May 2005

CCGbank: User's Manual

Julia Hockenmaier
University of Pennsylvania

Mark Steedman
University of Edinburgh

Recommended Citation

Julia Hockenmaier and Mark Steedman, "CCGbank: User's Manual", . May 2005.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-09.

CCGbank: User's Manual

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-09.

CCGbank: User's Manual

Julia Hockenmaier and Mark Steedman
juliahr@cis.upenn.edu, steedman@inf.ed.ac.uk

http://www.inf.ed.ac.uk/groups/ccg

Technical Report MS-CIS-05-09
Department of Computer and Information Science
University of Pennsylvania, Philadelphia

May 2005

Contents

Acknowledgements

This research has benefited enormously from many discussions with our colleagues at Edinburgh, the University of Pennsylvania and elsewhere.

In Edinburgh, Chris Brew's and also Henry Thompson's advice was instrumental during the early stages of this project. Jason Baldridge, Gann Bierner, Johan Bos, Stephen Clark and James Curran have also provided invaluable feedback. At Penn, we would like to thank in particular Aravind Joshi, Mitch Marcus, Martha Palmer, Ann Bies, Dan Bikel, David Chiang, Dan Gildea, Alexandra Kinyon, Seth Kulick, Rashmi Prasad, Carlos Prolo and Libin Shen. We would also like to thank Ted Briscoe, Ewan Klein, Mark Johnson and Josef van Genabith. We are very grateful to Doug Rohde for adapting his tgrep2 tool to CCGbank.

Most of this research was done at the University of Edinburgh, where it was supported by EPSRC grant GR/M96889, an EPSRC studentship, ICCS/HCRC and the Edinburgh Language Technology Group. Julia Hockenmaier is currently at the University of Pennsylvania, where she is supported by NSF ITR grant 0205456.

Chapter 1

Introduction

CCGbank is a version of the Penn Wall Street Journal Treebank (Marcus *et al.*, 1993) in which the standard trees have been mapped into ”normal form” derivations in Combinatory Categorial Grammar (CCG, Steedman (1996, 2000)). The raison d’etre for developing CCGbank was as a resource for extracting CCG grammars (and in particular a wide-coverage CCG lexicon) and statistical head-dependency models from the WSJ treebank for use in a series of wide-coverage CCG-based parsers (Hockenmaier (2001, 2003a,b), Hockenmaier and Steedman (2002b), Clark *et al.* (2002), Clark and Curran (2003, 2004)), using the method pioneered by Collins (1997, 1999), Goodman (1997, 1998) and others for context-free phrase structure grammars with this more expressive grammar formalism. The point of this exercise was to deliver more accurate-wide coverage parsers capable of building interpretable structure (which standard context-free Treebank grammars do not in general do), which could then be used in applications such as question answering or summarization (Clark *et al.* (2004), Bos *et al.* (2004)). CCG categories such as (S\NP)/NP, the lexical category of the transitive verb, map in a regular way to the lexical category types of other lexicalized grammar formalisms, such as the elementary trees of Lexicalized Tree-Adjoining Grammar (TAG, Joshi and Schabes (1992))the signs of Head-driven Phrase Structure Grammar (HPSG, Pollard and Sag (1994)), and Lexical-Functional Grammar (LFG, Bresnan (1982)), among many others. Therefore CCGbank might be more readily translatable into these other frameworks for use in grammar extraction and parser development than the original treebank. In particular, in order to make the extraction of a CCG grammar and lexicon possible, it was necessary to resolve a number of inconsistencies and minor errors in the original treebank. Thus, CCGbank is a somewhat improved resource in linguistic terms. One of the purposes of the present document is also to systematically identify these problems in the original treebank so that they can be dealt with in any future issue.

A (rightmost) normal form derivation in CCG (Hepple and Morrill (1989), Wittenburg (1986), Wittenburg and Wall (1991)) can informally be thought of as one in which the combinatory rules of composition and type-raising are only used when the reading that results is not obtainable without them. For many sentences, such normal form derivations are structurally isomorphic to standard Treebank trees, except that the 48 non-terminal labels of the Penn POS tag set are replaced by a much larger and more informative set of around 1300 CCG category types. However, some sentences, notably those involving topicalization, relativization, and various types of reduction under coordination—in short, those for which the involvement of combinatory rules is crucial—have non-standard trees, sometimes radically non-standard, since CCG operates without the classical transformational operations of movement and deletion, which are implicit in the original treebank annotations. Some of the differences between CCGbank and the original are more willful, and arise from the lexicalist perspective. Thus the ”small clause” analysis of the complements in ditransitives, object control, and ”exceptional case marking” have been replaced by more surfacey, ”object XP” analyses, as they have in other lexicalist theories such as HPSG. The reason for doing this is that the supposed subject in these constructions can extract, a fact that makes them unlike true subjects, and leads to an increase in the size of the lexicon if small clauses are assumed.

Other details of the CCG analyses are forced by decisions in the original treebank annotation schema which we were unable to overcome by automatic or semiautomatic means. Of these the one which we regret most is that postnominal modifiers such as (restrictive) relative clauses have to be analyzed as NP modifiers, rather than N or $\overline{\mathrm{N}}$ modifiers. This is one of the few places where a category's syntactic type clashes with its semantic type, and endangers the otherwise beautifully simple ”surface compositional” relation between CCG derivation and the building of interpretable structure. We are painfully aware of the difficulties that this will cause users with an interest in interpretation, not least ourselves.

We try to comment on these details as we describe CCGbank and its uses. The report proceeds as follows: First, we briefly outline the nature and assumptions of CCG. Next we discuss in detail the process of transforming the original treebank, providing an algorithm inclusing a number of preprocessing and postprocessing steps. Appendix A describes the procedure for identifying heads, complements and adjuncts which is crucial to this process. Appendix B lists an number of systematic changes to the original Treebank that were made and may be of general interest to anyone using any version of the Penn Treebank, particularly the original Treebank II, to whose originators and annotators we remain eternally indebted. Appendix D describes the file format of this release of CCGbank.

Previous versions of CCGbank are described in Hockenmaier (2003a) and also in Hockenmaier and Steedman (2002a). Hockenmaier *et al.* (2004) describes a precursor to the translation algorithm presented here which only aims to extract a CCG lexicon from the Penn Treebank, not to create a corpus of CCG derivations. This manual describes the version of CCGbank that is released by the Linguistic Data Consortium (LDC) under catalog number LDC2005T13 (ISBN 1-58563-340-2).

Chapter 2

Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG, Steedman (1996, 2000)), is a grammatical theory which provides a completely transparent interface between surface syntax and underlying semantics. Each (complete or partial) syntactic derivation corresponds directly to an interpretable structure. This allows CCG to provide an account for the incremental nature of human language processing. The syntactic rules of CCG are based on the categorial calculus of Ajdukiewicz (1935) and Bar-Hillel (1953) as well as on the combinatory logic of Curry and Feys (1958). The main attraction of using CCG for parsing is that it facilitates the recovery of the non-local dependencies involved in constructions such as extraction, coordination, control and raising.

This chapter provides a brief introduction to CCG. We focus in particular on the definition of categories and the combinatory rules, the role of derivations within CCG, the problem of so-called "spurious ambiguity", and on the representation of predicate-argument structure in CCGbank.

The main references for CCG are Steedman (2000, 1996), and the reader is referred to these for further details on other aspects of CCG. Wood (1993) provides a good overview of different variants of categorial grammar.

2.1 Categories and the lexicon

In categorial grammar, words are associated with very specific categories which define their syntactic behaviour. A set of universal rules defines how words and other constituents can be combined according to their categories. Variants of categorial grammar differ in the rules they allow, but categories as defined below are the building blocks of any categorial grammar.

In general, the set of syntactic categories $\mathscr{C}$ is defined recursively as follows:

Atomic categories: the grammar for each language is assumed to define a finite set of atomic categories, usually $\mathsf{S}, \mathsf{NP}, \mathsf{N}, \mathsf{PP} ... \in \mathscr{C}$

Complex categories: if $\mathsf{X}, \mathsf{Y} \in \mathscr{C}$, then $\mathsf{X/Y}, \mathsf{X\backslash Y} \in \mathscr{C}$

Complex categories $\mathsf{X/Y}$ or $\mathsf{X\backslash Y}$ are functors with an argument Y and a result X. Here we assume a directional variant of categorial grammar, which differentiates between arguments to the right of the functor (indicated by the forward slash $/$) and arguments to the left of the functor (indicated by the backslash $\backslash$). We follow Steedman's notation to represent complex functor categories. In this notation, the result category always precedes the argument. Hence the category $(\mathsf{S\backslash NP})/\mathsf{NP}$ for transitive verbs in English encodes the information that the verb takes a noun phrase to its right, and another noun phrase to its left to form a sentence. [1]

The lexicon specifies the categories that the words of a language can take. For instance, a categorial lexicon for English might contain the following entries:

(1) *John* $\vdash$ NP
shares $\vdash$ NP
buys $\vdash$ $(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP}$
sleeps $\vdash$ $\mathsf{S}\backslash\mathsf{NP}$
well $\vdash$ $(\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP})$

Here, *John* and *shares* are noun phrases. *Buys* is a transitive verb. *Sleeps* is intransitive, as it only takes one NP argument. *Well* can modify *sleeps*, as it takes an intransitive verb (or a verb phrase) as argument to return a constituent of the same category.

As defined above, the set of categories is infinite. However, it is tacitly assumed that the lexicon of any specific language will only use a finite subset of these categories, and in CCG there is a strong interaction between the lexical categories of a language and the combinatory rules allowed in its grammar.[2]

Each syntactic category also has a semantic interpretation whose type must correspond to that of the syntactic category. If one chooses to represent semantic interpretations in the language of the λ-calculus (as is commonly done), then each (syntactic) argument of a complex functor category has a semantic counterpart in the form of a bound variable in the λ-term. Semantic interpretations can be arbitrarily complex objects (as long as there is a correspondence to the syntactic category). However, logical forms are not represented in CCGbank. Therefore it is sufficient for our purposes to represent semantic interpretations as simple expressions like the following:

(2) *John* $\vdash$ $\mathsf{NP} : john'$
shares $\vdash$ $\mathsf{NP} : shares'$
buys $\vdash$ $(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x.\lambda y.buys'xy$
sleeps $\vdash$ $\mathsf{S}\backslash\mathsf{NP} : \lambda x.sleeps'x$
well $\vdash$ $(\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP}) : \lambda f.\lambda x.well'(fx)$

In CCGbank, a different representation of predicate-argument structure was chosen. This representation is explained in section 2.5.

2.2 AB categorial grammar

The system defined by Ajdukiewicz (1935) and Bar-Hillel (1953) (hence: AB categorial grammar) forms the basis for CCG and all other variants of categorial grammar. In AB categorial grammar, categories can only combine through function application. In directional variants of categorial grammar, there are two versions of function application, respecting the directionality of the slash in the syntactic category. However, their effect on the semantic interpretation is the same:[3]

(3) *Forward Application*:
$\mathsf{X}/\mathsf{Y} : f \quad \mathsf{Y} : a \quad \Rightarrow \quad \mathsf{X} : f(a)$

[1] There is an alternative notation originating in Lambek (1958), which also uses forward slashes and backslashes, but puts the argument "underneath" the backslash. In this notation, a transitive verb becomes $(\mathsf{NP}\backslash\mathsf{S})/\mathsf{NP}$

[2] A recent proposal by Baldridge (2002) presents a refinement of CCG, called Multi-Modal CCG, which assumes a universal set of combinatory rules, with all the language-specific variation being accounted for in the lexicon.

[3] Categorial grammar rule schemata are traditionally presented in a bottom-up fashion. In the remainder of this report, rules that are presented in the form $\alpha \Rightarrow X$ are written in bottom-up manner.

Backward Application:
$\mathsf{Y} : a \quad \mathsf{X\backslash Y} : f \quad \Rightarrow \quad \mathsf{X} : f(a)$

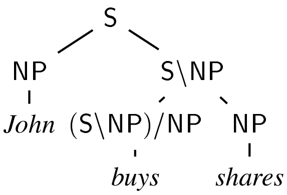
The rule of forward application states that if a constituent with category $\mathsf{X/Y}$ is immediately followed by a constituent with category Y, they can be combined to form a constituent with category X. Analogously, backward application allows a constituent $\mathsf{X\backslash Y}$ that is immediately preceded by a constituent with category Y to combine with this constituent to form a new constituent of category X.

