *two*, *too*}, termed the *confusion set*, because even native English speakers can have difficulty choosing the correct word from the set), all instances of the words in the set are replaced with a single marker and the system must recover the correct alternative. Any reasonably edited raw text can be used to create annotated training data for this problem, so large amounts of training data can be generated extremely cheaply.

Using this task, Banko and Brill (2001) evaluate the disambiguation performance of four different supervised machine learners when trained on data sets up to 1 billion words in size, a thousand times larger than the Wall Street Journal section of the Penn Treebank. They find that even at these training corpus sizes, the systems have not yet begun to asymptote, and in fact all systems perform similarly, suggesting that the choice of learning algorithm becomes unimportant.

If these findings apply to parsing as well as to confusion set disambiguation, parser performance will continue to improve for some time as more training data is added. Whether our aim is to improve performance or simply determine whether Banko and Brill's findings apply to parsing, we must increase the amount of training data we have available. Unfortunately, supervised parsing cannot use raw text as annotated training data.

4.2 Annotation cost

Annotation is the most expensive part of the creation of new corpora, even when this process is partially automated. Marcus *et al*. (1993) give average annotation speeds for the Penn Treebank; we can use these to estimate the cost of annotating a new corpus of equivalent size. Marcus *et al*. (1993) give an average speed for correcting part of speech (POS) tags of 3000 words per hour, and for correcting skeletal syntactic analyses with no distinctions between arguments and adjuncts, 750 words per hour. For a corpus of 1.05 million words, roughly the size of the Wall Street Journal section of the Penn Treebank, this translates to 350 hours on POS tags and 1400 hours on the syntactic analyses. Assuming a single part-time annotator working at 3 hours a day, this translates to 117 weeks of annotation, or over two years. Assuming a cost of $40 per hour, the total annotation cost for this new corpus would be $70,000. Manually creating a 1 billion word corpus, like that used by Banko and Brill (2001), would be one thousand times as costly.

4.3 Semi-supervised learning

Banko and Brill (2001) suggest two strategies for increasing the amount of annotated training data without incurring full annotation costs. Both of these strategies involve a system trained on a small, annotated seed corpus which is used to process a large, unannotated corpus. In the first strategy, *active learning*, the system selects the examples from the unannotated data about which it is least certain, or will be most useful for learning in some other sense. These examples are passed to a human, who annotates the examples and adds them to the seed corpus. This process is repeated as often as desired, each time using the larger seed corpus. Active learning reduces annotation time by focussing human effort on the most useful examples. However, we are interested in removing the requirement for manual annotation completely. This is the aim of the second strategy suggested by Banko and Brill (2001), which they refer to as *weakly supervised learning*.

Weakly supervised learning, also called *semi-supervised learning*, is a broad term covering any approach combining elements of supervised and unsupervised learning. Banko and Brill (2001) use the term to refer to an approach where multiple systems are used to process the unannotated corpus. Examples which have been confidently processed, as indicated by agreement between several systems, are added to the seed corpus. The larger seed corpus is then used to retrain all of the systems involved, and the process repeats.

A similar approach, referred to as *cotraining*, is described by Mitchell (1999), in the context of classification of Web pages. In Mitchell's approach, two independent systems classify Web pages from a large set of unclassified pages. Each system then adds its most confident classifications to the seed corpus. Unlike the approach outlined by Banko and Brill (2001), addition to the seed corpus is not based on agreement between the classifiers but rather by each classifier's own confidence in the decision. What is most significant about Mitchell's strategy is that each of the two classifiers uses a different set of features, and that each set of features is "redundantly sufficient" to classify each page. It is this redundancy that enables the technique to succeed.

4.4 The Web in Natural Language Processing

In Sections 4.1 and 4.2, we established the importance of acquiring more training data for parsing and showed that the cost of acquiring it manually is prohibitive. In Section 4.3, we introduced semi-supervised approaches that use a large, unannotated corpus to extend a smaller, annotated seed corpus. We turn now to the large, unannotated corpus we propose to use here—the Web.

Kilgarriff and Grefenstette (2003) report that in an experiment conducted in March 2001 they used the frequency of occurrence of function words (for example *the* or *with*) to estimate the size of the Web. Their findings indicated that at that time the Web contained over 76 billion words of English text, with 7 billion words of German and six other languages represented by more than 1 billion words each. A more recent estimate of the amount of English Web text is over 1 trillion words, the claimed source of ngram counts in the Google Web 1T corpus (Brants and Franz, 2006). If we can harness the text on the Web, we will therefore have access to a corpus that dwarfs our existing training corpora.

The main criticism levelled at the use of the Web as a corpus is that it is not *representative*, meaning that conclusions drawn from it are not transferable to other corpora. Kilgarriff and Grefenstette (2003) explore this issue of representativeness, as well as the more basic question of whether the Web may be considered a corpus at all. They conclude that a corpus should be defined as any collection of texts "when considered as an object of language or literary study", a definition which the Web satisfies, and that the Web may indeed not be considered representative, but that our understanding of representativeness and of the notion of text type is so limited that no other corpus can be definitively said to be representative either. Considering that our aim is to be able to parse any text in computer-readable form, of which the

Web is a part, it seems that although, as always, we must be mindful of how our training data differs from our desired application, the Web is a reasonable choice for the source of our new training data.

Some early work involving exploiting the Web for natural language processing tasks involved estimation of frequencies of bigrams (pairs of contiguous words) that were previously unseen (Keller and Lapata, 2003). They compared counts determined from the Web with counts retrieved from the British National Corpus, a balanced corpus of 100 million words of text (90% written, 10% spoken), and the North American News Text Corpus, a 350 million word corpus of news text. Although the Web is a noisier corpus, Keller and Lapata (2003) found that its size compensates for this disadvantage and that the Web and corpus counts show statistically significant correlation. Furthermore, they find that the Web counts correlate with human judgements of plausibility and with counts recreated using standard smoothing techniques.

Keller and Lapata (2003) also carried out a disambiguation task that makes use of bigram counts and found that Web counts performed better than standard recreated counts. In these experiments, Web counts were retrieved using queries to the search engines AltaVista and Google, but since the search engine's method for determining these counts is unknown, their reliability is also unknown. This makes it all the more remarkable that these results suggest that the Web can provide seemingly reliable linguistic information.

In an extension to this work, Lapata and Keller (2005) investigate the effect on accuracy of the use of Web counts in several NLP tasks. They find that in unsupervised models Web counts improve performance over corpus counts, and that in some cases combining Web and corpus counts yields further improvements, but that these results are still generally eclipsed by state-of-the-art supervised models. This suggests that, despite promising initial work, bigram counts drawn from the Web are not sufficient to produce state-of-the-art results when incorporated in only simple systems. The question remains, however, whether more sophisticated information can be obtained from the Web with a semi-supervised approach, which is the strategy that we propose here.

Brill *et al*. (2001) exploited the size of the Web in their entry in the TREC (Text REtrieval Conference) *Question Answering* track (Voorhees, 2001). The aim of this task is to identify the answer to a given question within a given set of texts. Unlike current search engines, the system does not return a ranked set of documents that might contain the answer, but rather returns the precise string of words that answers the question, or a ranked list of such answers. The main evaluation score used in the competition is the

Mean Reciprocal Rank (MRR), which is calculated by identifying for each question the position of the correct answer in the list of answers the system provides, and taking the average of the reciprocals of these numbers (0 if the correct answer was not returned).

Brill *et al*. (2001) observe that for some questions, identification of the correct answer using only the given questions and texts may require deep linguistic processing and logical inference to connect the question to the answer. In contrast, the Web is so large that the same piece of information is likely to be expressed many times, and that many of these will be simple contexts in which the answer can be extracted with only simple linguistic processing. Therefore, in their system, Banko and Brill (2001) first identify an answer to the question on the Web, and then use this answer to locate the final answer in the given texts. Despite the simplicity of their approach, the system achieved an MRR of 0.35 under the strict evaluation scheme. This placed the team ninth at the competition, where only three systems achieved an MRR over 0.45 (Voorhees, 2001), a remarkable result.

4.5 Using the Web to overcome the annotation bottleneck

In this chapter, we have described the problem facing supervised approaches to parsing—the need for more training data containing prohibitively expensive annotations. In this thesis, we address this annotation bottleneck by using a semi-supervised approach to turn raw Web text into annotated training data. We start with a seed corpus of 39604 sentences, CCGbank (Section 3.2), and use a state-of-the-art parser, the C&C parser (Section 3.3), to annotate additional sentences from the Web. To do this, we exploit the property of the Web identified by Banko and Brill (2001), namely that its size increases the probability of finding a piece of information being expressed in varied and often simple contexts.

Our approach is based on the assumption that if two sentences contain the same set of relatively unambiguous and related words, such as *Mozart*, *born* and *1756*, then the sentences will refer to the same event and, further, that the words will be *related by the same grammatical relations*. This assumption is in some sense related to the *one sense per collocation* principle proposed by Yarowsky (1993), which states that in the context of particular neighbouring or grammatically related words, a polysemous word is highly likely to exhibit a particular one of its several possible meanings.

Our procedure for creating annotated training data is illustrated in Figure 4.1 and consists of the following steps. First, we select a number of different facts, such as those sought in the questions of the TREC Question Answering task, and identify the keywords of each (**A**). This is the only manual step in our

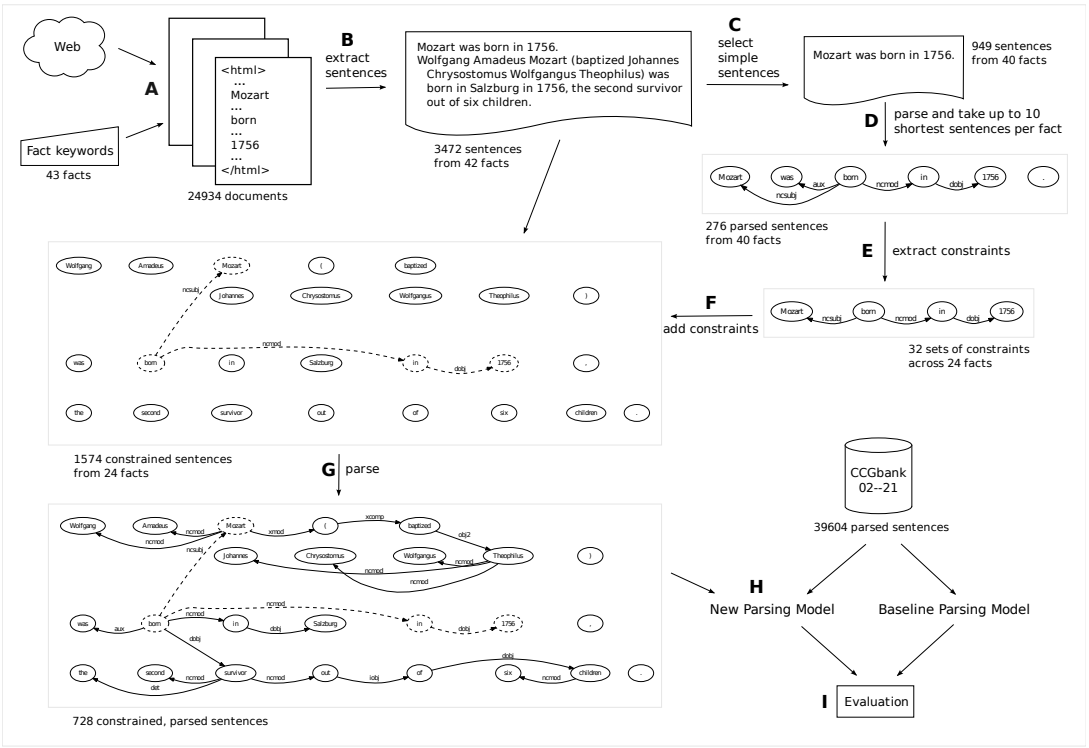


FIGURE 4.1: Outline of our method for automatically acquiring annotated training data.

process. Next, we use the fact keywords to collect HTML documents and then split the documents into sentences. Of these sentences, we keep those at most 40 tokens long that contain all of the keywords of the given fact (**B**). These sentences form our unannotated corpus. This stage of the process is described in Chapter 5.

Next, from the sentences we collect, we choose those that we consider easy to parse (**C**). We use the analyses of these simple sentences (**D**) to determine how the keywords of each fact should be related, as illustrated at **E**. Chapter 6 discusses how we determine which sentences are used for this process and the algorithm by which we extract the keyword relationships.

Finally, we parse all of the sentences collected (**G**), using the relationships extracted in the previous stage as constraints on the analyses (**F**). The application of these constraints is described in Chapter 7. We assume that if an analysis for a sentence can be found that is consistent with these relationships, then it is correct, and these sentences with their analyses become our annotated training data (**H**). In Chapter 8, we evaluate our new system in comparison with a baseline system using only CCGbank as training data (**I**).

CHAPTER 5

Building a Corpus from the Web

This chapter outlines the first component of our method, introduced in Section 4.5. Here, we describe the process used to collect an unannotated corpus of sentences from the Web. In Section 5.1, we discuss the facts that underpin our method, then in Section 5.2, the procedure by which we collect Web documents. In Section 5.3, we describe how these documents are split into sentences, and finally (Section 5.4) which of these sentences we select and how we prepare them for inclusion in the unannotated corpus.

5.1 Facts

It is not possible to use a computer system to generate its own training data without adding some extra information, as this would merely reinforce the system's current knowledge. To provide the extra information in our approach, we consider *facts* of the kind sought in the *factoid questions* of the TREC Question Answering track. Voorhees (2004) describe these questions as "fact-based, short answer question[s] such as *How many calories are there in a Big Mac?*."

The hypothesis upon which our approach is based is that whenever a sentence contains all of the keywords of a fact, the syntactic analysis of the sentence will connect those keywords by the same chain of grammatical relations. In order for this hypothesis to hold, we must adopt a principle similar to the *one sense per collocation* constraint (Yarowsky, 1993); essentially, we expect one fact per keyword set.

To support this principle, it is critical to choose facts that refer to relatively unambiguous entities. Using the factoid questions from TREC 2004 (Voorhees, 2004) and the ISI Question Answer Typology (Hovy *et al.*, 2002) as guides, we manually compiled a list of facts and expressed each as one or more keyword queries. A total of 43 queries were compiled; these are listed in Tables 5.1 and 5.2. The former lists various equative and locative facts, such as appositions and definitions (*X is Y* structures), while the

Fact	Keywords
Abbreviations	
AARP stands for American Association of Retired Persons	American Association of Retired Persons AARP
CNN stands for Cable News Network	CNN Cable News Network
OEM stands for Original Equipment Manufacturer	Original Equipment Manufacturer OEM
People	
The President of the AARP is Tess Canja	AARP President Tess Canja
Edmund Barton was the first Prime Minister of Australia	Barton Prime Minister Australia
Duke Ellington was an American composer	Duke Ellington American composer
Galileo Galilei was an astronomer	Galileo astronomer
James Wright was the inventor of Silly Putty	James Wright inventor Silly Putty
Jennifer Capriati is a tennis player	Jennifer Capriati tennis player
Mark Twain's real name was Samuel Langhorne Clemens	Mark Twain Samuel Langhorne Clemens
The king of Morocco is Mohammed VI	Mohammed VI king Morocco
The lead singer of Nirvana was Kurt Cobain	Nirvana lead singer Kurt Cobain
Terry Pratchett is an author	Pratchett author
Johan Vaaler was the inventor of the paper clip	Vaaler inventor paper clip
Dates	
Bastille Day is the 14th of July	Bastille Day 14 July
CNN's first broadcast was in 1980	CNN first broadcast 1980
The Great Depression lasted from 1929 to 1939	Great Depression 1929 1939
Places	
Canberra is the capital of Australia	Canberra capital Australia
The river Elbe runs through the city of Dresden	Elbe runs through Dresden
Olympus Mons is on Mars	Olympus Mons Mars
New Delhi is the capital of India	capital India New Delhi
Other	
Gordon Gekko was the main character in *Wall Street*	Gordon Gekko main character Wall Street
Horus was the Egyptian god of the sky	Horus Egyptian god Horus god sky
James Dean's first move was *Fixed Bayonets*	James Dean first movie Fixed Bayonets
Saturn is the sixth planet from the Sun	Saturn sixth planet Sun
The Tale of Genji was written by Muraaki Shikibu	Tale of Genji Murasaki
A cataract is a clouding of the lens of the eye	cataract clouding lens eye
Hydrogen is the lightest element	hydrogen lightest element
A nematode is a roundworm	nematode roundworm
Platinum is a chemical element	platinum chemical element
Prions are made of proteins	prions made of proteins
A vertebrate is an animal with a backbone	vertebrate animal backbone

TABLE 5.1: Facts used to generate the training corpus and the corresponding keyword queries: Equative and locative structures

Fact	Keywords
Active	
Armstrong landed on the Moon in 1969	Armstrong landed Moon 1969
Christopher Columbus discovered America	Columbus discovered America
Edmund Hillary climbed Everest	Edmund Hillary climbed Everest
James Dean died in a car crash	James Dean died car crash
Marie Curie discovered radium	Marie Curie discovered radium
Stanley Prusiner discovered prions	Prusiner discovered prions
Passive	
The Black Panthers were founded in 1966	Black Panthers founded 1966
The comet Hale Bopp was discovered in 1995	Hale Bopp discovered 1995
Martin Luther King, Jr. was assassinated in 1968	Martin Luther King Jr assassinated 1968
Mozart was born in 1756	Mozart born 1756

TABLE 5.2: Facts used to generate the training corpus and the corresponding keyword queries: Events

latter lists the event-based facts, some expressed in the active voice (*X does Y*) and some in the passive voice (*Y is done by X*).

5.2 Document collection

Each of the 43 sets of keywords determined above was submitted as a query through the Google SOAP Search API. The query was restricted to files with the extension `.htm` or `.html` to create a more homogenous set of documents to simplify document processing by avoiding PDF and Microsoft Word documents, which Google also indexes. Each unique page returned was saved.

Document collection proceeds by specifying a start index and a number of documents to be returned; the search API restricts the latter to no more than ten, and the sum of start index and number of documents to no more than 1000. That is, documents are retrieved in lots of ten up to a maximum of 1000 results. Document collection ceases when this maximum is reached or when the list of results returned for a particular lot is empty.

5.3 Sentence identification

There are sections in HTML documents whose function is largely to provide metadata or formatting instructions, and carry little in the way of content. Such sections are unlikely to contain useful sentences,

html	body	div	p	hr	blockquote
h1	h2	h3	h4	h5	h6
table	tr	td	dl	dt	dd
ol	ul	li	br *(except br soft)*		

TABLE 5.3: HTML tags identified as likely to indicate sentence boundaries.

and so were stripped out in the initial stages of document processing. The sections treated in this way were the `head`, `script`, `style` and `form` HTML blocks.

The process of segmenting a document into sentences is known as sentence boundary identification. This is a non-trivial task that is complicated by use of periods for purposes other than sentence boundary marking, mainly abbreviations. The task is, however, further complicated here by the fact that material on the Web is rarely edited, uses punctuation erratically, and may contain sentence fragments which may or may not be marked with punctuation. We use the sentence boundary identification algorithm available in NLTK v0.9.3[1], an implementation of the highly successful unsupervised Punkt algorithm (Kiss and Strunk, 2006).

Although successful, the Punkt algorithm is designed to operate on plain text and does not take HTML markup into consideration. Simply stripping out all HTML tags prior to the segmentation process, however, would be unwise as some tags, for example heading delimiter tags such as `<h1>` and `</h1>`, are likely to indicate sentence boundaries in place of, rather than in addition to, standard punctuation. To account for this, we did not pass whole documents to the sentence identification algorithm. Instead, we divided each document into chunks by splitting on certain HTML tags and passed each chunk to the sentence identification algorithm separately.

The HTML tags chosen as sentence breaks, listed in Table 5.3, were identified manually based on intuitions that they are more likely to separate sentences than divide them. No distinction was made between begin and end tags. Although list and list item markers may form part of a coherent sentence, it was decided to have extra divisions rather than attempt to reconstruct the list in an in-sentence form.