A string α is considered grammatical if each word in the string can be assigned a category (as defined by the lexicon) so that the lexical categories of the words in α can be combined (according to the rules of the grammar) to form a constituent. The process of combining constituents in this manner is called a derivation, although we will also sometimes refer to the syntactic structure constructed by this process as a derivation. Since derivations proceed bottom-up, starting from the words, they are usually represented in the following manner:

(4)
$$\begin{array}{ccc}
\textit{John} & \textit{buys} & \textit{shares}\\
\hline
\mathsf{NP} : \mathit{John}' & (\mathsf{S\backslash NP})/\mathsf{NP} : \lambda x.\lambda y.\mathit{buys}'xy & \mathsf{NP}\mathit{shares}'\\
\cline{2-3}
 & \multicolumn{2}{c}{\mathsf{S\backslash NP} : \lambda y.\mathit{buys}'\mathit{shares}'y} \;> \\
\hline
\multicolumn{3}{c}{\mathsf{S} : \mathit{buys}'\mathit{shares}'\mathit{John}'} \;<
\end{array}$$

But note that the constituent structure induced by this derivation corresponds to a tree:

(5)



In this report, derivations will often be represented as trees, since this representation follows naturally from the translation algorithm. The correspondence to the more traditional notation is assumed to be understood.

2.3 The combinatory rules of CCG

CCG extends AB categorial grammar by a set of rule schemata based on the combinators of combinatory logic (Curry and Feys, 1958). These combinatory rules enable a succint analysis of the long-range dependencies involved in extraction and coordination constructions.

Syntactically, they allow analyses of extraction and coordinate constructions which use the same lexical categories for the heads of such constructions as in the canonical case. Semantically, they guarantee that non-local dependencies fill the same argument slots as local dependencies.

Vijay-Shanker and Weir (1994) demonstrate that both CCG and Tree-Adjoining Grammar (TAG, Joshi *et al.* (1975)) are weakly equivalent to Linear Indexed Grammar (LIG) and belong to a family of languages whose generative power they identify as "mildly context-sensitive". Therefore, CCG is more expressive than AB categorial grammar, which has been shown by Bar-Hillel *et al.* (1960) to be context-free. Note that combinatory logic itself does not impose any restrictions on the generative power of such combinatory rules. However, Steedman (2000, p. 54) advocates the following principles to which all combinatory rules must adhere in order to keep the generative power of the grammar under control:

The Principle of Adjacency Combinatory rules may only apply to finitely many phonologically realized and string-adjacent entities

The Principle of Consistency All syntactic combinatory rules must be consistent with the directionality of the principal function.

The Principle of Inheritance If the category that results from the application of a combinatory rule is a function category, then the slash defining directionality for a given argument in that category will be the same as the one(s) defining directionality for the corresponding argument(s) in the input function(s).

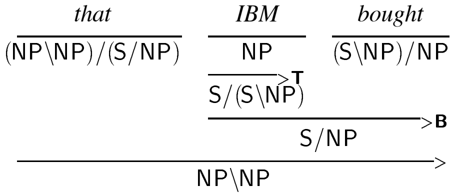
Composition allows two functor categories to combine to form another functor, whereas type-raising is a unary rule which reverts the roles of functor and argument by allowing an argument category X to change into a functor category $\mathsf{T/(T\backslash X)}$ (or $\mathsf{T\backslash(T/X)}$), where $\mathsf{T\backslash X}$ can be instantiated by any functor category that takes X as argument.

(6) a. *Forward Composition*:
$\mathsf{X/Y}: f \quad \mathsf{Y/Z}: g \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X/Z}: \lambda x.f(g(x))$
b. *Forward Crossing Composition*:
$\mathsf{X/Y}: f \quad \mathsf{Y\backslash Z}: g \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X\backslash Z}: \lambda x.f(g(x))$
c. *Backward Composition*:
$\mathsf{Y\backslash Z}: g \quad \mathsf{X\backslash Y}: f \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X\backslash Z}: \lambda x.f(g(x))$
d. *Backward Crossing Composition*:
$\mathsf{Y/Z}: g \quad \mathsf{X\backslash Y}: f \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X/Z}: \lambda x.f(g(x))$

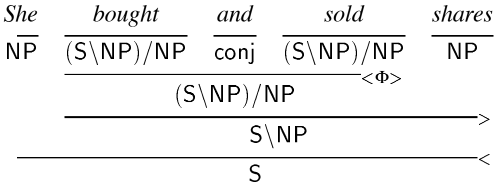
(7) a. *Forward Type-raising*:
$\mathsf{X}: a \quad \Rightarrow_{\mathbf{T}} \quad \mathsf{T/(T\backslash X)}: \lambda f.f(a)$
where $\mathsf{T\backslash X}$ is a parametrically licensed category for the language.
b. *Backward Type-raising*:
$\mathsf{X}: a \quad \Rightarrow_{\mathbf{T}} \quad \mathsf{T\backslash(T/X)}\lambda f.f(a)$
where $\mathsf{T/X}$ is a parametrically licensed category for the language.

Composition and Type-raising interact to capture the kinds of long-distance dependencies involved in extraction (8a) and right node raising (8b) as well as argument cluster coordinations (8c) among others:

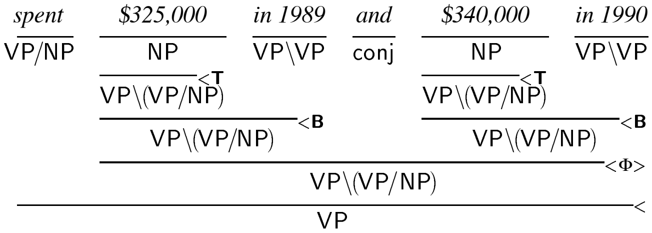
(8) a.



b.



c.



Note that in all of these constructions the verbs, *bought* and *spent*, have the same lexical categories as when their arguments are in canonical position.

In fact, in CCG all bounded and unbounded dependencies are projected from the lexicon, something which is expressed by the following two principles (Steedman, 2000, p.32):

The Principle of Lexical Head Government: Both bounded and unbounded syntactic dependencies are specified by the lexical syntactic type of their head.

The Principle of Head Categorial Uniqueness: A single nondisjunctive lexical category for the head of a given construction specifies both the bounded dependencies that arise when its complements are in canonical position and the unbounded dependencies that arise when those complements are displaced under relativization, coordination, and the like.

As stated above, composition is only allowed into functions of one argument. Note that this is also the case in the argument cluster construction above, since the second functor has only one argument $\mathsf{VP/NP}$, albeit a complex functor category in itself. However, generalized composition is required for sentences such as the following (Steedman, 2000, p.42):

$$\begin{array}{lccccccc}
(9) & \underline{\textit{I}} & \underline{\textit{offered}} & \underline{\textit{and}} & \underline{\textit{may}} & \underline{\textit{give}} & \underline{\textit{a flower}} & \underline{\textit{to a policeman}} \\
 & \mathsf{NP} & \mathsf{(S\backslash NP)/PP/NP} & \mathsf{conj} & \mathsf{(S\backslash NP)/(S\backslash NP)} & \mathsf{(S\backslash NP)/PP/NP} & \mathsf{NP} & \mathsf{PP} \\
\cline{5-6}
 & & & & \multicolumn{2}{c}{\mathsf{(S\backslash NP)/PP/NP} \quad {>}\mathbf{B}^2} & &
\end{array}$$

Steedman defines a "\$ convention" which allows him to schematize over functor categories with a varying number of argument but the same target, or innermost result category.

(10) The \$ convention:
For a category α, $\alpha\$$ (respectively $\alpha/\$$, $\alpha\backslash\$$) denotes the set containing α and all functions (respectively leftward function, rightward functions) into a category in $\alpha\$$ (respectively $\alpha/\$$, $\alpha\backslash\$$).

Then generalized composition can be defined as follows:

(11) a. *Generalized Forward Composition*:
$\mathsf{X/Y}{:}f \quad (\mathsf{Y/Z})/\$_1{:}...\lambda z.gz... \quad \Rightarrow_{>\mathbf{B}^n} \quad (\mathsf{X/Z})/\$_1{:}...\lambda z.f(g(z...))$

b. *Generalized Forward Crossing Composition*:
$\mathsf{X/Y}{:}f \quad (\mathsf{Y\backslash Z})\$_1{:}...\lambda z.gz... \quad \Rightarrow_{>\mathbf{B}^n{}_\times} \quad (\mathsf{X/Z})\$_1{:}...\lambda z.f(g(z...))$

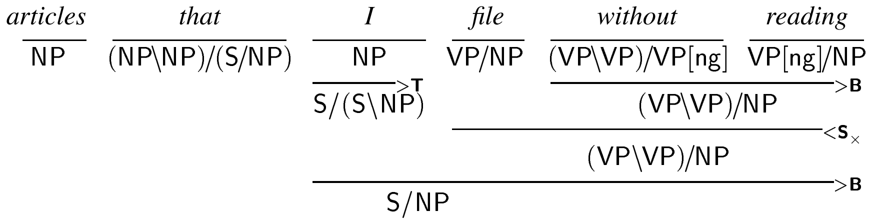
c. *Generalized Backward Composition*:
$(\mathsf{Y\backslash Z})\backslash\$_1{:}...\lambda z.gz... \quad \mathsf{X\backslash Y}{:}f \quad \Rightarrow_{<\mathbf{B}^n} \quad (\mathsf{X\backslash Z})\backslash\$_1{:}...\lambda z.f(g(z...))$

d. *Generalized Backward Crossing Composition*:
$(\mathsf{Y\backslash Z})/\$_1{:}...\lambda z.gz... \quad \mathsf{X\backslash Y}{:}f \quad \Rightarrow_{<\mathbf{B}^n{}_\times} \quad (\mathsf{X\backslash Z})\$_1{:}...\lambda z.f(g(z...))$

Each of these rules corresponds to a family of rules for each arity *n* of the secondary functor. Without any restriction on the arity of the secondary functor, full context-sensitivity would be obtained. However, there has not yet been any evidence that this is required to capture natural language syntax. Therefore, only schemata up to a bounded arity *n* (Steedman assumes 4 for English) are allowed in practice. The restrictions on the type-raising rules stated above serve a similar purpose. Together, they preserve mild context-sensitivity.

Another combinatory rule, substitution, is required for parasitic gaps, such as the following example:

(12)



Substitution is defined as follows:

(13) a. *Forward Crossing Substitution*:
$\mathsf{(X/Y)\backslash Z}{:}f \quad \mathsf{Y\backslash Z}{:}g \quad \Rightarrow_{\mathbf{S}} \quad \mathsf{X\backslash Z}{:}\lambda x.fx(g(x))$

b. *Backward Substitution*:
$\mathsf{Y\backslash Z}{:}g \quad \mathsf{(X\backslash Y)\backslash Z}{:}f \quad \Rightarrow_{\mathbf{S}} \quad \mathsf{X\backslash Z}{:}\lambda x.fx(g(x))$

c. *Backward Crossing Substitution*:
$\mathsf{Y/Z}{:}g \quad \mathsf{(X\backslash Y)/Z}{:}f \quad \Rightarrow_{\mathbf{S}} \quad \mathsf{X/Z}{:}\lambda x.fx(g(x))$

d. *Forward Substitution*:
$\mathsf{(X/Y)/Z}{:}f \quad \mathsf{Y/Z}{:}g \quad \Rightarrow_{\mathbf{S}} \quad \mathsf{X/Z}{:}\lambda x.fx(g(x))$

2.4 Normal-form derivations

“Spurious” ambiguity arises through the availability of type-raising and composition. Depending on the lexical categories, these operations may make any binary branching derivation structure for a string available. Under these circumstances, the number of derivations grows as a function of the Catalan number[4] of the words in the string. However, Hepple and Morrill (1989) show for a fragment of CCG consisting of application, forward type-raising and forward composition that derivations can always be reduced to a normal form, in which composition and type-raising are only used when syntactically necessary. Vijay-Shanker and Weir (1990) demonstrate that spurious ambiguity that arises through composition can be detected and eliminated by comparing alternative derivations of the form $((\alpha\beta)\gamma)$ and $(\alpha(\beta\gamma))$ for substrings α, β and γ. If the right-branching derivation structure is considered normal-form, then it suffices to mark all derivations that have a left-branching structure and to prefer unmarked derivations whenever there is a choice. Although Vijay-Shanker and Weir propose this to be done in a stage following recognition, their method could also be used during parsing. Such a proposal is made by Eisner (1996), who considers a restricted version of CCG without type-raising. Like Vijay-Shanker and Weir, Eisner suggests to mark constituents that are not in normal form. However, he makes the slightly stronger proposal that it suffices to assign tags to categories that are produced by composition rules, so that no constituent which is the result of a forward (backward) composition can serve as the primary functor in another forward (backward) composition.

The derivations in CCGbank are in a normal form which uses type-raising and composition only when necessary, eg. for relative clauses, right node raising and argument cluster coordination. Composition is also used when the adjuncts are combined with their heads: VP adjuncts all have the category (S\NP)\(S\NP) or (S\NP)/(S\NP), regardless of whether they modify a constituent with category S\NP, (S\NP)/NP, or ((S\NP)/NP)/PP.

2.5 Predicate-argument structure in CCGbank

In this report, the term predicate-argument structure is used to refer to the list of local and long-range dependencies between lexical items that are defined by a CCG derivation. The `*.parg` and the `*.html` files

[4] The Catalan number $C_n = \frac{(2n)!}{(n+1)!n!}$ gives the number of possible binary bracketings of a string of $n+1$ words.

contain this list for each derivation in CCGbank. These bilexical dependencies can be used to approximate the underlying logical form. They can also serve as features in the probability models used by statistical parsers (see eg. Hockenmaier (2003b) and Clark and Curran (2004)).

This section explains how predicate-argument structure is represented in CCGbank and gives a description of the data structures and operations that implement this representation in our parser (Hockenmaier, 2003a). Categories and their lexical dependencies are represented as feature structures, following previous work in categorial grammar such as Zeevat *et al.* (1987), Uszkoreit (1986) and Villavicencio (2002). However, these feature structures are merely an implementational device to express the information represented in CCG categories. General operations that might raise the generative power of the grammar (as discussed by Carpenter (1991)) are not allowed.

After some introductory terminology, lexical entries for proper nouns, intransitive verbs and adverbs are given, and function application is used to exemplify the *unify*-operation which is necessary to implement the combinatory rules. Then, implementations of coordination and the combinatory rules of type-raising, composition and substitution are presented. In CCGbank, a number of non-combinatory rules are used to deal with complex adjuncts and extraposed phrases and to represent coordination. Although these rules are only motivated in detail in the following chapter, their effect on predicate-argument structure is described here.

In CCG, bounded and unbounded dependencies are projected from the lexicon. For instance, the lexical catgory for a relative pronoun is $(\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S}\backslash\mathsf{NP})$ or $(\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S}/\mathsf{NP})$: the relative pronoun takes a sentence missing a subject or an object to its right and a noun phrase to its left.[5] The entire constituent is also a noun phrase. The noun phrase argument of the relative pronoun is also the missing object or subject of its sentential argument. Informally, this identity relation can be represented by co-indexing the two NPs, eg. $(\mathsf{NP}\backslash\mathsf{NP}_i)/(\mathsf{S}\backslash\mathsf{NP}_i)$. We will use this example to demonstrate how such identity relations can be implemented to obtain the correct predicate-argument dependencies.

The lexical categories in CCGbank have to be augmented with appropriate co-indexation information in order to represent these projected dependencies. Therefore, each lexical category in the `*.auto` files is represented both by its syntactic type (eg. $(\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S[dcl]}\backslash\mathsf{NP})$) and by a string representation of the underlying feature structure. This section concludes with a brief description of the dependencies that are projected by the lexical categories of CCGbank.

2.5.1 Category objects

We will distinguish categories (the abstract types the grammar is defined over) from category objects (the data structures a parser operates on); however, this distinction is omitted when the difference is clear, so that a symbol like $\mathsf{S[dcl]}\backslash\mathsf{NP}$ can either refer to the abstract category or to a data structure.

Atomic categories are categories without any arguments, eg. $\mathsf{S[dcl]}$ (declarative sentence), NP (noun phrase) etc. We assume a simple feature system such as the one described in section 3.4, whereby two atomic categories of the same type (eg. S) can unify if their features match. This is the case if either both carry the same feature (eg. $\mathsf{S[dcl]}$ and $\mathsf{S[dcl]}$), or if only one of them carries a feature. Hence, $\mathsf{S[dcl]}$ and S match, but $\mathsf{S[b]}$ and $\mathsf{S[dcl]}$ do not match.

Complex categories are functor categories with one or more arguments, eg. $\mathsf{NP[nb]}/\mathsf{N}$, $(\mathsf{S[dcl]}\backslash\mathsf{NP})/\mathsf{NP}$, $(\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S[dcl]}/\mathsf{NP})$. The arguments of complex categories are numbered from 1 to *n*, starting at the innermost argument, where *n* is the arity of the functor, eg. $(\mathsf{S[dcl]}\backslash\mathsf{NP}_1)/\mathsf{NP}_2$, $(\mathsf{NP}\backslash\mathsf{NP}_1)/(\mathsf{S[dcl]}/\mathsf{NP})_2$. Arguments can themselves be complex categories; for instance the second argument of $(\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S[dcl]}/\mathsf{NP})$ is $\mathsf{S[dcl]}/\mathsf{NP}$. Two complex categories of the same type match if each of their corresponding arguments match.

[5] Here and in the remainder of this report we assume the (semantically incorrect) relative clause analysis that is obtained from the Penn Treebank, in which the relative clause attaches at the NP level. In order to obtain correct scope, it should attach at the $\overline{\mathrm{N}}$ level.

Category objects are the data structures that the parser operates on. They represent syntactic categories and the predicate-argument dependencies that correspond to these categories. An atomic category object C has three fields,
$\langle \textsc{cat}, \textsc{heads}, \textsc{deps} \rangle$:

- A category symbol CAT.
- A list of lexical heads HEADS.
- A list of (unfilled) dependency relations DEPS.

HEADS and DEPS can be empty or uninstantiated. DEPS is a list of dependency relations that hold for each of the element of the HEADS list. When both lists are instantiated with non-empty lists, each elements of DEPS specifies one relation ("X is the *i*th argument of the lexical head Y") that holds for all elements X of HEADS. Lexical functor categories instantiate the DEPS attribute of their arguments. In the following the DEPS attribute is usually omitted when both it and the HEADS attribute are instantiated to non-empty lists, and it is also sometimes omitted when it is an empty list.

A complex category object $\langle \textsc{cat}, \textsc{heads}, \textsc{deps}, \textsc{res}, \textsc{arg}, \textsc{dir} \rangle$ represents a functor with argument category ARG and result category RES. DIR indicates the directionality of the argument: *FW* (forward, /) or *BW* (backward, \).

In principle, this representation could be extended; in particular, a semantic interpretation (or logical form) which corresponds to the syntactic category could be provided. However, since logical forms are not represented in CCGbank, this is omitted in the present description. Similarly, in the current version of our parser, filled dependency relations are not stored within the categories themselves, but in another data structure that forms part of the representation of edges in the chart. Therefore, filled dependency relations are also omitted here.

The lexicon $\mathscr{L}$ is a set of category-word pairs (lexical entries) $\langle c, w \rangle$. If $\langle c, w \rangle$ is a lexical entry, then c is a lexical category of w. We consider words to be fully inflected word forms, eg. *buys*, *shares*. A derived category is a category object that arises from one or more lexical categories through the application of zero or more combinatory rules. Each derived category has a list of lexical heads. A lexical head is a $\langle c, w \rangle$ pair, where w is a word and c a symbol denoting a lexical category of w.

If the lexical category of a word is complex, the lexicon specifies dependency relations that hold between the heads of the arguments of the category and the head of the lexical category itself. Formally, we represent a **dependency relation** as a 3-tuple $\langle \langle c, w \rangle, i, \langle c', w' \rangle \rangle$, where c is a functor category with arity $\geq i$, and $\langle c', w' \rangle$ is a lexical head of the ith argument of c.

A dependency relation $\langle \langle c, w \rangle, i, \langle c', w' \rangle \rangle$ holds for a derived category C if $\langle c', w' \rangle$ is a lexical head of C, and C is the ith argument of $\langle c, w \rangle$. The attribute DEPS is a list of all dependency relations that hold for this particular category. We will use the abbreviation $Arg(i, \langle c, w \rangle)$ to indicate that the lexical heads of this category are the ith argument of $\langle c, w \rangle$. If HEAD has more than one element, this relation holds for each of its elements.

If an argument of a lexical category is complex, the lexicon can also specify **identity relations** between arguments of the complex argument and other arguments of the same category. There are two kinds of identity relations: the **identity of categories** is used to indicate that the result and argument categories of a modifer $\mathsf{X/X}$ or $\mathsf{X\backslash X}$ are the same, whereas the **identity of heads** is used to indicate that head of the result X of a functor $\mathsf{X/Y}$ is the same as the head of the functor itself. Each category object has therefore two indices: a head index and a category index. Objects with the same category index necessarily have the same head index, but not vice versa. Head identity relations also encode certain kinds of (bounded and unbounded) non-local dependencies. For instance, in the lexical category for object extraction relative pronouns, $\mathsf{(NP\backslash NP)/(S[dcl]/NP)}$, the head of the first argument is identical to the head of the NP argument of the second argument. We indicate this identity by indices on the categories: $\mathsf{(NP\backslash NP_{\mathit{i}})/(S[dcl]/NP_{\mathit{i}})}$.

Section 2.5.6 explain this mechanism in detail, and appendix C lists all lexical entries in sections 02-21 where this mechanism is used.

2.5.2 Some lexical entries

Here is the atomic category object representing the NP *John*. The lexical head of this category is $\langle \mathsf{NP}, \textit{John}\rangle$. No dependency relations are specified.

$$\begin{bmatrix} \textsc{cat:} & \mathsf{NP} \\ \textsc{deps:} & \langle\ \rangle \\ \textsc{head:} & \langle\langle \mathsf{NP}, \textit{John}\rangle\rangle \end{bmatrix}$$

This is the complex category object for the intransitive verb *resigned*:[6]

$$\begin{bmatrix} \textsc{cat:} & \mathsf{S[dcl]\backslash NP} \\ \textsc{dir:} & \textit{BW} \\ \textsc{head:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \textit{resigned}\rangle\rangle\,\boxed{1} \\ \textsc{arg:} & \begin{bmatrix} \textsc{cat:} & \mathsf{NP} \\ \textsc{deps:} & \langle \textit{Arg}(\textit{1}, \langle \mathsf{S[dcl]\backslash NP}, \textit{resigned}\rangle)\rangle \\ \textsc{head:} & \langle\ \rangle \end{bmatrix} \\ \textsc{res:} & \begin{bmatrix} \textsc{cat:} & \mathsf{S[dcl]} \\ \textsc{deps:} & \langle\ \rangle \\ \textsc{head:} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Its lexical head is $\langle \mathsf{S[dcl]\backslash NP}, \textit{resigned}\rangle$. The head of the NP argument is empty; however, a dependency relation between the NP and the lexical head of the category is established. The head of the result is the same as the head of the entire category (indicated by the index $\boxed{1}$).

The category object of a modifier, such as the adverb $\langle (\mathsf{S\backslash NP})\backslash(\mathsf{S\backslash NP}), \textit{yesterday}\rangle$, is different from that of a verb or a noun phrase. When *yesterday* is combined with a verb or verb phrase, such as *resigned*, the head of the resulting constituent *resigned yesterday* is not the adverb, but the head of the verb phrase, *resigned*. There is a dependency relation between the lexical head of the modifier and the lexical head of the argument. The result category is unified with the argument category; therefore its head is the same as the head of the argument:

$$\begin{bmatrix} \textsc{cat:} & (\mathsf{S\backslash NP})\backslash(\mathsf{S\backslash NP}) \\ \textsc{dir:} & \textit{BW} \\ \textsc{head:} & \langle\langle (\mathsf{S\backslash NP})\backslash(\mathsf{S\backslash NP}), \textit{yesterday}\rangle\rangle \\ \textsc{arg:} & \boxed{1}\begin{bmatrix} \textsc{cat:} & \mathsf{S\backslash NP} \\ \textsc{deps:} & \langle \textit{Arg}(\textit{2}, \langle (\mathsf{S\backslash NP})\backslash(\mathsf{S\backslash NP})), \textit{yesterday}\rangle\rangle \\ \textsc{head:} & \langle\ \rangle \end{bmatrix} \\ \textsc{res:} & \boxed{1} \end{bmatrix}$$

When this category is applied to a verb phrase argument (or composed with a partially completed verb phrase), and the HEAD of its argument is instantiated with a non-empty list, the dependencies specified by the DEPS of the argument are filled. We could therefore omit the DEPS feature in the co-indexed result category RES.