It is unclear what effect the decision to divide the documents into chunks has had on the quality of the sentence boundary identification process. Ideally this process would use a boundary identification algorithm that takes HTML markup into account rather than the heuristic used here, e.g. Liu and Curran (2006).

[1] http://nltk.sourceforge.net

Before being passed to the sentence identification algorithm, each chunk was processed to remove any remaining HTML markup. All remaining HTML tags were stripped from the text and HTML character entities such as ` ` and `>` were replaced with their character equivalents according to Python's `htmlentitydefs` module. The chunk is then finally passed to the sentence identification algorithm, which splits each chunk into sentences.

5.4 Sentence selection

Having divided each document into sentences, the last stage in sentence collection is to choose which sentences to keep and prepare them for the next phase of the corpus creation process.

Because the high potential for *noise* amongst sentences retrieved from the Web, we use a fairly aggressive pruning strategy. We are not concerned about losing a large number of sentences, because there are a vast number of sentences and facts that we have not yet begun to consider. Firstly, sentences are only kept if they consist entirely of standard ASCII alphanumeric characters and punctuation from a restricted set, namely `!?,.:;-"'&()\/#$%`. We also considered to be noisy any sentences containing whitespace-separated sequences with fewer alphanumeric characters than punctuation.

Secondly, some HTML character entities, particularly those specified with Unicode code points rather than HTML entity names, may remain after the conversion using `htmlentitydefs` described above. Any sentences still containing HTML entities at this point are discarded.

Next, sentences are *tokenised*. Tokenisation is the splitting of text strings into units, roughly corresponding to words, for part of speech tagging and parsing. This is another non-trivial task, mainly complicated by the variety of uses for punctuation characters in written text. The tokeniser used was that developed for the C&C parser for the TREC 2007 competition Question Answering track (Bos *et al*., 2007). Sentences longer than 40 tokens were discarded as this might indicate noise or an error in sentence identification. The cut-off of 40 tokens was chosen since, as mentioned in Section 2.4, parser performance is typically evaluated on sentences up to 40 tokens long.

Finally, since the aim of this process is to extract sentences for each set of keywords, sentences were only kept if each of the keywords of the corresponding query appeared exactly once in the tokenised sentence. Clearly the sentence must contain every keyword at least once, as we aim to extract chains of

grammatical relations connecting every keyword; the limit of exactly once was used to avoid having to disambiguate between repetitions of keywords in any given sentence.

Once sentences were extracted from each document, the list was further reduced by discarding duplicate sentences. Although repetition of simple sentences should theoretically increase our level of confidence in their structures, the extraction of chains of grammatical relations is the only stage of the process to benefit from this redundancy, and indeed it would be inadvisable to trust duplicate sentences even at that stage in the event that the parser analyses the sentence incorrectly.

At the conclusion of this process, between 0 and 299 sentences remain for each query, giving a total of 3472 sentences. Ten queries produced fewer than ten sentences; the average yield of the remaining 33 queries was just over 100 sentences each. Table 5.4 specifies the figures for each query. In addition to the total number of documents retrieved, it lists the number of documents that yielded at least one sentence and the final number of sentences collected. The discrepancies between the latter two numbers stem from the fact that one document may yield many sentences and from the removal of duplicate sentences.

5.5 Summary

In this chapter, we have described the method by which we automatically collect a corpus of sentences from the Web, using keywords of facts that were identified manually. In the next chapters, we will describe how we automatically annotate sentences from this corpus to create new training data for the C&C parser, thus addressing the annotation bottleneck described in Chapter 4.

Keywords	# Documents	# Yielding sentences	# Unique sentences
American Association of Retired Persons AARP	708	202	165
CNN Cable News Network	286	75	62
Original Equipment Manufacturer OEM	573	135	214
AARP President Tess Canja	204	5	6
Barton Prime Minister Australia	629	36	34
Duke Ellington American composer	656	28	28
Galileo astronomer	674	204	243
James Wright inventor Silly Putty	257	3	2
Jennifer Capriati tennis player	509	14	13
Mark Twain Samuel Langhorne Clemens	558	238	291
Mohammed VI king Morocco	563	10	9
Nirvana lead singer Kurt Cobain	697	128	122
Pratchett author	534	135	160
Vaaler inventor paper clip	184	24	24
Bastille Day 14 July	827	163	181
CNN first broadcast 1980	657	2	2
Great Depression 1929 1939	547	21	27
Canberra capital Australia	601	75	78
Elbe runs through Dresden	334	4	4
Olympus Mons Mars	528	156	182
capital India New Delhi	473	83	94
Gordon Gekko main character Wall Street	665	2	2
Horus Egyptian god	681	87	90
Horus god sky	702	104	95
James Dean first movie Fixed Bayonets	717	0	0
Saturn sixth planet Sun	816	39	30
Tale of Genji Murasaki	543	121	132
cataract clouding lens eye	770	319	213
hydrogen lightest element	582	92	85
nematode roundworm	553	129	143
platinum chemical element	516	22	28
prions made of proteins	426	1	1
vertebrate animal backbone	682	33	34
Armstrong landed Moon 1969	805	21	20
Columbus discovered America	683	248	299
Edmund Hillary climbed Everest	548	30	29
James Dean died car crash	698	34	33
Marie Curie discovered radium	598	76	73
Prusiner discovered prions	254	10	9
Black Panthers founded 1966	707	14	13
Hale Bopp discovered 1995	580	37	35
Martin Luther King Jr assassinated 1968	675	6	6
Mozart born 1756	734	165	161
Total	24934	3411	3472

TABLE 5.4: Number of documents collected for each query, the number of documents that yielded at least one sentence, and the number of unique sentences gleaned from these documents.

CHAPTER 6

Identifying Constraints

Chapter 5 described the collection of a corpus of unannotated sentences from the Web. In this chapter, we consider how we use the parser's analysis of simpler sentences from this corpus to identify chains of grammatical relations connecting the keywords of the facts that the sentences represent. We refer to these chains as *constraint chains* since they will be used as constraints on parser analyses in Chapter 7.

We begin by determining constraint chains manually based partly on a visual inspection of the C&C parser's analyses of short sentences. Next, we describe how we can automatically select sentences that we expect to be able to parse reliably. Once these sentences have been chosen, we use the C&C parser's analysis to identify the chain of grammatical relations connecting the keywords in each sentence. Section 6.3 describes the algorithm by which these chains are identified. We conclude by comparing the constraint chains identified by the algorithm with the chains selected by hand.

6.1 Manual constraints

Initially, we examined the analyses of the ten shortest sentences for those facts where more than ten sentences were collected in total. Based on these analyses and on intuitions about how the words should be related, we manually identified one constraint chain for each of these facts. For example, we anticipated that the fact that *OEM* stands for *Original Equipment Manufacturer* would be expressed by a sentence containing the sequence *Original Equipment Manufacturer (OEM)*. The analysis of this sequence of words by the parser under the grammatical relations scheme would be

```
(ncmod Manufacturer Equipment)
(ncmod Manufacturer Original)
    (ncmod Manufacturer ()
        (dobj ( OEM)
```

These manually-identified constraints provide a kind of gold standard against which we can compare the constraints discovered automatically by our algorithm, described in Section 6.3. We will return to these manual constraints and give further examples in Section 6.4, where we compare the manual and automatic constraints for a sample of facts.

6.2 Simple sentences

The observation in Brill *et al*. (2001) suggests that a given fact will appear multiple times on the Web, sometimes in simple contexts. To explore this claim, and to exploit the simple sentences as we suggest, we first need a definition of a *simple sentence*. We proceed from the intuition that a simple sentence is short and consists of only a single *clause*. That is, the sentence does not contain any *subordinate clauses* such as *who went* in *He asked who went* or *relative clauses* such as *that had been taken* in *He found the book that had been taken*. Sentences are also only usefully simple if the parser can find an analysis for them.

It is difficult to choose a maximum length for simple sentences for all facts, since facts with more keywords automatically require longer sentences in order to express all the relevant information. Here, we take a rather large upper bound on sentence length, 20 tokens, and supplement this by restricting our attention in the next section to the ten shortest sentences that satisfy all our simplicity criteria. Another possibility would be to specify the maximum sentence length for a given fact as a function of the number of keywords for that fact.

We approximate our second criterion, that the sentence consists of only a single clause, by choosing sentences that contain exactly one finite (tensed) verb. We do this by considering the part of speech (POS) tags assigned to each word in the sentence prior to parsing. In the Penn Treebank POS tagset, which the C&C parser uses, finite verbs are indicated by the tags `VBD` (past tense, e.g. *ran* in *they ran*), `VBZ` (third person singular present tense, e.g. *he runs*) and `VBP` (other present tense, e.g. *they run*). In order to consider a sentence simple, we require that it contains exactly one token tagged with one of these three POS tags. This requirement is somewhat problematic because the POS tagging itself is open to error, for example due to many English verbs using the same form for both the past tense and the perfect participle, for example *looked* in *they looked* (past tense) and *they have looked* (perfect participle).

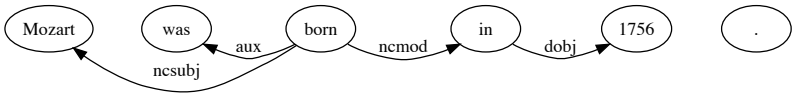


FIGURE 6.1: A simple, parsed sentence from which we will extract constraints.

Taking simple sentences to be those that we can parse that are no more than 20 tokens long and contain exactly one finite verb, Table 6.1 shows the division of the sentences collected into three sets—simple, unable to be parsed, and other. This table shows that the C&C parser is capable of finding an analysis (not necessarily correct) for most sentences collected, and that indeed a large number of these sentences are considered simple by the criteria given above. For most facts we found the same number or fewer simple sentences than other sentences. For the fact *a cataract is a clouding of the lens of the eye*, however, the majority of the sentences we collected were classed as simple. It would be interesting to do a thorough study of how different facts yield different distributions of simple and non-simple sentences.