[6] In CCGbank, categories are represented as strings, eg. $\mathsf{S[dcl]\backslash NP}$. Features such as $\mathsf{[dcl]}$ (declarative) or $\mathsf{[b]}$ (bare infinitive) are used to distinguish different kinds of sentences and verb phrases (see section 3.4 for a description of these features). These features ought to be represented directly as attribute-value pairs in the representation used in this chapter. However, for simplicity's sake, in CCGbank these features are also represented on the category strings. Since the grammar underlying CCGbank does not have agreement features, this is also omitted in the examples in this chapter.

We assume that the head of a noun phrase is the head of the noun. Therefore, the lexical category of a determiner such as $\langle \mathsf{NP[nb]/N}, \mathit{the}\rangle$ specifies that the head of its results is the same as the head of its argument:[7]

$$\begin{bmatrix} \textsc{cat:} & \mathsf{NP[nb]/N} \\ \textsc{dir:} & \mathit{FW} \\ \textsc{head:} & \langle\langle \mathsf{NP[nb]/N}, \mathit{the}\rangle\rangle \\ \textsc{arg:} & \begin{bmatrix} \textsc{cat:} & \mathsf{N} \\ \textsc{deps:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{NP[nb]/N}, \mathit{the}\rangle)\rangle \\ \textsc{head:} & \langle\,\rangle_{\boxed{1}} \end{bmatrix} \\ \textsc{res:} & \begin{bmatrix} \textsc{cat:} & \mathsf{NP} \\ \textsc{deps:} & \langle\,\rangle \\ \textsc{head:} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

The lexical category for possessive $\langle (\mathsf{NP[nb]/N})\backslash\mathsf{NP}, \mathit{'s}\rangle$ is defined analogously.

2.5.3 Function application and the *unify*-operation

When a functor category (such as the $\mathsf{S[dcl]\backslash NP}$ *resigned*) is applied to an argument (eg. to the NP *John*), its argument is unified with the category it is applied to:

$$\begin{bmatrix} \textsc{cat:} & \mathsf{S[dcl]\backslash NP} \\ \textsc{dir:} & \mathit{BW} \\ \textsc{head:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle\rangle_{\boxed{1}} \\ \textsc{arg:} & \begin{bmatrix} \textsc{cat:} & \mathsf{NP} \\ \textsc{deps:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle)\rangle \\ \textsc{head:} & \langle\langle \mathsf{NP}, \mathit{John}\rangle\rangle \end{bmatrix} \\ \textsc{res:} & \begin{bmatrix} \textsc{cat:} & \mathsf{S[dcl]} \\ \textsc{deps:} & \langle\,\rangle \\ \textsc{head:} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

This establishes the dependency relation $\langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle, 1, \langle \mathit{John}, \mathsf{NP}\rangle\rangle$ between *John* and *resigned*, and returns the result category of the functor:

$$\begin{bmatrix} \textsc{cat:} & \mathsf{S[dcl]} \\ \textsc{deps:} & \langle\,\rangle \\ \textsc{head:} & \langle\langle \mathit{resigned}, \mathsf{S[dcl]\backslash NP}\rangle\rangle \end{bmatrix}$$

We can define a unification operation $\mathit{unify}\ C, C \rightarrow C$ over category objects as follows:

- Two atomic category objects $C' = \langle \textsc{cat}', \textsc{heads}', \textsc{deps}'\rangle$ and $C'' = \langle \textsc{cat}'', \textsc{heads}'', \textsc{deps}''\rangle$ can unify if the values of $\textsc{cat}'$ and $\textsc{cat}''$ match. The result category $C = \langle \textsc{cat}, \textsc{heads}, \textsc{deps}\rangle$ is defined as follows:
  - CAT: unify $\textsc{cat}', \textsc{cat}''$.
  - HEADS: concatenate $\textsc{heads}'$ and $\textsc{heads}''$.
  - DEPS: concatenate $\textsc{deps}'$ and $\textsc{deps}''$.

[7] In the current version, this treatment was not adopted for lexical categories of determiners which have other categories, such as that of temporal modifiers (eg. *"that year"*).

If one of CAT' and CAT'' carries a feature (such as $\mathsf{S[dcl]}$), the result of unifying CAT' and CAT'' also carries this feature. If HEADS and DEPS are both non-empty, then all $dep \in$ DEPS must hold for all elements h of HEADS. Hence, the dependencies filled by this unification operation are given as the elements of the Cartesian product of HEADS and DEPS.

- Two complex category objects C' and C'' with $C' = \langle \text{CAT}', \text{HEADS}', \text{DEPS}', \text{ARG}', \text{RES}' \rangle$ and $C'' = \langle \text{CAT}'', \text{HEADS}'', \text{DEPS}'', \text{ARG}'', \text{RES}'' \rangle$ can unify if CAT' and CAT'' match. The result category $C = \langle \text{CAT}, \text{HEADS}, \text{DEPS}, \text{ARG}, \text{RES} \rangle$ is defined as follows:

  - CAT: unify CAT' and CAT''.
  - HEADS: concatenate HEADS' and HEADS''.
  - DEPS: concatenate DEPS' and DEPS''.
  - ARG: unify ARG' and ARG''
  - RES: unify RES' and RES''

  Here, unification of category strings is CAT' and CAT'' is extended in the obvious manner to deal with features on either CAT' or CAT''.

The concatenation of two lists L_1, L_2 is defined in the usual manner: if L_2 is empty, return L_1. Otherwise, append all elements of L_2 to the end of L_1, and return the result.

Function application is then defined as follows:

(14) A functor category $\mathsf{X'/Y'}$ can be applied to an argument category $\mathsf{Y''}$ if $\mathsf{Y'}$ and $\mathsf{Y''}$ unify to Y. If $\mathsf{X'/Y'}$ is applied to $\mathsf{Y''}$, unify $\mathsf{Y'}$ and $\mathsf{Y''}$ to yield a category $\mathsf{X/Y}$, where X is identical to $\mathsf{X'}$ except for any possible instantiation of variables that might have taken place through the unification of $\mathsf{Y'}$ and $\mathsf{Y''}$. Return the result X of $\mathsf{X/Y}$.

Above, it was stated informally that in the lexical entry of the adverb $\mathsf{(S\backslash NP)\backslash(S\backslash NP)}$ *yesterday*, the result $\mathsf{S\backslash NP}$ is unified with the argument $\mathsf{S\backslash NP}$. Here is the intermediate result of applying *yesterday* to *resigned*; the result category is co-indexed with the argument category:

$$\left[\begin{array}{ll}
\text{CAT:} & \mathsf{(S\backslash NP)\backslash(S\backslash NP)} \\
\text{DIR:} & \mathit{BW} \\
\text{HEAD:} & \langle\langle \mathsf{(S\backslash NP)\backslash(S\backslash NP)}, \mathit{yesterday}\rangle\rangle \\
\text{ARG:} & \boxed{1}\left[\begin{array}{ll}
\text{CAT:} & \mathsf{S[dcl]\backslash NP} \\
\text{DEPS:} & \langle \mathit{Arg}(\mathit{2}, \langle \mathsf{(S\backslash NP)\backslash(S\backslash NP)}, \mathit{yesterday}\rangle)\rangle \\
\text{DIR:} & \mathit{BW} \\
\text{HEAD:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle\rangle_{\boxed{2}} \\
\text{ARG:} & \left[\begin{array}{ll}
\text{CAT:} & \mathsf{NP} \\
\text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle)\rangle \\
\text{HEAD:} & \langle\ \rangle
\end{array}\right] \\
\text{RES:} & \left[\begin{array}{ll}
\text{CAT:} & \mathsf{S[dcl]} \\
\text{DEPS:} & \langle\ \rangle \\
\text{HEAD:} & \boxed{2}
\end{array}\right]
\end{array}\right] \\
\text{RES:} & \boxed{1}
\end{array}\right]$$

Thus, if we omit the (filled) dependency between *yesterday* and *resigned*, the result of this function application is like the category of *resigned* itself:

$$\begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle)\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle\rangle_{\boxed{1}} \end{bmatrix} \end{bmatrix}$$

When this verb phrase combines with a subject such as *John*, the subject is an argument of the verb, not the verb phrase modifier.

2.5.4 The combinatory rules

This section explains how the combinatory rules type-raising, composition and substitution are implemented.

Type-raising Type-raising a constituent with category X and lexical head H_X to $\mathsf{T/(T\backslash X)}$ results in a category whose head is H_X. The two T categories unify, and have the same head as the argument $\mathsf{T\backslash X}$. Backward type-raising is defined in a similar fashion.

Here is the NP *John* typeraised to $\mathsf{S/(S\backslash NP)}$:

$$\begin{bmatrix} \text{CAT:} & \mathsf{S/(S\backslash NP)} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \langle\langle \mathsf{NP}, \mathit{John}\rangle\rangle_{\boxed{2}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{S\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \boxed{1} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \boxed{2} \end{bmatrix} \\ \text{RES:} & \boxed{3}\begin{bmatrix} \text{CAT:} & \mathsf{S} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \text{RES:} & \boxed{3} \end{bmatrix}$$

This can then be applied to the $\mathsf{S[dcl]\backslash NP}$ *resigned* (since the $\mathsf{S\backslash NP}$ and $\mathsf{S[dcl]\backslash NP}$ match). Here is the intermediate result of unifiying the argument of the type-raised category with *resigned*:

$$\begin{bmatrix} \text{CAT:} & \mathsf{S/(S\backslash NP)} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \langle\langle \mathsf{NP}, \mathit{John}\rangle\rangle_{\boxed{2}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \boxed{1} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle))\rangle \\ \text{HEAD:} & \boxed{2} \end{bmatrix} \\ \text{RES:} & \boxed{3}\begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle\rangle_{\boxed{1}} \end{bmatrix} \end{bmatrix} \\ \text{RES:} & \boxed{3} \end{bmatrix}$$

And here is the result category of this application:

$$\begin{bmatrix} \text{CAT:} & \mathsf{S} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{S[dcl]\backslash NP}, \mathit{resigned}\rangle\rangle_{\boxed{1}} \end{bmatrix}$$

Composition Function composition is defined in a similar manner to function application. Here we only give the definition of forward composition, since backward composition and the crossing variants are defined analogously.

(15) A (primary) functor category $\mathsf{X'/Y'}$ can be composed with a (secondary) functor category $\mathsf{Y''/Z''}$ if $\mathsf{Y'}$ and $\mathsf{Y''}$ unify to Y.[8] If $\mathsf{X'/Y'}$ is composed $\mathsf{Y''/Z'}$, unify $\mathsf{Y'}$ and $\mathsf{Y''}$ to yield categories $\mathsf{X/Y}$ and $\mathsf{Y/Z}$, where X is identical to $\mathsf{X'}$ and Z is identical to $\mathsf{Z''}$, except for any possible instantiation of variables that might have taken place through the unification of $\mathsf{Y'}$ and $\mathsf{Y''}$. The result of this composition is a category $\mathsf{X/Z}$.

Let us consider an example. A transitive verb such as *buys* has the following lexical entry:

$$\begin{bmatrix} \text{CAT:} & \mathsf{(S[dcl]\backslash NP)/NP} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{2}, \langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle))\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle))\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle\rangle_{\boxed{1}} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

[8] We make the usual assumption that the arity of $\mathsf{Y'}$ and $\mathsf{Y''}$ is bounded.

When the $\mathsf{S/(S\backslash NP)}$ *"John"* is composed with this category, its argument $\mathsf{S\backslash NP}$ is instantiated with the result $\mathsf{S[dcl]\backslash NP}$ of the $\mathsf{(S[dcl]\backslash NP)/NP}$ *"buys"*, which in turn instantiates the subject NP of *buys* with *John*. The S in the type-raised $\mathsf{S/(S\backslash NP)}$ (*"John"*) is instantiated with an $\mathsf{S[dcl]}$ headed by *buys*:

$$\begin{bmatrix} \text{CAT:} & \mathsf{S/(S\backslash NP)} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \langle\langle \mathsf{NP}, \mathit{John}\rangle\rangle_{\boxed{2}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \boxed{1} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathit{buys}, \mathsf{(S[dcl]\backslash NP)/NP}\rangle)\rangle \\ \text{HEAD:} & \boxed{2} \end{bmatrix} \\ \text{RES:} & \boxed{3}\begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathit{buys}, \mathsf{(S[dcl]\backslash NP)/NP}\rangle\rangle_{\boxed{1}} \end{bmatrix} \end{bmatrix} \\ \text{RES:} & \boxed{3} \end{bmatrix}$$

Unifying the $\mathsf{S[dcl]\backslash NP}$ of the transitive $\mathsf{(S[dcl]\backslash NP)/NP}$ *"buys"* with the $\mathsf{S\backslash NP}$ of the type-raised noun phrase does not instantiate any variables in the object NP of *"buys"*. Therefore, the resulting $\mathsf{S[dcl]/NP}$ (*"John buys"*) is as follows:

$$\begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]/NP} \\ \text{DIR:} & \mathit{FW} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(2, \langle\langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{buys}\rangle\rangle)\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

So-called argument cluster coordination is another example where composition and type-raising are required. This is a construction where, unlike in the case considered above, composition does not result in any filled dependencies. Consider the following derivation:[9]

(16)

$$\begin{array}{cccccc} \mathit{give} & \mathit{a\ dog} & \mathit{a\ bone} & \mathit{and} & \mathit{a\ policeman} & \mathit{a\ flower} \\ \hline \mathsf{TV/NP} & \mathsf{NP} & \mathsf{NP} & \mathsf{conj} & \mathsf{NP} & \mathsf{NP} \\ \cline{2-3}\cline{5-6} & \mathsf{TV\backslash(TV/NP)}\ {<}\mathbf{T} & \mathsf{VP\backslash TV}\ {<}\mathbf{T} & & \mathsf{TV\backslash(TV/NP)}\ {<}\mathbf{T} & \mathsf{VP\backslash TV}\ {<}\mathbf{T} \\ \cline{2-3}\cline{5-6} & \multicolumn{2}{c}{\mathsf{VP\backslash(TV/NP)}\ {<}\mathbf{B}} & & \multicolumn{2}{c}{\mathsf{VP\backslash(TV/NP)}\ {<}\mathbf{B}} \\ \cline{2-6} & \multicolumn{5}{c}{\mathsf{VP\backslash(TV/NP)}\ {<}\Phi{>}} \\ \hline \multicolumn{6}{c}{\mathsf{VP}\ {<}} \end{array}$$

This is the type-raised category for *a bone*:

[9] We use TV to abbreviate the category of a transitive verb, $\mathsf{(S\backslash NP)/NP}$.

$$\left[\begin{array}{ll}
\text{CAT:} & (\mathsf{S}\backslash\mathsf{NP})\backslash((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP}) \\
\text{DIR:} & \mathit{BW} \\
\text{ARG:} & \left[\begin{array}{ll}
\text{CAT:} & (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} \\
\text{DIR:} & \mathit{FW} \\
\text{ARG:} & \left[\begin{array}{ll} \text{CAT:} & \mathsf{NP} \\ \text{HEAD:} & \langle\langle \mathsf{N}, \mathit{dog}\rangle \end{array}\right] \\
\text{RES:} & \boxed{1}
\end{array}\right] \\
\text{RES:} & \boxed{1}\left[\begin{array}{ll}
\text{CAT:} & \mathsf{S}\backslash\mathsf{NP} \\
\text{DIR:} & \mathit{BW} \\
\text{ARG:} & \left[\text{CAT:}\ \mathsf{NP}\right] \\
\text{RES:} & \left[\text{CAT:}\ \mathsf{S}\right]
\end{array}\right]
\end{array}\right]$$

And here is the (simplified) type-raised category for *a dog*:

$$\left[\begin{array}{ll}
\text{CAT:} & ((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP})\backslash(((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP})/\mathsf{NP}) \\
\text{DIR:} & \mathit{BW} \\
\text{ARG:} & \left[\begin{array}{ll}
\text{CAT:} & ((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP})/\mathsf{NP} \\
\text{DIR:} & \mathit{FW} \\
\text{ARG:} & \left[\begin{array}{ll} \text{CAT:} & \mathsf{NP} \\ \text{HEAD:} & \langle\langle \mathsf{N}, \mathit{dog}\rangle \end{array}\right] \\
\text{RES:} & \boxed{1}
\end{array}\right] \\
\text{RES:} & \boxed{1}\left[\begin{array}{ll}
\text{CAT:} & (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} \\
\text{DIR:} & \mathit{FW} \\
\text{ARG:} & \left[\text{CAT:}\ \mathsf{NP}\right] \\
\text{RES:} & \left[\text{CAT:}\ \mathsf{S}\backslash\mathsf{NP}\right]
\end{array}\right]
\end{array}\right]$$

This is the result of composing *a bone* with *a dog* – no dependencies are filled:

$$\left[\begin{array}{ll}
\text{CAT:} & ((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP})\backslash(((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP})/\mathsf{NP}) \\
\text{DIR:} & \mathit{BW} \\
\text{ARG:} & \left[\begin{array}{ll}
\text{CAT:} & ((\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP})/\mathsf{NP} \\
\text{DIR:} & \mathit{FW} \\
\text{ARG:} & \left[\begin{array}{ll} \text{CAT:} & \mathsf{NP} \\ \text{HEAD:} & \langle\langle \mathsf{N}, \mathit{bone}\rangle \end{array}\right] \\
\text{RES:} & \left[\begin{array}{ll}
\text{CAT:} & (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} \\
\text{DIR:} & \mathit{FW} \\
\text{ARG:} & \left[\begin{array}{ll} \text{CAT:} & \mathsf{NP} \\ \text{HEAD:} & \langle\langle \mathsf{N}, \mathit{dog}\rangle \end{array}\right] \\
\text{RES:} & \boxed{1}\left[\text{CAT:}\ \mathsf{S}\backslash\mathsf{NP}\right]
\end{array}\right]
\end{array}\right] \\
\text{RES:} & \boxed{1}
\end{array}\right]$$

Substitution Backward crossing substitution is used to account for parasitic gaps in English, such as the following:

(17)
$$\begin{array}{ccccc}
\mathit{articles} & \mathit{that} & \mathit{I} & \mathit{file} & \mathit{without\ reading} \\
\hline
\mathsf{NP} & (\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S}/\mathsf{NP}) & \mathsf{NP} & (\mathsf{S[dcl]}\backslash\mathsf{NP})/\mathsf{NP} & ((\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP}))/\mathsf{NP} \\
\cline{4-5}
 & & & \multicolumn{2}{c}{(\mathsf{S[dcl]}\backslash\mathsf{NP})/\mathsf{NP}\quad{<}\mathbf{S}_{\times}}
\end{array}$$

Recall that backward crossing substitution is defined as follows:

(18) $\mathsf{Y/Z} \quad \mathsf{(X\backslash Y)/Z} \quad \Rightarrow_{\mathsf{S}} \mathsf{X/Z}$

Here, the Ys and Zs are both unified. Substitution is similar to conjunction in that the dependencies that hold for the Z argument of the resulting $\mathsf{X/Z}$ are the union of the dependencies that hold for the Z argument of the first functor and those that hold for the Z argument of the second functor. This operation can be implemented as follows:

(19) A functor category $\mathsf{Y'/Z'}$ and a functor category $\mathsf{(X''\backslash Y'')/Z''}$ can be combined through substitution if $\mathsf{Y'}$ and $\mathsf{Y''}$ unify to Y and if $\mathsf{Z'}$ and $\mathsf{Z''}$ unify to Z. If $\mathsf{Y'/Z'}$ and $\mathsf{(X''\backslash Y'')/Z''}$ are combined through substitution, unify $\mathsf{Y'}$ and $\mathsf{Y''}$ and $\mathsf{Z'}$ and $\mathsf{Z''}$ to yield categories $\mathsf{Y/Z}$ and $\mathsf{(X\backslash Y)/Z}$, where X is identical to $\mathsf{X'}$ except for any possible instantiation of variables that might have taken place through the unification of $\mathsf{Y'}$ and $\mathsf{Y''}$ and $\mathsf{Z'}$ and $\mathsf{Z''}$. The result of this substitution is a category $\mathsf{X/Z}$.

In the parasitic gap example, the $\mathsf{Y/Z}$ of the rule schema is instantiated with $\mathsf{(S[dcl]\backslash NP)/NP}$, and the $\mathsf{(X\backslash Y)/Z}$ with $\mathsf{((S\backslash NP)\backslash(S\backslash NP))/NP}$. This is the category object for *without reading*:

$$\begin{bmatrix} \text{CAT:} & \mathsf{((S\backslash NP)\backslash(S\backslash NP))/NP} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(2,\mathsf{(S[ng]\backslash NP)/NP},\mathit{reading}\rangle))\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{(S\backslash NP)\backslash(S\backslash NP)} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \\ \text{ARG:} & {}^{\boxed{2}}\begin{bmatrix} \text{CAT:} & \mathsf{S\backslash NP} \\ \text{DIR:} & \mathit{BW} \end{bmatrix} \\ \text{RES:} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

The $\mathsf{(S\backslash NP)\backslash(S\backslash NP)}$ result of this category is similar to the adverb category given above for *yesterday* in that its result is unified with its argument. Hence, the X of the rule schema is unified with the Y. Therefore, substitution results in the following category for *file without reading*:

$$\begin{bmatrix} \text{CAT:} & \mathsf{(S[dcl]\backslash NP)/NP} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP},\mathit{file}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(2,\langle\mathsf{(S[dcl]\backslash NP)/NP},\mathit{file}\rangle),\mathit{Arg}(2,\mathsf{(S[ng]\backslash NP)/NP},\mathit{reading}\rangle))\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP},\mathit{file}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(1,\langle\mathsf{(S[dcl]\backslash NP)/NP},\mathit{file}\rangle)\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/NP},\mathit{file}\rangle\rangle_{\boxed{1}} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

2.5.5 Non-combinatory rules

The grammar underlying CCGbank (described in chapter 3) uses a number of non-combinatory rules in order to deal with punctuation, coordination, and certain types of complex adjuncts. This section describes how these rules can be implemented in the framework developed in this chapter.

Punctuation In CCGbank, punctuation marks such as commas, semicolons, full stops etc. do not have real CCG categories, but carry instead categories derived from their part-of-speech tags, such as ".", ";" etc. There are a number of rules dealing with punctuation, such as $\mathsf{S[dcl]}\ \ .\ \Rightarrow \mathsf{S[dcl]}$. We assume that there are no dependencies between words and punctuation marks, and that the result of such a rule is the same as the category of the non-punctuation category.

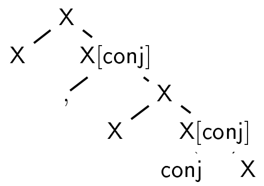
Here is the lexical category for the full stop:

$$\begin{bmatrix} \text{CAT:} & . \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle .,. \rangle\rangle \end{bmatrix}$$

Certain types of punctuation marks, such as opening brackets or dashes, can have specific lexical categories (such as $\langle(\mathsf{NP}\backslash\mathsf{NP})/\mathsf{S[dcl]},–\rangle$, see section 3.7.4) and are therefore treated like ordinary lexical items. The ampersand "&" is also treated like an ordinary lexical item. The plural possessive ' (as in *the parents'*) has also real categories (mainly $(\mathsf{NP}/\mathsf{N})\backslash\mathsf{NP}$), and is thus treated like an ordinary lexical item.