6.3 Constraint chain identification

Having established which sentences for each fact can be considered simple, and therefore likely to have been correctly parsed, our next step is to extract from these sentences the chains of grammatical relations connecting the fact keywords, which will be used as constraints on syntactic analyses from the parser in the next stage of the process.

As described above, we place a high length bound on simple sentences, leading to a large number of such sentences for many facts. We therefore further restrict our attention to the ten shortest sentences in those facts where more than ten simple sentences are found.

The intuition behind our algorithm is as follows. We have a parsed, simple sentence, such as in Figure 6.1, containing the keywords of a fact, *Mozart*, *born*, and *1756*. We want to extract the grammatical relations, represented by arrows in the diagram, that connect these keywords, such that we can apply them to another sentence. We expect that the most transferable set of relations will be those that *most directly* connect the keywords. To find this set of relations, we draw on shortest path and minimum spanning tree algorithms. Having found a set of relations from one simple sentence, we would like to increase our confidence that the relations we have identified are independent of and so can be separated from the sentence from which we obtained them. To this end, we only accept a set of relations if we have seen it in multiple of our simple sentences.

Keywords	Simple	Other	Unparsed	Total
American Association of Retired Persons AARP	27	137	1	165
CNN Cable News Network	4	58	0	62
Original Equipment Manufacturer OEM	96	116	2	214
AARP President Tess Canja	0	6	0	6
Barton Prime Minister Australia	9	25	0	34
Duke Ellington American composer	7	21	0	28
Galileo astronomer	52	188	3	243
James Wright inventor Silly Putty	1	1	0	2
Jennifer Capriati tennis player	5	8	0	13
Mark Twain Samuel Langhorne Clemens	61	227	3	291
Mohammed VI king Morocco	3	6	0	9
Nirvana lead singer Kurt Cobain	27	94	1	122
Pratchett author	39	117	4	160
Vaaler inventor paper clip	8	16	0	24
Bastille Day 14 July	23	157	1	181
CNN first broadcast 1980	1	1	0	2
Great Depression 1929 1939	5	21	1	27
Canberra capital Australia	28	49	1	78
Elbe runs through Dresden	1	3	0	4
Olympus Mons Mars	45	133	4	182
capital India New Delhi	29	63	2	94
Gordon Gekko main character Wall Street	1	1	0	2
Horus Egyptian god	29	60	1	90
Horus god sky	35	58	2	95
James Dean first movie Fixed Bayonets	0	0	0	0
Saturn sixth planet Sun	11	18	1	30
Tale of Genji Murasaki	13	111	8	132
cataract clouding lens eye	113	99	1	213
hydrogen lightest element	23	60	2	85
nematode roundworm	39	103	1	143
platinum chemical element	1	26	1	28
prions made of proteins	0	1	0	1
vertebrate animal backbone	5	29	0	34
Armstrong landed Moon 1969	1	19	0	20
Columbus discovered America	57	234	8	299
Edmund Hillary climbed Everest	4	25	0	29
James Dean died car crash	10	23	0	33
Marie Curie discovered radium	23	50	0	73
Prusiner discovered prions	1	8	0	9
Black Panthers founded 1966	3	10	0	13
Hale Bopp discovered 1995	5	30	0	35
Martin Luther King Jr assassinated 1968	3	3	0	6
Mozart born 1756	72	88	1	161
Total	920	2503	49	3472

TABLE 6.1: Division of sentences for each query into simple, non-simple and unable to be parsed.

To identify the set of relations in the simple sentence, we treat the sentence as an undirected graph G, where nodes are words and edges are grammatical relations linking pairs of words. For the purposes of extracting grammatical relations connecting keywords, the direction of the edge (i.e. which word is the head of the relation) is unimportant.

Next, we construct a second undirected graph G', where the nodes correspond to the keywords of the fact. This graph we make a complete graph by connecting all pairs of keywords with an edge. For each pair of keywords x and y, we identify the shortest path connecting x and y in G. The length of this shortest path becomes the weight on the edge connecting x and y in G'. With each edge in G', we also store the path in G that generated the weight on that edge. If multiple shortest paths were found in G, the edge in G' will store each of these paths. Once this is process is complete, G' contains information about the shortest path in G between any pair of keywords.

Finally, we compute T, the minimum spanning tree of G'. This allows us to find the most direct paths connecting all of the keywords in the original graph representing the parse of the sentence G. Each edge of T corresponds to one or more paths in G'—more if two keywords were connected by two paths of equal length in G. We construct our final sets of relations by taking every combination of paths indicated by T. That is, if the keywords x and y are connected by two shortest paths in G, say p_1 and p_2, the edge between x and y in G' will both. If this edge is selected for T, we can generate two sets of relations for this sentence, one using p_1 and one using p_2. Therefore, depending on the relations expressed by edges in G, we may generate several sets of relations for a single sentence.

Although we may generate several sets of relations for a single sentence, we may not generate all possible sets. This is due to a limitation in the implementation of the computation of T, namely that the minimum spanning tree algorithm will make an arbitrary choice between two edges of the same weight. It would be advisable, in a reimplementation of this algorithm, to calculate all possible minimum spanning trees of G' and thus avoid this difficulty.

Having determined one or more sets of relations for each sentence in our set of up to ten simple sentences, we count the number of times each set of relations occurs. In order to accept a set of relations as a constraint chain, we require that it has occurred at least twice. In order to increase the reliability of each set of relations, we could increase this number, but an inspection of the constraints created, in Section 6.4, suggests that even a number as low as two performs adequately. It may be prudent to use a higher number if we increase the number of simple sentences that we consider in this process.

Keywords	# Chains
American Association of Retired Persons AARP	1
CNN Cable News Network	1
Original Equipment Manufacturer OEM	2
Barton Prime Minister Australia	1
Galileo astronomer	1
Jennifer Capriati tennis player	2
Mark Twain Samuel Langhorne Clemens	1
Nirvana lead singer Kurt Cobain	3
Pratchett author	2
Bastille Day 14 July	1
Canberra capital Australia	2
Olympus Mons Mars	2
capital India New Delhi	1
Horus Egyptian god	2
Horus god sky	1
Saturn sixth planet Sun	1
Tale of Genji Murasaki	1
cataract clouding lens eye	1
hydrogen lightest element	2
nematode roundworm	1
Columbus discovered America	1
Edmund Hillary climbed Everest	1
James Dean died car crash	1
Marie Curie discovered radium	2
Hale Bopp discovered 1995	1
Martin Luther King Jr assassinated 1968	1
Mozart born 1756	1
Total	37

TABLE 6.2: The facts for which our algorithm automatically identifies constraints, and the number of chains identified.

This algorithm generates constraints for 27 out of our original 43 facts. Since we have up to ten simple sentences per fact, and only require that a chain appears in two, for some facts we identify multiple sets of relations as valid constraint chains. Table 6.2 lists the facts for which chains were extracted, along with the number of chains identified for each.

6.4 Evaluation of automatic constraints

In order to evaluate the quality of the automatically-identified constraint chains, we performed a manual inspection to compare them with the manually-identified chains discussed in Section 6.1. In this section,

we present the automatic and manual constraints for a selection of our facts. Note that not all facts with automatic constraints have manual constraints, and vice versa.

When identifying constraint chains manually, we chose to consider only those facts with more than ten sentences, anticipating that in the automatic constraint identification we would make a similar restriction. However, when developing the automated process, this decision was reversed. As a result, the automatic procedure attempts to identify constraints for all facts, and so some facts have automatically-identified constraints but no manually-identified constraints.

The other mismatches in the comparison, that is, facts with manual constraints but no automatic constraints and facts with different constraint chains from the two methods, are due to two factors. Firstly, the automatic constraint identification algorithms require that candidate constraint chains must appear in a number of simple sentences in order to be accepted. Secondly, the manual identification process was ad hoc and not necessarily driven by parser analyses, so the manual constraints may not reflect actual expressions of the fact in simple sentences.

Tables 6.3 and 6.4 display the constraint chains identified by the two methods for a selection of facts, along with sentences or sentence fragments that would generate these chains. Note that these example sentences are illustrative only, and do not necessarily correspond to sentences in the corpus. A number of interesting points arise from these examples.

Firstly, we have suggested that the manual and automatic constraint chains for the keywords *Barton Prime Minister Australia* would be generated by the same sentence fragment, yet the manual chain contains two additional links that are not found in the automatic chain. This is due to the use of intuition in generating the manual constraints; the words *first* and *Edmund* do appear in the sentence fragment, but are not involved in connecting the fact keywords. It is not the case that the manual constraints always contain more information than the automatic; for example, the automatic constraints capture the fact that *Olympus Mons* is a volcano, and that Galileo's full name is *Galileo Galilei*.

It is also worth noting that the form of grammatical relations used here ignores possessive information. For example, the constraint chains make no distinction between *Australia's first Prime Minister* and the ungrammatical *Australia first Prime Minister*. Under the RASP GR scheme, the grammatical relation is more correctly represented `(ncmod poss Minister Australia)`, with the possession indicated by subtype information in the relation. In our current approach, we ignore subtype information in the conversion between CCG dependencies and grammatical relations.

Keywords	Manual	Automatic
Original Equipment Manufacturer OEM	ncmod Manufacturer Equipment ncmod Manufacturer Original ncmod Manufacturer (dobj (OEM *Original Equipment Manufacturer (OEM)*	ncmod Manufacturer Original ncmod Manufacturer Equipment xcomp is Manufacturer ncsubj is OEM *OEM is Original Equipment Manufacturer*
		ncmod Manufacturer Original ncmod Manufacturer Equipment dobj for Manufacturer ncmod stands for ncsubj stands OEM *OEM stands for Original Equipment Manufacturer*
Barton Prime Minister Australia	ncmod Minister Prime ncmod Minister first ncmod Minister Australia ncmod Barton Edmund conj , Barton conj , Minister *Australia's first Prime Minister, Edmund Barton*	ncmod Minister Prime ncmod Minister Australia conj , Barton conj , Minister *Australia's first Prime Minister, Edmund Barton*
Galileo astronomer	ncsubj was Galileo xcomp was astronomer *Galileo was an astronomer*	ncmod Galilei Galileo ncsubj was Galilei xcomp was astronomer *Galileo Galilei was an astronomer*
Nirvana lead singer Kurt Cobain	ncmod singer Nirvana ncmod singer lead ncmod Cobain singer ncmod Cobain Kurt *Nirvana lead singer Kurt Cobain*	ncmod singer lead ncmod Cobain Kurt dobj of Nirvana ncmod singer of conj , singer conj , Cobain *The lead singer of Nirvana, Kurt Cobain*
		ncmod Cobain Nirvana ncmod Cobain lead ncmod Cobain singer ncmod Cobain Kurt *Nirvana lead singer Kurt Cobain*
		ncmod singer lead ncmod Cobain Kurt dobj of Nirvana ncmod singer of xcomp was singer ncsubj was Cobain *Kurt Cobain was the lead singer of Nirvana*

TABLE 6.3: The constraint chains identified manually and automatically for a sample of facts.