Coordination Coordination (see section 3.7.1) is encoded in CCGbank with an intermediate level $\mathsf{X[conj]}$, eg.:

(20)



This corresponds to rules of the form:

(21) a. $\mathsf{conj}\ \ \mathsf{X} \Rightarrow \mathsf{X[conj]}$
b. $,\ \ \mathsf{X} \Rightarrow \mathsf{X[conj]}$
c. $\mathsf{X}\ \ \mathsf{X[conj]} \Rightarrow \mathsf{X}$

Commas which occur in coordinate constructions have the same lexical category as described in the previous paragraph. Conjunctions have similar categories, eg.:

$$\begin{bmatrix} \text{CAT:} & . \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{conj}, \mathit{and} \rangle\rangle \end{bmatrix}$$

Coordination itself is implemented as follows: When coordinating two constituents X and X' with the same category, the result X'' is the unification of X and X' as defined above. Hence, the head and dependency relations of X'' and its subcategories are the concatenation of the corresponding head and dependency relations of X and X'. Here is the result of the coordination of two transitive verbs *buy* and *sell* with lexical category $(\mathsf{S[b]}\backslash\mathsf{NP})/\mathsf{NP}$:

$$\begin{bmatrix}
\text{CAT:} & (\mathsf{S[b]\backslash NP})/\mathsf{NP} \\
\text{DIR:} & \mathit{FW} \\
\text{HEAD:} & \langle\langle(\mathsf{S[b]\backslash NP})/\mathsf{NP}, \mathit{buys}\rangle, \langle(\mathsf{S[b]\backslash NP})/\mathsf{NP}, \mathit{sells}\rangle\rangle\,\boxed{1} \\
\text{ARG:} & \begin{bmatrix}
\text{CAT:} & \mathsf{NP} \\
\text{DEPS:} & \langle \mathit{Arg}(\mathit{2}, \langle(\mathsf{S[b]\backslash NP})/\mathsf{NP}, \mathit{buy}\rangle), \mathit{Arg}(\mathit{2}, \langle(\mathsf{S[b]\backslash NP})/\mathsf{NP}, \mathit{sell}\rangle)\rangle \\
\text{HEAD:} & \langle\ \rangle
\end{bmatrix} \\
\text{RES:} & \begin{bmatrix}
\text{CAT:} & \mathsf{S[b]\backslash NP} \\
\text{DIR:} & \mathit{BW} \\
\text{HEAD:} & \boxed{1} \\
\text{ARG:} & \begin{bmatrix}
\text{CAT:} & \mathsf{NP} \\
\text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle(\mathsf{S[b]\backslash NP})/\mathsf{NP}, \mathit{buy}\rangle), \mathit{Arg}(\mathit{1}, \langle(\mathsf{S[b]\backslash NP})/\mathsf{NP}, \mathit{sell}\rangle)\rangle \\
\text{HEAD:} & \langle\ \rangle
\end{bmatrix} \\
\text{RES:} & \begin{bmatrix}
\text{CAT:} & \mathsf{S[b]} \\
\text{DEPS:} & \langle\ \rangle \\
\text{HEAD:} & \boxed{1}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

Note that now there are two dependency relations that hold for each of the arguments. Furthermore, this category has two lexical heads.

In CCGbank, the coordination of constituents that do not have the same category (as in *"is 69 years old and chairman"*) is treated by the following rule schema (see section 3.7.2 for details):

(22) conj Y ⇒ X[conj]

This is implemented in the same way as the unary type-changing rules described in the next paragraph; the head of the resulting X[conj] is the same as the head of the Y conjunct.

Unary type-changing rules Apart from type-raising, there are several genuine type-changing rules in CCGbank. The simplest and most common one is N ⇒ NP. However, this is merely a change from a complete noun phrase to a bare noun phrase, which can be implemented by simply replacing the top-level category string N with NP. Another type of unary rule which appears once in every complete tree is of the form X ⇒ TOP. All other type-changing rules are of the form Y ⇒ X/X or Y ⇒ X\X, where a complement category Y is changed into an adjunct X/X or X\X (see section 3.8 for more details).

We assume that type-changing rules of the form Y ⇒ X/X or Y ⇒ X\X can project dependencies if Y is a complex category whose argument is of category X, as in the following rule for postmodifiers of nouns and noun phrases (again, using the semantically undesirable analysis of $\overline{\mathrm{N}}$ modifiers attaching at the NP level that is obtained from the Penn Treebank):

- S[dcl]/NP ⇒ NP\NP
  "the shares John bought"
- S[pss]\NP ⇒ NP\NP
  "the shares bought by John"

Binary type-changing rules CCGbank contains a number of binary type-changing rules that are triggered by certain kinds of NP extraction (explained in section 3.7.5), such as:

(23) a. *No dummies, the drivers pointed out they still had space (...)*
b. *Factories booked $236.74 billion in orders in September,*
[$_{NP}$ nearly the same as the $236.79 billion in August]

The binary type-changing rules are as follows:

$$\begin{array}{cclcl} \mathsf{NP} & , & \Rightarrow & \mathsf{S/S} \\ , & \mathsf{NP} & \Rightarrow & \mathsf{S\backslash S} \\ , & \mathsf{NP} & \Rightarrow & \mathsf{(S\backslash NP)\backslash(S\backslash NP)} \end{array}$$

Note that this is equivalent to assigning the comma categories such as $\mathsf{(S/S)\backslash NP}$. Being essentially anaphoric, it is difficult to establish a clear semantic relation between these extracted noun phrases and the sentence or verb phrase they belong to. Therefore, we assume for the purpose of the model that there is no dependency between them.

2.5.6 Co-indexation and non-local dependencies

As mentioned above, certain types of bounded and unbounded non-local dependencies are captured in the predicate-argument structure by co-indexation within complex lexical categories. For example, relative pronouns, auxiliaries, modals and control verbs are all cases where one (NP) argument fills the argument slot of another $\mathsf{S\backslash NP}$ argument. We will use the example of relative pronouns and of auxiliaries to show in detail the effect of co-indexation. Section 2.5.7 lists the different classes of lexical categories that project long-range dependencies in the current version of CCGbank.

Relative pronouns Here is the category object corresponding to the object extraction relative pronoun *that*:[10]

$$\begin{bmatrix} \text{CAT:} & \mathsf{(NP\backslash NP)/(S[dcl]/NP)} \\ \text{DIR:} & \mathit{FW} \\ \text{HEAD:} & \langle\langle \mathsf{(NP\backslash NP)/(S[dcl]/NP)}, \textit{that}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]/NP} \\ \text{DIR:} & \mathit{FW} \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{2}, \langle \mathsf{(NP\backslash NP)/(S[dcl]/NP)}, \textit{that}\rangle)\rangle \\ \text{HEAD:} & \langle\ \rangle \\ \text{ARG:} & \boxed{2} \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \boxed{1} \\ \text{ARG:} & \boxed{2}\begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{HEAD:} & \langle\ \rangle \\ \text{DEPS:} & \langle \mathit{Arg}(\mathit{1}, \langle \mathsf{(NP\backslash NP)/(S[dcl]/NP)}, \textit{that}\rangle)\rangle \end{bmatrix} \\ \text{RES:} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

This entry specifies that the head of the NP argument of the relative clause be the same as the head of the NP it modifies. Note that no dependency relations are introduced for the NP argument of the relative clause, but that instead the dependencies are percolated up to the NP argument of the relative pronoun. Here is the category of the relative clause *which John likes*:

[10] This category assumes that the relative clauses modifies the NP. However, in order to obtain the desired semantic interpretation of quantified noun phrases, it ought to be attached at the $\overline{\mathrm{N}}$ level. In the Penn Treebank, relative clauses and other modifiers are attached at the NP level. For simplicity's sake, we decided to keep this analysis in CCGbank, even though we believe it is incorrect.

$$\begin{bmatrix} \text{CAT:} & \mathsf{NP\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \langle\langle \mathsf{(NP\backslash NP)/(S[dcl]/NP)}, \mathit{which}\rangle\rangle \\ \text{ARG:} & \boxed{2}\begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(2,\langle \mathsf{(S[dcl]\backslash NP)/NP}, \mathit{likes}\rangle), \mathit{Arg}(1,\langle \mathsf{(NP\backslash NP)/(S[dcl]/NP)}, \mathit{which}\rangle)\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix} \\ \text{RES:} & \boxed{2} \end{bmatrix}$$

When this is applied to a noun phrase *books*, the intermediate result is as follows:

$$\begin{bmatrix} \text{CAT:} & \mathsf{NP\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \langle\langle \mathsf{(NP\backslash NP)/(S/NP)}, \mathit{which}\rangle\rangle \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(2,\langle \mathsf{S\backslash NP/NP}, \mathit{likes}\rangle), \mathit{Arg}(1,\langle\langle \mathsf{(NP\backslash NP)/(S/NP)}, \mathit{which}\rangle\rangle)\rangle \\ \text{HEAD:} & \langle\langle \mathit{books}, \mathsf{N}\rangle\rangle \end{bmatrix} \\ \text{RES:} & \boxed{2} \end{bmatrix}$$

Auxiliaries We assume that when an auxiliary verb such as *will* combines with an untensed verb such as *buy*, the subject NP is an argument of both verbs. However, the head of this constituent is the auxiliary. Therefore, the lexical entry for $\langle \mathsf{(S[dcl]\backslash NP)/(S[b]\backslash NP)}, \mathit{will}\rangle$ is as follows:

$$\begin{bmatrix} \text{CAT:} & \mathsf{(S[dcl]\backslash NP)/(S[b]\backslash NP)} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \langle\langle \mathsf{(S[dcl]\backslash NP)/(S[b]\backslash NP)}, \mathit{will}\rangle\rangle_{\boxed{1}} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[b]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \langle\ \rangle_{\boxed{2}} \\ \text{ARG:} & \boxed{3} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]\backslash NP} \\ \text{DIR:} & \mathit{BW} \\ \text{HEAD:} & \boxed{1} \\ \text{ARG:} & \begin{bmatrix} \text{CAT:} & \mathsf{NP} \\ \text{DEPS:} & \langle \mathit{Arg}(1,\langle \mathsf{(S[dcl]\backslash NP)/(S[b]\backslash NP)}, \mathit{will}\rangle)\rangle \\ \text{HEAD:} & \langle\ \rangle \end{bmatrix}_{\boxed{3}} \\ \text{RES:} & \begin{bmatrix} \text{CAT:} & \mathsf{S[dcl]} \\ \text{DEPS:} & \langle\ \rangle \\ \text{HEAD:} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Note that the subject NP argument of the untensed $\mathsf{S[b]\backslash NP}$ verb phrase argument is co-indexed with the subject NP of the auxiliary. When this category is applied to an $\mathsf{S[b]\backslash NP}$ argument (which in turn specifies dependencies on its subject NP), these dependencies also hold for the NP argument of the resulting tensed $\mathsf{S[dcl]\backslash NP}$ verb phrase.

2.5.7 Non-local dependencies in CCGbank

Non-local dependencies projected by the lexicon This section lists the types of lexical categories in the current version of CCGbank that project non-local dependencies. As described in the previous section, non-local dependencies are encoded by co-indexing arguments of complex arguments of the lexical category with other arguments of the same lexical category (eg $(\mathsf{S[.]\backslash NP}_i)/(\mathsf{S[.]\backslash NP}_i)$, so that there is a dependency of the co-indexed argument both on the head of this lexical category and on the head of the argument of this lexical category. Note that expletive *it* noun phrases ($\mathsf{NP[expl]}$) are not co-indexed with other arguments.

Our treatment assumes that all lexical categories of the same type project the same dependencies. While this generalization leads to the correct results in almost all cases, a more careful word-by-word examination of the co-indexation rules would be necessary to guarantee that the correct dependencies are obtained in all cases. We leave this for future versions of the corpus.

When evaluating the accuracy of different parsing models, we might want to distinguish between three different kinds of dependencies in the predicate-argument structure. **Local** argument/adjunct dependencies are dependencies between a head and its immediate argument or adjunct when both are in canonical position, and when the dependency is not mediated through another lexical item. There are two kinds of **non-local** dependencies: **locally mediated** dependencies are dependencies that arise through raising, control and related constructions. They are not strictly local, since they are projected through co-indexation in the lexical category of a third lexical item (such as a modal, auxiliary or control verb). Locally mediated dependencies hold between two arguments of the same lexical item, and are therefore bounded. **Long-range** dependencies are unbounded dependencies, such as object extraction, topicalization, *tough* movement and right node raising. Long-range dependencies that arise through co-indexation in the lexicon differ from locally mediated dependencies in that the projected dependency need not hold between two arguments of the lexical item whose category projects the dependency. For example, object relative pronouns ($(\mathsf{NP\backslash NP}_i)/(\mathsf{S[dcl]/NP}_i)$) project a long-range dependency between the noun phrase they modify and an (unboundedly distant) element inside the relative clause $\mathsf{S[dcl]/NP}$. However, the noun phrase itself need not be an argument of the head of the relative clause. According to this classification, unembedded subject relative pronouns project a locally mediated dependency between the noun phrase they modify and the head of the relative clause (whereas embedded subject relative clauses are analyzed like object relatives in CCG, and project an unbounded dependency).

A complete list of lexical entries from sections 02-21 that use this co-indexation mechanism to project dependencies is given in appendix C. Appendix D.2 describes how the indices are represented in the machine-readable files of CCGbank.