Keywords	Manual	Automatic
Olympus Mons Mars	ncmod Mons Olympus dobj on Mars ncmod Mons on *Olympus Mons on Mars*	ncmod Mons Olympus ncsubj is Mons xcomp is volcano ncmod volcano on dobj on Mars *Olympus Mons is a volcano on Mars*
		ncmod Mons Olympus ncmod Mons on dobj on Mars *Olympus Mons on Mars*
Armstrong landed Moon 1969	ncsubj Armstrong landed ncmod landed on dobj on Moon ncmod landed in dobj in 1969 *Armstrong landed on the Moon in 1969*	—
Marie Curie discovered radium	ncmod Curie Marie ncsubj discovered Curie dobj discovered radium *Marie Curie discovered radium*	ncmod Curie Marie ncsubj discovered Curie dobj discovered radium *Marie Curie discovered radium*
		ncsubj discovered Marie ncsubj discovered Curie dobj discovered radium *Marie and (Pierre) Curie discovered radium*
Hale Bopp discovered 1995	ncmod Bopp Hale ncsubj discovered Bopp ncmod discovered in dobj in 1995 *Hale Bopp was discovered in 1995*	conj and Hale conj and Bopp dobj by Hale ncmod discovered by ncmod discovered in dobj in July ncmod July 1995 *. . . was discovered in July 1995 by Hale and Bopp*
Martin Luther King Jr assassinated 1968	—	ncmod Jr Martin ncmod Jr Luther ncmod Jr King ncsubj assassinated Jr ncmod Jr 1968 *1968 Martin Luther King Jr assassinated*
Mozart born 1756	ncsubj born Mozart ncmod born in dobj in 1756 *Mozart was born in 1756*	ncsubj born Mozart ncmod born in dobj in 1756 *Mozart was born in 1756*

TABLE 6.4: The constraint chains identified manually and automatically for a sample of facts (cont.).

Comparing the manual and automatic constraints for the sentence fragment *Nirvana lead singer Kurt Cobain*, we see the effect of the poor treatment of the internal structure of noun phrases in CCGbank, discussed in Section 3.2. Although the manual constraints identify *lead* and *Nirvana* as modifiers of *singer*, the parser attributes a right-branching structure to all noun phrases, giving *lead* and *Nirvana* as modifiers of *Cobain*, as if they were also the singer's first names, in addition to *Kurt*. This is an example where the use of automatic constraints is likely to reinforce the parser's existing knowledge, rather than introduce new knowledge, unless facts that might correct this mistake are explicitly targeted, for example, that focus on the *Nirvana lead singer* component and ignore the name *Kurt Cobain*.

Another example where the parser's existing models interfere with the identification of constraints is illustrated with the fact *Marie Curie discovered radium*. The C&C parser is inclined to analyse the sentence *Marie and Pierre Curie discovered radium* as *[Marie] and [Pierre Curie] discovered radium* rather than the correct *[Marie and Pierre] Curie discovered radium*. As a result, in addition to the correct constraint chain, our algorithm identifies a second chain corresponding to *Marie and Curie discovered radium*. To correct this, we suggest adjusting the definition of simple sentences to exclude sentences containing coordination structures.

Finally, the automatic constraint identification may result in a different interpretation of the fact than was originally intended. For example, we included the fact *The comet Hale Bopp was discovered in 1995*, an expectation reflected by our manual constraints. The automatic procedure, however, finds evidence for *The comet was discovered by Hale and Bopp*, an unexpected but correct piece of information.

6.5 Summary

In this chapter, we have presented a new algorithm for automatically extracting relationships between the keywords of a fact. We do this by identifying simple sentences and using the output of the C&C parser to identify grammatical relations that link the keywords most directly. An inspection of the chains of relations returned by the algorithm, in comparison with chains identified by hand, shows that the algorithm is largely successful at identifying appropriate chains. A few interesting cases also show that care may be required when a major weakness of the system is involved and that our definition of simple sentences is too permissive. Having automatically identified reliable relationships between fact keywords, we use these relationships to annotate additional training data for the parser, discussed in the next chapter.

CHAPTER 7

Annotating the Corpus

The previous chapters described the methods by which we use facts to create a corpus of sentences from the Web and by which we use the analyses of the simple sentences to identify chains of grammatical relations connecting the fact keywords. In this chapter, we describe how we use these chains of grammatical relations to constrain the analyses of all sentences collected in order to create annotated training data.

First, we describe a new method for constraining the parser's analysis and how it differs from the previous extensions to the C&C parser discussed in Section 3.5. In Section 7.2, we explain how the chains of grammatical relations are used with this new constraint method to force the parser to produce analyses consistent with the specified relations. Finally, we describe how we handle multiple such chains of relations for the same fact, and how the parser's analyses become annotated training data.

7.1 Constraining the parser

As described in Section 3.1, CCG dependencies are formed when the application of a combinatory rule combines two constituents and the value of a variable on one constituent is set to the head word of the second constituent. We can force the parser to generate particular dependencies by extending these variables with a second function, that of constraints. Now, in addition to *filled* and *unfilled*, a variable may exist in a third state, *constrained*.

Suppose we wish to parse (16), and that we wish to force the following dependency to be created between the words *work-force* and *cuts*.

$$\langle \textit{work-force},\ N/N_1,\ 1,\ \textit{cuts},\ - \rangle$$

(16) The work-force cuts that Mobil alluded to

In order to do this, we pass the sentence into the C&C parser, call the supertagger, and initialise the chart by filling in the cells in the bottom row with the categories assigned by the supertagger. We then locate the category N/N in the cell for the word *work-force*, which is annotated $(N_Y/N_{Y,1})_X$, and identify the variable associated with dependency slot 1, namely Y. We set the value of Y to *cuts* and mark it as a constrained variable. We then parse the sentence and pass constraints up to categories on larger constituents by the action of combinatory rules.

Before illustrating this process, we shall describe the behaviour of constrained variables with respect to variable unification. Like a filled variable, a constrained variable may contain one or more values, but its behaviour is quite different. When a filled variable unifies with an unfilled variable, a dependency is created and, if the variable continues to exist in the result category, it will be filled. In contrast, when a constrained variable unifies with an unfilled variable, no dependency is created, and any resulting variable becomes a constrained variable, copying the constraint higher in the tree.

Secondly, a constrained variable may be unified with a filled variable, whereas the unification of two filled variables cannot succeed[1]. When a constrained variable and a filled variable attempt to unify, the values stored in the constrained variable are examined to check that they form a subset of the values in the filled variable. If so, the unification may proceed, and dependencies form as if the constrained variable were unfilled. Any resulting variable becomes a filled variable and is not marked as a constraint. If the values of the constrained variable do not form a subset of the filled variable, unification is prevented and one of the parser's options is rejected.

As an example, consider sentence (16), which has the CCG analysis in Figure 7.1. As mentioned above, in order to force the dependency

$$\langle \textit{work-force},\ N/N_1,\ 1,\ \textit{cuts},\ - \rangle$$

we set the value of the variable Y on *work-force* to *cuts* and mark Y as constrained. When *work-force* combines with *cuts* through forward application, the constrained variable Y (constraint *cuts*) with the filled variable X (value *cuts*) on *cuts*. Since the constraint is a subset of the filled variable, unification can proceed, and the variable Y on the *work-force* category N/N becomes a filled (not constrained)

[1] This restriction comes about because the unification of two filled variables should never be attempted and doing so may indicate some error in the parser. It is reasonable that in fact two filled variables should be allowed to unify if their values are identical, although it is not clear in what circumstances this would be useful or how it should be interpreted in terms of dependencies.

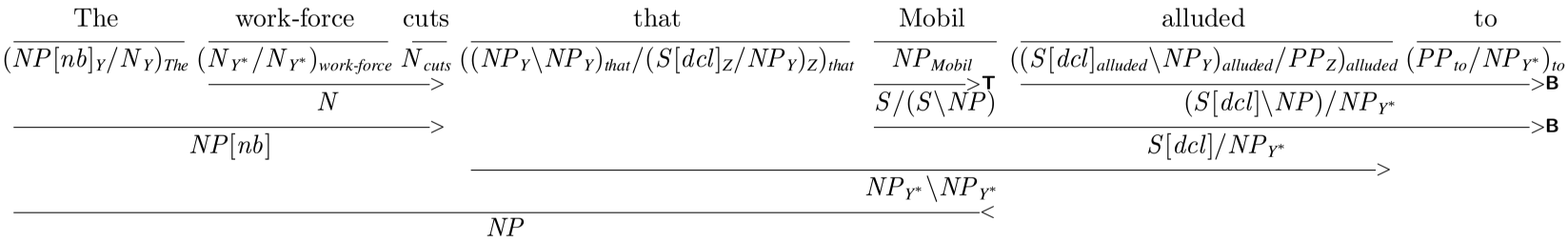


FIGURE 7.1: CCG derivation of sentence (16). Asterisks on variables indicate they are constrained. Note that not all variables are included in this diagram.

variable with the value *cuts*. This variable survives as the head of the result constituent; that is, the forward application creates a constituent *work-force cuts* with head Y (=*cuts*).

Secondly, suppose we wish to force the dependency

$$\langle to,\ PP/NP_1,\ 1,\ cuts,\ -\rangle$$

In (16), the dependency cannot be a local dependency due to the use of a relative clause. However, this does not affect our constraint method; all we need to know is the two words to relate, not specifically how.

For this example, *to* has the category $(PP_X/NP_{Y,1})_X$, so we set the variable Y to *cuts* and mark it as a constraint. In Figure 7.1, *to* first combines with *alluded* and then *Mobil* through forward composition, and the constrained variable is left untouched. This is indicated in Figure 7.1 by repeating the variable Y, marked with an asterisk, on the constituents thus formed. When the constituent *Mobil alluded to* is combined with *that*, the constrained variable originally from *to* is unified with the variable Y on *that*. Since Y on *that* is unfilled, the variable produced by the unification remains constrained. Finally, *that Mobil alluded to* combines with *The work-force cuts*, providing a value to fill the constrained variable. Once again, the value on the constrained variable, *cuts* is a subset of the values on filled variable it unifies with (also *cuts*), so the unification can proceed.

This method of constraining the parser applies to the standard CCG binary combinatory rules, building on the unification mechanism already in place. However, the parser also contains a number of other rules, including unary type-raising and type-changing rules, coordination, and punctuation rules, which do not operate in the same ways, and as a result, this implementation cannot enforce constraints in these cases. Two important illustrations of this are the unary type-changing rules and coordination.

As described in Section 3.2, unary type-changing rules were introduced in CCGbank to avoid the proliferation of possible CCG categories. In the example given there, a constituent with category $S\backslash NP$ is changed to a constituent with category $NP\backslash NP$. These rules are irregular and the relationship between the variables on each side would need to be determined manually for each rule. In this case, the variables would be

$$(S_X\backslash NP_Y)_X \Rightarrow (NP_Y\backslash NP_Y)_X$$

where the X and Y variables are the same on both sides of the rule. That is, the preceding NP that is modified by the constituent with category $S\backslash NP$ (a verb phrase) is also the subject of that verb phrase.

We do not determine the possible mappings for these rules, and as a result we cannot transfer a constraint from one side of a unary rule to the other.

Secondly, the status of coordination in CCG is quite different from other rules, as described in Chapter 3. Words such as *and* are not given an ordinary category with variables, but are instead simply given the label $conj$. Since $conj$ currently contains no variable markup, there are no variables for us to set as constraints.

The method for constraining the parser that we have just outlined is similar to the extension by Djordjevic *et al*. (2007) described in Section 3.5 in that we are reducing the parser's options in the parse chart. Unlike Djordjevic *et al*. (2007), however, we do not target the constituent structure of the sentence; rather, we manipulate the dependency structure, which does not necessarily entail constraints on which sequences of words form constituents. We therefore only operate on cells in the lowest row in the chart, corresponding to individual words, and not on cells corresponding to larger constituents.

Our approach is quite unlike the extension explored by Clark and Curran (2006) (Section 3.5) because that work operates on the training phase of the parser, while our approach targets the parsing phase. Here, we constrain the parser to find, when parsing, one analysis of a sentence consistent with some dependencies, and this analysis is later used to train the parser. An alternative method of including dependency constraints, closer to the method used by Clark and Curran (2006), would be to pass all possible analyses of the sentence consistent with the dependency constraints into the training phase, instead of just one such analysis, and train the parser using the dependencies that appear in $k\%$ of the possible analyses.

7.2 Applying constraint chains

The previous section described how an individual CCG dependency is applied as a constraint to the parser's analysis of a single sentence. The constraints extracted in the previous chapter, however, are sets of several dependencies expressed under the RASP grammatical relation scheme. In Section 3.4 we mentioned that RASP grammatical relations capture generalisations across sentences better than the CCG dependencies; it is for this reason that we use grammatical relations for constraints rather than CCG dependencies. However, this means that we must first translate GR constraints into CCG dependency constraints. We now describe this process.

Firstly, the constraint chains may include words other than the keywords. Although the keywords will only be expressed once in each sentence (we eliminated others in the sentence collection phase), the other words may be repeated. To avoid having to disambiguate between instances of the words, we simply throw out any sentences that do not contain exactly one copy of each constraint chain word. For example, the long Mozart sentence we first introduced in Chapter 1, sentence (17), contains two copies of the word *in* which appears in the constraint chain identified for that fact. This sentence is therefore among those discarded at this point.

(17) Wolfgang Amadeus Mozart (baptized Johannes Chrysostomus Wolfgangus Theophilus) was born in Salzburg in 1756, the second survivor out of six children.

If the sentence contains each constraint word exactly once, we proceed to apply each grammatical relation in the constraint chain. To illustrate the constraining process, we will use sentence (18) with the constraints in (19) as a running example.

(18) Mozart was born in 1756.

(19) a. `(ncsubj born Mozart)`

b. `(ncmod born in)`

c. `(dobj in 1756)`

The grammatical relation `(ncsubj H D)` corresponds to a number of different CCG dependencies, including

$$\langle \texttt{H},\ (S[dcl]\backslash NP_{Y,1})/NP_{Z,2},\ 1,\ \texttt{D},\ -\rangle$$

$$\langle \texttt{H},\ (S[dcl]\backslash NP_{Y,1})/(S[b]_{Z,2}\backslash NP)),\ 1,\ \texttt{D},\ -\rangle$$

$$\langle \texttt{H},\ S[pss]\backslash NP_{Y,1},\ 1,\ \texttt{D},\ -\rangle$$

$$\langle \texttt{H},\ ((S[dcl]\backslash NP_{Y,1})/NP_{Z,2})/PP_{W,3},\ 1,\ \texttt{D},\ -\rangle$$

and so on. In these cases, the categories appear on the word `H`. In order to place the `ncsubj` constraint, we examine the categories that the supertagger assigns to the word *born*. In this case, the supertagger assigns only the category $S[pss]\backslash NP$. Since this category appears in our list of possible CCG dependencies, we place the constraint on the variable indicated, Y.

Although this is not the case in this sentence, it is possible that when supertagging the sentence, the parser assigns some category to *born* that is not in our list of possible `ncsubj` categories. In this event, we delete this extra category from the chart, thus preventing the parser from using it in the analysis and avoiding the constraint.

We repeat this process for the remaining constraints. Since we are enforcing multiple GR constraints simultaneously, the same word may have multiple variables filled in. In this example, the word *in* has the category $((S\backslash NP)\backslash(S\backslash NP))/NP$. Since the dependency $\langle$H, $((S\backslash NP)\backslash(S_{Y,1}\backslash NP))/NP_{Z,2}$, 1, D, $-\rangle$ is one option for the `ncmod` constraint (19b) and $\langle$H, $((S\backslash NP)\backslash(S_{Y,1}\backslash NP))/NP_{Z,2}$, 2, D, $-\rangle$ is one option for the `dobj` constraint (19c), our constraining process fills in both variables on *in*, resulting in $((S\backslash NP)\backslash(S_{born,1}\backslash NP))/NP_{1756,2}$.

As described in Sections 3.1 and 7.1, variables may contain multiple values for coordination cases. For example, sentence (20) with constraints (21) would require the variable Z on the word *sell* with the category $(S[dcl]\backslash NP_{Y,1})/NP_{Z,2}$ to be filled with both *stocks* and *bonds*, as in $(S[dcl]\backslash NP_{Y,1})/NP_{\{stocks,bonds\},2}$.

(20) They sell stocks and bonds.

(21) a. `(dobj sell stocks)`

b. `(dobj sell bonds)`

Further, some categories may license two dependencies with the same grammatical relation. For example, *than* in *less than six* may take the category $(NP/NP_{Y,1})\backslash(S[adj]_{Z,2}\backslash NP)$ and create two `ncmod` GRs, listed in (22) and (23).

(22) `(ncmod less than)`

(23) `(ncmod six than)`

Since either of these two GRs may be offered as a constraint and there is no way to distinguish between them, we must allow either possibility. Consider the case where the constraint to be applied is (22), and the only category on *than* is $(NP/NP)\backslash(S[adj]\backslash NP)$. We enforce this constraint by duplicating the category and filling in each possible variable once. That is, the parser ends up with two choices for *than*, $(NP/NP_{less})\backslash(S[adj]\backslash NP)$ and $(NP/NP)\backslash(S[adj]_{less}\backslash NP)$.

GR	First	Second	Total
xcomp	163	34	197
xmod	1	10	11
ncsubj	201	1	202
xsubj	4	0	4
ncmod	15	276	291
cmod	11	21	32
det	0	16	16
dobj	163	1	164
obj2	9	0	9
iobj	81	3	84
csubj	3	0	3
aux	1	28	29
ccomp	87	37	124

TABLE 7.1: Number of times a grammatical relation (GR word1 word2) corresponds to a CCG category on the first and second words in the relation.

It may also be the case that both (22) and (23) are provided as constraints. In this event, we use duplication of the category to offer all four of the following possibilities:

$$(NP/NP_{less,six})\backslash(S[adj]\backslash NP)$$

$$(NP/NP_{less})\backslash(S[adj]_{six}\backslash NP)$$

$$(NP/NP_{six})\backslash(S[adj]_{less}\backslash NP)$$

$$(NP/NP)\backslash(S[adj]_{less,six}\backslash NP)$$

(The third option is in fact the correct one in this case.)

Although it may seem unusual to allow both *less* and *six* to fill the same variable, this must be allowed for the case where the constraints capture coordination as well, for example *less than six and eight respectively* with the constraints (ncmod six than) and (ncmod eight than).

From these examples it should be clear that there is no direct relationship between which word bears the CCG dependency and which word is listed first in the grammatical relation constraint. For example, the ncsubj constraints were translated to CCG dependencies on their first word, while ncmod constraints were placed on the second word. In fact, most GRs correspond to some CCG dependencies on the first word and some on the second. Table 7.1 lists the 13 relations from the RASP GR scheme that are generated by the mapping from CCG dependencies and how often each corresponds to a dependency on its first word and how often on its second word.