Auxiliaries, modals, raising verbs – $(\mathsf{S[.]\backslash NP}_i)/(\mathsf{S[.]\backslash NP}_i)$: In lexical categories that match this pattern, both NPs are co-indexed. Apart from auxiliaries, modals and raising verbs, this includes entries such as:

$$\langle \textit{expects}, (\mathsf{S[dcl]\backslash NP})/(\mathsf{S[to]\backslash NP})\rangle,$$
$$\langle \textit{began}, (\mathsf{S[dcl\backslash NP})/(\mathsf{S[ng]\backslash NP})\rangle,$$
$$\langle \textit{declined}, (\mathsf{S[dcl\backslash NP})/(\mathsf{S[to]\backslash NP})\rangle$$

These dependencies are locally mediated.

Control verbs – $((\mathsf{S[.]\backslash NP})/(\mathsf{S[.]\backslash NP}))/\mathsf{NP}$: in subject control verbs (forms of *promise*), the subject NP argument is co-indexed with the subject of the verb phrase argument:

$$((\mathsf{S[.]\backslash NP}_i)/(\mathsf{S[.]\backslash NP}_i))/\mathsf{NP}$$

All other verbs with a category matching $((\mathsf{S[.]\backslash NP})/(\mathsf{S[.]\backslash NP}))/\mathsf{NP}$ (eg. forms of *persuade*) are assumed to be object control. In these cases, the object NP is co-indexed with the subject of the verb phrase argument:

$$((\mathsf{S}[.]\backslash\mathsf{NP})/(\mathsf{S}[.]\backslash\mathsf{NP}_i))/\mathsf{NP}_i.$$

Other cases of object control are categories that have both a verb phrase and a noun phrase argument, such as $\langle$*find*, $((\mathsf{S}[\mathsf{dcl}\backslash\mathsf{NP})/(\mathsf{S}[\mathsf{ng}]\backslash\mathsf{NP}))/\mathsf{NP}\rangle$ etc. Furthermore, if a verb has a verb phrase argument, but no other noun phrase argument than the subject, the subject of the verb phrase argument is co-indexed with the subject of the verb. Dependencies projected by control verbs are also locally mediated

Subject extraction verbs – $((\mathsf{S}[.]\backslash\mathsf{NP})/\mathsf{NP}_i)/(\mathsf{S}[.]\backslash\mathsf{NP}_i)$**:** lexical entries where the category matches this pattern (unless the $\mathsf{S}[.]\backslash\mathsf{NP}$ is $\mathsf{S}[\mathsf{adj}]\backslash\mathsf{NP}$), are special cases of subject extraction from embedded sentences. See section 3.9.6. Since this analysis is similar to object extraction, we classify these dependencies as long-range.

VP modifiers – $((\mathsf{S}\backslash\mathsf{NP}_i) \mid (\mathsf{S}\backslash\mathsf{NP}_i))/(\mathsf{S}[.]\backslash\mathsf{NP}_i)$**:** In categories of pre- and post-verbal modifiers that take an additional verb phrase argument, the subject of their verb phrase argument is co-indexed with the subject of the verb phrase they modify. This is a locally mediated dependency.

Small clause PPs – $(\mathsf{X}/(\mathsf{S}[.]\backslash\mathsf{NP}_i))/\mathsf{NP}_i$**:** in categories that match this pattern (where X can be one of $\mathsf{S}[\mathsf{for}]$, PP, or an adjunct category, such as $\mathsf{NP}\backslash\mathsf{NP}$ or $(\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP})$), the noun phrase argument is co-indexed with the subject of the verb phrase argument.
Examples: $\langle$*with*, $(((\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP}))/(\mathsf{S}[\mathsf{ng}]\backslash\mathsf{NP}))/\mathsf{NP}\rangle$, $\langle$*for*, $(\mathsf{S}[\mathsf{for}]/(\mathsf{S}[\mathsf{to}]\backslash\mathsf{NP}))/\mathsf{NP}\rangle$.
This is also a locally mediated dependency.

Relative pronouns – $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S}[.]\backslash\mathsf{NP}_i)$ **and** $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S}[.]/\mathsf{NP}_i)$**:** The noun phrase argument (and result) of the lexical category of a relative pronoun is co-indexed with the noun phrase argument of the relative clause. Subject relative pronouns project locally mediated dependencies, whereas object relative pronouns project long-range dependencies. This analysis of relative clauses as adjuncts to the NP is obtained from the Penn Treebank annotation. However, users of CCGbank who are interested in semantic interpretation should note that this analysis will not yield the desired interpretation for quantified noun phrases.

Relative pronouns that take a noun argument – $(\mathsf{X}/(\mathsf{S}[.] \mid \mathsf{NP}_i))/\mathsf{N}_i$**:** The head of the noun argument is co-indexed with the head of the noun phrase argument of the relative clause.
Examples: $\langle$*whose*, $((\mathsf{NP}\backslash\mathsf{NP})/(\mathsf{S}[\mathsf{dcl}]/\mathsf{NP}))/\mathsf{N}\rangle$, $\langle$*whatever*, $(\mathsf{NP}/(\mathsf{S}[\mathsf{dcl}]/\mathsf{NP}))/\mathsf{N}\rangle$.
As before, we consider subject extraction a locally mediated dependency and object extraction long-range.

Yes-no question words – $(\mathsf{S}[\mathsf{q}]/(\mathsf{S}[.]\backslash\mathsf{NP}_i))/\mathsf{NP}_i$**:** The noun phrase argument is co-indexed with the subject of the verb phrase argument. This is a locally mediated dependency.

***Tough*-adjectives –** $(\mathsf{S}[\mathsf{adj}]\backslash\mathsf{NP}_i)/((\mathsf{S}[\mathsf{to}]\backslash\mathsf{NP})/\mathsf{NP}_i)$**:** eg. *tough*, *difficult*, *easy*, *impossible*, *reasonable*, *wrong*. These dependencies are long-range.

Non-local dependencies through coordination and type-changing rules Coordination of functor categories can also create unbounded long-range dependencies. If the direction of the argument of the left conjunct is forward (as in right node raising), this is a long-range dependency. Similarly, if the direction of the argument of the right conjunct is backward, then we classify this as a long-range dependency. This also includes VP coordination.

Unary type-changing rules can also create long-range dependencies. Type-changing rules that correspond to subject extraction are considered locally mediated, whereas type-changing rules that encode object extraction are considered long-range.

Distinguishing non-local from local dependencies In order to be able to distinguish locally mediated and long-range dependencies from local dependencies, each lexical functor category specifies for each of its arguments whether this dependencies are local or not, and if the dependencies are not local, whether they are locally mediated (bounded) or long-range (unbounded). For example, dependencies on the NP argument of the S/NP in the lexical category of a relative pronoun, (NP\NP)/(S/NP) are long-range dependencies, whereas the dependency between the relative pronoun and the head of the noun phrase it modifies is local. When the relative pronoun is applied to a relative clause, all dependencies that the relative clause S/NP defines for its argument are marked as unbounded long-range dependencies. Similarly, coordination of functor categories marks the dependencies on right arguments of left conjuncts and on left arguments of right conjuncts as unbounded long-range dependencies. This allows us to evaluate the parser's recovery of different classes of non-local dependencies.

2.5.8 A note on the depdendencies in CCGbank

Although we believe that the dependencies in CCGbank are largely linguistically motivated and correct, there are a number of problems that we have summarized below:

Problems due to the syntactic derivations: Some syntactic analyses in CCGbank are semantically undesirable or incorrect because the Penn Treebank annotation does not provide sufficient linguistic structure, and additional manual annotation would be required to obtain the correct analyses. Therefore, the internal structure of complex nouns, which is largely flat in the Penn Treebank, is mostly blatantly wrong. Similarly, a large number of NP coordinations in CCGbank are actually appositives, because these two constructions are not distinguished in the Penn Treebank. Also, all post-nominal modifiers such as relative clauses are attached at the NP level. Semantically, this is problematic, since it fails to distinguish between restrictive and non-restrictive relative clauses, for example. We also have reason to believe that the distinction between complements and adjuncts is somewhat inconsistent, in particular for PPs, but also for *to*-VPs and other cases. These inconsistencies are reflected in our lexical categories.

Problems due to the co-indexation of lexical categories: The co-indexation of lexical categories as described above is necessary to obtain the non-local dependencies that arise through control, raising, extraction and other constructions. We co-indexed all lexical categories of the same type in an identical manner, even though we believe that this should be done on a word-by-word basis. Inconsistencies in the complement-argument distinction in the Penn Treebank are reflected in our lexical entries.

Problems due to the nature of the representation: We would like to emphasize that the word-word dependencies in CCGbank are only an approximation to the underlying predicate-argument structure. They are not logical forms in themselves. This problem is most obvious in the case of VP coordination. For example, the dependencies of a sentences such as *"I was early yesterday and late today"* indicate that *yesterday* and *today* modify *was*, and that *early* and *late* are both complements of *was*, but fail to represent that this is a statement about two distinct events. Our implementation also fails to capture control dependencies in argument cluster coordination, eg. the dependencies between *you* and *go* and between *him* and *stay* in *"I want you to go and him to stay"*.

Chapter 3

The translation algorithm

This chapter presets an algorithm for translating phrase-structure trees from the Penn Treebank (Marcus *et al.*, 1993, 1994) to CCG normal-form derivation trees, notes some of the changes to the Penn Treebank representation that are necessary for the translation procedure to yield adequate CCG analyses, and describes CCGbank, the corpus that results from applying this algorithm to the Penn Treebank.

The work presented in this chapter is an extension of an algorithm to extract categorial lexicons from the Penn Treebank, which was originally presented in (Hockenmaier *et al.*, 2004) and (Hockenmaier *et al.*, 2000). Hockenmaier and Steedman (2002a) also describes an earlier version of CCGbank.

3.1 Introduction

The Penn Treebank (Marcus *et al.*, 1993, 1994) is the largest available manually parsed corpus of English, and has been used as the standard test and training material for statistical parsing of English (see e.g. Collins, 1999; Charniak, 1999). This report presents an algorithm for translating Penn Treebank trees to normal-form CCG derivations. The resulting corpus, CCGbank, can be used to test and train statistical parser. We can also obtain a large CCG lexicon from this corpus that can be used by any CCG parser. Moreover, since CCG derivations have a corresponding semantic interpretation, the creation of a corpus of CCG derivations can be seen as a first step towards a corpus annotated with logical forms.

This chapter is organized in the following way: section 3.3 presents an algorithm for translating simple phrase structure trees to CCG. After an overview of the feature system used by the Treebank CCG (section 3.4), the analysis of basic sentence and noun phrase structure as well as the necessary modifications to the algorithm are shown (sections 3.5–3.7). Section 3.8 shows how unary type-changing rules for certain types of adjuncts can be introduced into the grammar to ensure a compact lexicon without augmenting the generative power of the system, demonstrating that a wide coverage CCG does not require a prohibitively large lexicon. The algorithm is then extended to deal with the various kinds of null elements in the Penn Treebank which encode long-range dependencies arising through extraction (section 3.9) and coordination (section 3.10). The analysis of specific syntactic constructions is covered in detail; therefore this chapter serves a double purpose in providing a concise overview over the coverage and analyses of our CCG for English, which is largely based on Steedman (2000, 1996). In order to obtain the desired categorial derivation trees, some changes on the original Treebank trees need to be performed before translation. Changes that are required because the Treebank analysis does not conform to the CCG account are noted in sections 3.5–3.7. However, a large number of changes are necessary to correct inconsistencies and annotation errors in the data. These are discussed in sections 3.12 and in more detail in appendix B. In section 3.15 the algorithm developed here is compared with an alternative procedure for the acquisition of AB categorial grammar lexicons from a subcorpus of the Penn Treebank without null elements (Watkinson and Manandhar, 2001). A direct comparison with related algorithms for the extraction of Lexicalized Tree-Adjoining Grammars and Lexical Functional Grammars from the Penn Treebank is less straightforward, however. The size and coverage of the acquired lexicon and grammar are described in section 4.

The appendix gives more implementational details, such as the head-finding rules, the rules for distinguishing complements from adjuncts, and the changes made to the Penn Treebank in the current implementation.

3.2 The Penn Treebank

The Wall Street Journal subcorpus of the Penn Treebank contains about 1 million words of parsed and tagged Wall Street Journal text collected in 1989.

The Treebank markup encloses constituents in brackets, with a label indicating the part of speech tag or syntactic category. A typical example is shown here:

```
(S (PP-TMP (IN In)
       (NP (DT the) (NN past) (NN decade)))
   (, ,)
   (NP-SBJ (JJ Japanese) (NNS manufacturers))
   (VP (VBD concentrated)
       (PP-CLR (IN on)
        (NP (NP (JJ domestic) (NN production))
            (PP (IN for)
                (NP (NN export))))))
    (. .))
```

In the following, part of speech tags and other irrelevant details of the trees will be omitted when presenting examples.