Given this, it is possible to imagine a case where a grammatical relation `(GR word1 word2)` may be expressed with the two dependencies

$$\langle \texttt{word1},\ X/Y_1,\ 1,\ \texttt{word2},\ - \rangle$$

$$\langle \texttt{word2},\ W\backslash Z_1,\ 1,\ \texttt{word1},\ - \rangle$$

and the parser, given a sentence containing `word1` and `word2`, assigns to `word1` the possible categories X/Y and Z and to `word2` the possible categories Y and $W\backslash Z$. If we wish to allow both options for this grammatical relation simultaneously, we must allow Z and Y to remain as possible categories on `word1` and `word2` respectively. However, this opens the possibility that the parser may choose the categories Z and Y and avoid the constraint entirely.

Therefore, we cannot allow both options simultaneously; rather, we must try each option in turn. That is, we delete Z from `word1` and set the variable on X/Y, parse the sentence and calculate the probability of the analysis produced. Then, we reload all the categories, delete Y from `word2`, set the variable on $W\backslash Z$, and parse the sentence a second time. We then choose which of these two parses has the highest probability, and return that as the final analysis.

Now, since we are applying several GR constraints and each may allow either word as head, we must parse each sentence several times, corresponding to each combination of word choices. For n constraints, this may lead to 2^n parses of the sentence in the worst case, although many of these will be impossible for any given sentence.

7.3 Multiple constraint chains

As described in Section 6.3, for some facts we return multiple possible chains of grammatical relations, for example the fact *OEM stands for Original Equipment Manufacturer* has two possible constraint chains, listed in (24) and (25).

(24)
```
(ncmod Manufacturer Original)
(ncmod Manufacturer Equipment)
(xcomp is Manufacturer)
(ncsubj is OEM)
```

(25) `(ncmod Manufacturer Original)`
`(ncmod Manufacturer Equipment)`
`(dobj for Manufacturer)`
`(ncmod stands for)`
`(ncsubj stands OEM)`

We handle these options using the same iteration technique described above. That is, we parse the sentence repeatedly, enforcing each set of constraints in turn, and keep the analysis with the highest probability from all of the analyses proposed.

In Section 7.2, we mentioned that if a sentence contains multiple copies of a word that appears in the constraint chain, then we do not attempt to enforce the constraint chain for that sentence. This does not affect the application of other constraint chains in the case where there are multiple such chains. It is possible that we could overcome our requirement for uniqueness in keywords and constraint words by the same iterative technique, although this would once again increase the number of times each sentence is parsed.

7.4 The annotated corpus

Chapter 5 described the collection of a corpus of sentences from the Web, and Chapter 6 described the extraction of chains of grammatical relations that describe how the keywords of a fact should be related. The previous sections in this chapter described how a chain of grammatical relations may be applied to a sentence as constraints. All that remains is to describe how these components combine to produce an annotated corpus.

Consider one particular fact; say *Mozart was born in 1756*. We take the 161 unique sentences collected for this fact and the constraint chain(s) found in Chapter 6. If any of these constraint chains contain the `conj` grammatical relation, we skip them, since as described in Section 7.1 our constraining method cannot enforce such constraints.

Next, we parse each sentence in this set several times to try all combinations within the constraint chain (or chains, if there are more than one) as described above. Each time the parser finds an analysis, we check to ensure that the required GRs have in fact been generated, to guard against the case where unary rules have interfered. If the analysis is valid and consistent with the constraints, we assume that it is

correct[2], and check the probability assigned to the analysis by the C&C decoder. Of all analyses found this way, we keep that with the highest probability. If no analyses are found, we discard the sentence.

Finally, we use the C&C printer to output the final analysis for the sentence in the CCGbank-style format. The sentences printed out form our new annotated training data.

Table 7.2 summarises the yield of the annotation process for the 24 facts for which constraint chains were automatically identified in Chapter 6. This table lists: the number of constraint chains identified for each fact, not including chains containing the `conj` relation; the number of sentences discarded for not having exactly one copy of every constraint word, in all possible constraint chains; the number of sentences for which an analysis could not be found; and the number of sentences successfully parsed and added to the training corpus.

7.5 Summary

In this chapter, we presented our new method for constraining the syntactic analyses produced by the C&C parser. We then used this method to apply the chains of grammatical relations produced in the previous chapter as constraints, by converting them to CCG dependencies. By applying these constraints to all sentences collected in Chapter 5, we have automatically created an additional 728 sentences of annotated training data for the C&C parser. Although many sentences and some facts are lost along the way, the yield of the process is high, showing that it is feasible to create a large corpus of annotated training data in this manner.

[2]That is, we assume that if the given grammatical relations are not appropriate for this sentence, then the parser will not find an analysis.

Keywords	# Chains	# Thrown out	# Failed	# Parsed
American Association of Retired Persons AARP	1	43	27	95
CNN Cable News Network	1	41	13	8
Original Equipment Manufacturer OEM	2	91	112	11
Galileo astronomer	1	193	26	24
Jennifer Capriati tennis player	2	0	5	8
Mark Twain Samuel Langhorne Clemens	1	286	2	3
Nirvana lead singer Kurt Cobain	2	0	73	49
Pratchett author	2	0	38	122
Canberra capital Australia	2	44	27	7
Olympus Mons Mars	2	98	43	41
capital India New Delhi	1	72	18	4
Horus Egyptian god	2	0	61	29
Horus god sky	1	80	11	4
Saturn sixth planet Sun	1	11	7	12
Tale of Genji Murasaki	1	124	4	4
cataract clouding lens eye	1	124	88	1
hydrogen lightest element	1	51	30	4
nematode roundworm	1	77	33	33
Columbus discovered America	1	0	125	174
Edmund Hillary climbed Everest	1	0	24	5
James Dean died car crash	1	11	4	18
Marie Curie discovered radium	2	0	33	40
Martin Luther King Jr assassinated 1968	1	0	4	2
Mozart born 1756	1	93	38	30
Total		1439	846	**728**

TABLE 7.2: The loss of sentences in the annotation process.

CHAPTER 8

Corpus Evaluation

In Chapter 4, we discussed the major drawback of supervised parsing for English, specifically its dependence on an annotated corpus which is limited in size and variety and would be expensive to extend manually. We then introduced the notion of semi-supervised learning, and how we can use a semi-supervised approach to extend the corpus automatically and address this annotation bottleneck. Chapters 5, 6 and 7 described our method for collecting and annotating sentences from the Web, creating a corpus of 728 automatically-annotated sentences. In this chapter, we evaluate how useful this data is as training data for improving parser performance.

8.1 C&C standard evaluation

In order to evaluate the effectiveness of the additional 728 sentences, we train two models for the C&C parser. The *baseline system* is trained using only the parser's standard training data, the 39604 sentences from sections 02–21 of CCGbank. The second model, which we refer to as the *new system*, is trained using sections 02–21 of CCGbank plus the additional 728 sentences. We use the parser's normal-form model, as described in Section 3.3.2.

We then evaluate the two models on CCGbank section 23 using the C&C parser's standard evaluation method, described in Section 3.4. We calculate precision, recall and F-score over dependencies in section 23. We calculate both labelled and unlabelled scores; in the former, the dependency must be entirely correct, while in the latter, a dependency is considered correct if the two words involved are correct, even if the category or slot is incorrect. We also calculate category accuracy (proportion of words assigned the correct CCG category), sentence accuracy (proportion of sentences with all dependencies correct), and coverage (proportion of sentences that can be parsed). Following Clark and Curran (2007), gold standard part-of-speech (POS) tags are used for most of these calculations; the figure for LF (POS) gives

Model	LP	LR	LF	LF (POS)	SENT ACC	UP	UR	UF	CAT ACC	COV
Baseline	85.53	84.71	85.12	83.38	32.14	92.37	91.49	91.93	93.05	99.06
New	85.64	84.77	85.21	83.54	32.03	92.41	91.47	91.94	93.08	99.06

TABLE 8.1: Performance of the baseline and new parsing models on CCGbank section 23.

the F-score achieved using automatically-assigned POS tags, as a better indication of real-world parser performance. The results of this evaluation are given in Table 8.1.

From these figures, it appears that the additional training data has little effect. The numbers are all close; some have increased and others decreased, showing that the new parsing model correctly determines some dependencies that the baseline does not, and vice versa.

There are a number of factors that may contribute to this result, including the amount of data added, genre effects, and noise. Firstly, CCGbank sections 02–21 contain 39604 sentences. We are adding 728 additional sentences, a small 1.8% increase. Secondly, both the main training data and the test set are taken from CCGbank, a corpus of newswire text, while the additional data consists of more varied Web text. In Section 4.1, we discussed findings by Gildea (2001) that suggested that additional training data from a domain other than the test set does not improve performance in the evaluation, and yet this is exactly what we have done with this evaluation. Finally, it is also possible that the additional data we are adding is noisy, due to the automatic method of collection and annotation, and that this noise counteracts the benefits of the increased corpus size. From this evaluation alone, we cannot draw any conclusions about which factor or factors are predominantly responsible for the small effect on performance, and further evaluations will be necessary.

8.2 Alternative evaluations

To further evaluate the performance of the two C&C parsing models, it will be interesting to perform two additional evaluations, against the Briscoe and Carroll (2006) reannotation of DepBank (see Section 2.3.2), and against a new corpus of Web text.

Since DepBank, like our CCGbank test set, is descended from the Penn Treebank section 23, results of the two parsing models on DepBank should be very similar to those reported in Section 8.1. However, there is a slim possibility that the use of the RASP GR scheme in the creation of our additional training data may improve the performance of the C&C parser against a test corpus using that scheme.

Fact	Keywords
LASER stands for Light Amplification by Stimulated Emission of Radiation	LASER Light Amplification by Stimulated Emission of Radiation
Marilyn Monroe was an actress	Marilyn Monroe actress
Paris is the capital of France	capital Paris France
Alexander Graham Bell invented the telephone	Alexander Graham Bell invented telephone
Beethoven died in 1827	Beethoven died 1827
Captain Cook landed at Botany Bay	Cook landed Botany Bay
Jack Ruby shot Lee Harvey Oswald	Ruby shot Oswald
James Chadwick discovered the neutron	James Chadwick discovered neutron
Elvis was born in 1935	Elvis born 1935
John F. Kennedy was assassinated in 1963	Kennedy assassinated 1963
Richard II was deposed in 1399	Richard II deposed 1399

TABLE 8.2: The list of facts used to collect a second corpus of Web sentences, for a future Web text evaluation of the parsing models.

That is, the parser may be returning incorrect CCG dependencies which nevertheless correspond to the correct grammatical relations. This situation is unlikely, but an evaluation against DepBank will test the possibility.

Secondly, one of the factors identified in Section 8.1 as a possible influence on the new parsing model's performance was that the extra training data is Web text, while the test set remains newswire text. Although performance has not increased on newswire text, it is possible that the new parsing model improves on the baseline when parsing Web text. To test this possibility, it would be advisable to perform an evaluation of the two models on a corpus of Web text, collected in the same manner as the additional training data.

We have commenced the creation of the Web test corpus necessary for this evaluation. We have identified a number of new facts of similar kinds to the facts used to collect the training corpus, listed in Table 8.2. Using these facts and the procedure outlined in Chapter 5, we have collected a corpus of 1113 sentences. All that remains is to manually annotate all or a sample of these sentences to form a gold standard against which to evaluate.

8.3 Summary

We have trained a new model for the C&C parser using the annotated training data collected by the procedure outlined in the previous chapters. We find that the performance of this system is close to the

performance of the baseline system, where the parser is trained using only its standard training data. These results show that it is feasible to create an automatically-annotated training corpus to supplement existing data and still maintain high parser performance. There are number of factors that may be responsible for a greater improvement in performance not being observed, including the amount of data collected and the gold standard corpus that we use for evaluation. To address these factors, we need to collect a much greater additional training corpus, and perform an extra evaluation on a corpus of Web text.

CHAPTER 9

Conclusion

In the preceding chapters, we have described a novel method for automatically creating an annotated corpus of Web text, and evaluated its effectiveness as training data for a state-of-the-art parser. In this chapter, we examine some future directions in which our work could be extended, and the conclusions that we can draw about the technique we have proposed.

9.1 Future Directions

As discussed in Chapter 8, the additional training data that we collected and annotated has not led to a large increase in parser performance. One possible factor in this is a mismatch between the method for collecting data and the method of evaluation. To address this, it is important to evaluate the new system against a corpus of Web text, in addition to the newswire text in the standard evaluation. We have collected sentences that can constitute this evaluation corpus; all that remains is to create the gold-standard annotations to enable the evaluation to proceed.

A second factor that we identified is that the additional training data constitutes only a small increase in the amount of training data available. It is therefore crucial to increase the scale of the data collection to test the limits of the technique. Because only the initial step of our algorithm is performed manually, this large-scale data collection is feasible. Once this large corpus has been collected, comparative studies could investigate the effects on parser performance of the size and composition of the training corpus.

Since the only manual step in our process is the creation of facts and selection of fact keywords, we intend to investigate how this step could likewise be automated. This would involve extracting facts of particular fact types (e.g. dates of birth) from structured or semi-structured resources such as Wikipedia.

Other directions in which this work could be developed involve extensions to the implementation. For example, our sentence collection process uses heuristics to consider HTML markup in sentence identification, and uses an aggressive pruning strategy to avoid noise. The amount of text and the number of facts available to us on the Web mean that losing sentences in this way is an acceptable loss, but this does not mean that the techniques used could not be improved.

Future work could also extend the method of constraining the C&C parser that we have described. Currently, we can enforce constraints using the standard binary rules in CCG; additional unary rules introduced by CCGbank and complications with the coordination rule mean that the implementation of the constraint technique does not cover all circumstances. In addition, it would be interesting to investigate a method of weighting constraints in order to reflect probability of constraint correctness.

The method we have presented for extracting constraints from simple sentences has proven extremely promising. One result in particular motivates the future inclusion of an additional criterion for simple sentences, namely that they do not involve coordination structures. An extension of this work could investigate the effect of different criteria for simple sentences.

9.2 Conclusion

Our contribution is the development of an entirely automatic procedure for the creation of annotated training data for statistical parsing, using manually identified fact keywords and the text available on the Web. We have introduced a new method for constraining the syntactic analyses produced by the C&C parser to ensure that particular grammatical relations are expressed in the output. In addition, we introduce the notion of using the C&C parser to automatically extract grammatical relations from simple sentences for use as constraints.

We have demonstrated that it is feasible to use a fully automated method for creating annotated training data, thus addressing the annotation bottleneck that plagues supervised parsing. This data will enable us to train robust and accurate parsers for a range of natural language processing applications.

References

Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting of the Assocation for Computational Linguistics*, pages 26–33.

Yehoshua Bar-Hillel. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Douglas Biber. 1993. Using register-diversified corpora for general language studies. *Computational Linguistics*, 19(2):219–241, June.

Rens Bod. 2006. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 865–872.

Johan Bos, James R. Curran, and Edoardo Guzetti. 2007. The Pronto QA system at TREC-2007. In *Proceedings of the Sixteenth Text REtrieval Conference*.

Thorsten Brants and Alex Franz. 2006. Web 1t 5-gram, version 1. LDC2006T13, Linguistic Data Consortium, Philadelphia.

Eric Brill, Jimmy Lin, Michele Banko, Susan Dumais, and Andrew Y. Ng. 2001. Data-intensive question answering. In *Proceedings of the Tenth Text REtrieval Conference*, pages 393–400, November.

Ted Briscoe and John Carroll. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the Poster Session of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics*, pages 41–48, July.

Ted Briscoe, John Carroll, Jonathan Graham, and Ann Copestake. 2002. Relational evaluation schemes. In *'Beyond PARSEVAL' workshop at the Third International Conference on Language Resources and Evaluation*, pages 4–8.

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, July.

Ted Briscoe. 2006. An introduction to tag sequence grammars and the RASP system parser. Technical Report 662, University of Cambridge, March.

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, March.

John Carroll, Anette Frank, Dekang Lin, Detleft Prescher, and Hans Uszokoreit, editors. 2002. *Proceedings of the Workshop 'Beyond PARSEVAL—Towards improved evaluation measures for parsing systems' at the 3rd International Conference on Language Resources and Evaluation*.

Eugene Charniak. 2000. A maxmium entropy inspired parser. In *Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING-04*, pages 282–288.

Stephen Clark and James R. Curran. 2006. Partial training for a lexicalized-grammar parser. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 144–151, June.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552, December.

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

Michael Collins. 2003. Head-driven statistical models for natural language processing. *Computational Linguistics*, 29(4):589–637, December.

Bojan Djordjevic, James R. Curran, and Stephen Clark. 2007. Improving the efficiency of a wide-coverage CCG parser. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 39–47, Prague, Czech Republic, June.

Jason Eisner. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86.

Daniel Gildea. 2001. Corpus variation and parser performance. In Lillian Lee and Donna Harman, editors, *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202.

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

Eduard Hovy, Ulf Hermjakob, and Deepak Ravichandran. 2002. A question/answer typology with surface text patterns. In *Proceedings of the DARPA Human Language Technology conference*, pages 247–251.

Nancy Ide and Jean Véronis. 1998. Introduction to the special issue on word sense disambiguation: The state of the art. *Computational Linguistics*, 24(1):2–40.

Frank Keller and Mirella Lapata. 2003. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484.

Adam Kilgarriff and Gregory Grefenstette. 2003. Introduction to the special issue on the web as corpus. *Computational Linguistics*, 29(3):333–347.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *Proceedings of the EACL03: 4th International Workshop on Linguistically Interpreted Corpora*, pages 1–8.

Tibor Kiss and Jan Strunk. 2006. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

Mirella Lapata and Frank Keller. 2005. Web-based models for natural language processing. *ACM Transactions on Speech and Language Processing*, 2(1):3.

Dekang Lin. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering*, 4(2):97–114.

Vinci Liu and James R. Curran. 2006. Web text corpus for natural language processing. In *Proceedings of the 11th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 233–240.

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.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119.

Tom Mitchell. 1999. The role of unlabeled data in supervised learning. In *Proceedings of the Sixth International Colloquium on Cognitive Science*. (invited paper).

Stephan Oepen, Ezra Callahan, Dan Flickinger, Christopher D. Manning, and Kristina Toutanova. 2002. LinGO Redwoods: A rich and dynamic treebank for HPSG. In *'Beyond PARSEVAL' workshop at the Third International Conference on Language Resources and Evaluation*, pages 17–22.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional gramar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 271–278.

Brian Roark. 2002. Evaluating parser accuracy using edit distance. In *'Beyond PARSEVAL' workshop at the Third International Conference on Language Resources and Evaluation*, pages 30–36.

Geoffrey Sampson and Anna Babarczy. 2002. A test of the leaf-ancestor metric for parse accuracy. In *'Beyond PARSEVAL' workshop at the Third International Conference on Language Resources and Evaluation*, pages 23–29.

Yoav Seginer. 2007. Fast unsupervised incremental parsing. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 384–391, Prague, Czech Republic, June.

Satoshi Sekine. 1997. The domain dependence of parsing. In *Proceedings of the fifth conference on Applied Natural Language Processing*, pages 96–102.

Mark Steedman. 2000. *The Syntactic Process*. Language, Speech, and Communication series. The MIT Press, Cambridge, MA.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. The Penn Treebank: An Overview. In Anne Abeillé (Ed.), Treebanks: Building and Using Parsed Corpora.

Daniel Tse and James R. Curran. 2007. Extending CCGbank with quotes and multi-modal CCG. In *Proceedings of the Australasian Language Technology Workshop (ALTW)*.

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)*.

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

Ellen M. Voorhees. 2001. Overview of the TREC 2001 Question Answering track. In *Proceedings of the Tenth Text REtrieval Conference*, pages 42–51, November.

Ellen M. Voorhees. 2004. Overview of the TREC 2004 Question Answering track. In *Proceedings of the Thirteenth Text REtrieval Conference*.

David Yarowsky. 1993. One sense per collocation. In *Proceedings of the workshop on Human Language Technology*, pages 266–271.