The Treebank markup is designed so that complements and adjuncts can in general be distinguished, except for certain difficult cases, such as prepositional phrases. However, the complement-adjunct distinction is not always marked explicitly. Instead, we use heuristic procedures which rely on the label of a node and its parent to make this distinction. Syntactic heads are also not indicated explicitly, but existing head-finding procedures such as those originally given by Magerman (1994) were adapted to our purposes (see appendix A.1).

The Treebank markup uses different types of null elements to encode long-range dependencies arising through coordination and extraction. The presence of these null elements is what makes it possible to translate the Treebank trees to the corresponding CCG derivations for relative clauses, wh-questions and coordinate constructions such as right node raising. The treatment of these null elements is discussed in sections 3.9–3.10.

3.3 The basic algorithm

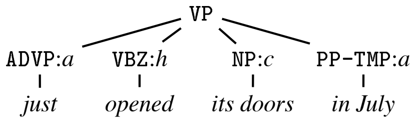
The basic algorithm for translating the Penn Treebank to CCG consists of three steps, each of which is a simple top-down recursive procedure:

```
(24) foreach tree $\tau$:
         determineConstituentType($\tau$);
         makeBinary($\tau$);
         assignCategories($\tau$);
```

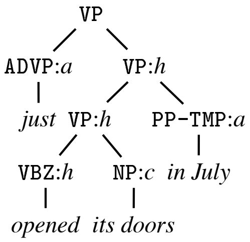
This algorithm presumes that the constituent structure of the original tree conforms to the desired CCG analysis. In practice, this is not always the case, and sections 3.5–3.7 give a detailed account of the reanalysis of specific constructions, and show how this algorithm can be adapted to deal with simple coordinate constructions. Sections 3.9 and 3.10 extend this algorithm to deal with the null elements in the Penn Treebank that encode long-range dependencies in constructions involving extraction and coordination.

This section explains the three steps of the basic algorithm.

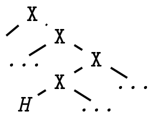
First, the constituent type of each node (head (*h*), complement (*c*), or adjunct (*a*)) is determined, using heuristics adapted from Magerman (1994) and Collins (1999) (for details, see appendix A.2):



Then the flat trees are transformed to binary trees.



This binarization process inserts dummy nodes into the tree such that all children to the left of the head branch off in a right-branching tree, and then all children to the right of the head branch off in a left-branching tree:



Categories are assigned to the nodes in a binary tree in the following manner (corresponding to a reverse CCG derivation):

The root node Each tree is rooted in a node labelled TOP. This node has one daughter, whose category is determined by the Treebank label of the root node of the Treebank tree (eg.$\{\mathtt{S},\mathtt{SINV},\mathtt{SQ}\} \rightarrow \mathsf{S}$, $\{\mathtt{VP}\} \rightarrow \mathsf{S\backslash NP}$). Section 3.4 explains the feature system used to distinguish different types of sentences (declarative, interrogative, embedded declarative, embedded interrogative) and verb phrases.

Complement nodes The category of a complement child is defined by a similar mapping from Treebank labels to categories.

Adjunct nodes Given a parent category X, the category of an adjunct child is a unary functor $\mathsf{X'/X'}$ if the adjunct child is the left daughter, or $\mathsf{X'\backslash X'}$ if it is the right daughter.

The category $\mathsf{X'}$ is determined by the parent category X. In order to avoid a proliferation of category types, adjunct categories do not carry any morphological features. This means for instance that VP adjuncts all have the category $\mathsf{(S\backslash NP)\backslash(S\backslash NP)}$ or $\mathsf{(S\backslash NP)/(S\backslash NP)}$ - that is, we do not distinguish between adjuncts appearing in declarative, infinitival, or passive verb phrases. *Quickly* is $\mathsf{(S\backslash NP)\backslash(S\backslash NP)}$ regardless of whether it modifies *buys*, *bought* or *buying*. In a version of categorial grammar without composition, $\mathsf{X'}$ would have to be equal to the current head category (without features). However, in the case of adjuncts of adjuncts, this leads to a proliferation of categories. But, in CCG, adjuncts can combine with the heads using (generalized) composition. Therefore, $\mathsf{X'}$ can be the current head catgory with the outermost arguments stripped off. Thus, the following strategy is used: A left adjunct, $\mathsf{X'/X'}$, can combine with the head using (generalized) forward non-crossing composition:

(25) *Forward Composition*:
$$\mathsf{X/Y} \quad \mathsf{Y/Z} \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X/Z}$$

Forward crossing composition is not permitted in English, since it would lead to greater freedom in word order than English allows:

(26) *Forward Crossing Composition*:
$$\mathsf{X/Y}: f \quad \mathsf{Y\backslash Z}: g \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X\backslash Z}$$

Hence, in the case of forward-looking (left) adjuncts ($\mathsf{X'/X'}$), $\mathsf{X'}$ is the parent category minus all outermost forward-looking arguments. $\mathsf{X'/X'}$ can then combine with the current head category through (generalized) forward non-crossing composition. If any backward-looking arguments were stripped off, $\mathsf{X'/X'}$ could only combine with the head through forward crossing composition.

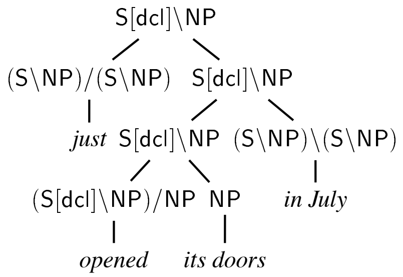
A CCG for English allows backward crossing as well as non-crossing composition. Therefore, in the case of backward-looking (right) adjuncts, $\mathsf{X'\backslash X'}$, all outermost arguments which have the same directionality as the last argument are stripped off from the parent category in order to obtain $\mathsf{X'}$ – that is, if the outermost argument of the current head category is forward-looking, then all outermost forward arguments are stripped off (corresponding to generalized backward crossing composition). If the outermost argument is backward-looking, all outermost backward arguments can be stripped off (generalized backward non-crossing composition).

In the case of VP-adjuncts, however, we stipulate that we do not generalize beyond the $\mathsf{S\backslash NP}$ level, since we want to distinguish verbal adjuncts from sentential adjuncts. Consider for instance the adjunct *in July*. Since its parent's category is $\mathsf{S\backslash NP}$ and it appears to the right of the head verb, it receives the category $\mathsf{(S\backslash NP)\backslash(S\backslash NP)}$.

Punctuation marks In general, the category of a punctuation mark is the POS tag assigned to the punctuation mark. Exceptions to this rule are discussed in section 3.7.4.

Head nodes The head child of a parent with category X has category X if the non-head child is an adjunct or a punctuation mark. If the non-head child is a complement with category Y, the category of the head child is X/Y if the head child is left, and X\Y if the head child is right.

Here is the previous tree annotated with CCG categories:



The category assignment procedure corresponds to a reverse derivation which always uses function application, except for adjuncts, where composition can be used in order to provide a more general analysis. The extensions to this algorithm described below use type-raising and composition only when syntactically necessary. Therefore, the derivations in CCGbank are in a normal form.

3.4 Atomic categories and features in CCGbank

We assume the atomic categories S, NP, N and PP, and employ features to distinguish between declarative sentences (S[dcl]), wh-questions (S[wq]), yes-no questions (S[q]), embedded declaratives (S[emb]) and embedded questions (S[qem]).[1] We also distinguish different kinds of verb phrases (S\NP), such as bare infinitives, to-infinitives, past participles in normal past tense, present participles, and past participles in passive verb phrases. This information is encoded as an atomic feature on the category, eg. S[pss]\NP for a passive VP, or S[dcl] for a declarative sentence. Predicative adjectives have the category S[adj]\NP.

The sentential features are as follows:

(27) a. S[dcl]: for declarative sentences
b. S[wq]: for wh-questions
c. S[q]: for yes-no questions (*Does he leave?*)
d. S[qem]: for embedded questions (*worry [whether he left]*)
e. S[em]: for embedded declaratives (*he says [that he left]*)
f. S[bem]: for embedded sentences in subjunctive mood (*I demand [that he leave]*)
g. S[b]: for sentences in subjunctive mood (*I demand (that) [he leave]*)
h. S[frg]: for sentence fragments (derived from the Treebank label `FRAG`)
i. S[for] for small clauses headed by *for* (*[for X to do sth]*)
j. S[intj]: for interjections
k. S[inv]: for elliptical inversion (*(as) [does President Bush]*)

These are the verb phrase features:

(28) a. S[b]\NP: for bare infinitives, subjunctives and imperatives
b. S[to]\NP: for to-infinitives

[1] In order to improve readability, we will sometimes abbreviate the verb phrase category S\NP as VP in derivations. A category such as VP[dcl] always stands for S[dcl]\NP. In some of the derivations given here, features are omitted.

c. S[pss]\NP: for past participles in passive mode

d. S[pt]\NP: for past participles used in active mode

e. S[ng]\NP: for present participles

We analyze attributive adjective phrases as S[adj]\NP.

The main purpose of these features is to specify subcategorization information—for instance, the verb *doubt* takes both embedded declaratives (*doubt that*) and questions (*doubt whether*) as argument, whereas *think* can only take embedded declaratives. The complementizer *that* takes a declarative sentence, and yields an embedded declarative: *that* ⊢ S[emb]/S[dcl]. These features are treated as atomic, and do not indicate the full morphosyntactic information. For instance, the infinitival particle *to* has the category (S[to]\NP)/(S[b]\NP), since it takes a bare infinitive (S[b]\NP) as its argument and yields a to-infinitival verb phrase S[to]\NP in both *”to give”* and *”to be given”*. Since the information whether the verb phrase is active of passive becomes available during the derivation, there is no need for separate lexical entries. The [dcl], [b], [ng], [pt] and [pss] features are determined from the POS tags of the head verbs and the presence of passive traces. [em], [qem], [q], [frg] and [tpc] are determined from the nonterminal labels of the trees. Adjunct categories other than modifiers of adjectives do not carry features.

Being derived from the POS tags in the Penn Treebank, this feature system is rather coarse; for instance it does not encode differences in verb mood: subjunctive and imperative appear as bare infinitive, whereas conditional and indicative appear both as declarative.

Determiners, such as *the*, are functions from noun phrases to nouns: NP[nb]/N[2]. Expletive *it* has the category NP[expl], and expletive *there* is NP[thr]. The category N[num] is for numbers that denote either amounts of a certain currency (eg. *“500 million”* in “$ 500 million″ or days of the month (eg. *30* in *“Nov. 30”*). In the first case, the currency symbol takes an N[num] argument; in the second case, the month takes the day as an argument.

There are a few features for specific function words or multiword expressions (explained in section 3.7.6), eg. S[as] and S[poss], and S[asup]\NP for *least*, *most* etc. in *at least.*

3.5 Basic clause structure

This section shows how standard CCG analyses of basic clause types can be obtained from the corresponding Treebank annotation. We look in turn at simple declarative sentences, infinitival and participial verb phrases, passive, control and raising, small clauses, yes-no questions, inverted sentences, ellipsis and fragments. Wh-questions are dealt with in section 3.9.2.

3.5.1 Simple declarative sentences

Declarative sentences are annotated as S in the Treebank. The annotation is flat, with modifiers and punctuation marks appearing at the same level as the subject noun phrase and main verb phrase:

```
(TOP (S (NP-TMP Today)
        (, ,)
        (NP-SBJ PC shipments)
        (ADVP-TMP annually)
        (VP (VBP total)
            (NP some $38.3 billion)
            (ADVP-LOC world-wide))
        (. .)))
```

[2] For historical reasons, the NP carries a “non-bare” [nb]-feature.

Since end-of-sentence punctuation marks modify the entire sentence, a separate sentence level needs to be inserted which includes everything but the end-of-sentence punctuation mark:

```
(TOP (S (S (NP-TMP Today)
           (, ,)
           (NP-SBJ PC shipments)
           (ADVP-TMP annually)
           (VP (VBP total)
               (NP some $38.3 billion)
               (ADVP-LOC world-wide)))
        (. .)))
```

Note that otherwise the binarization procedure would analyze the full stop as a modifier of $\mathsf{S\backslash NP}$. In order to improve readability, end-of-sentence punctuation marks will generally be omitted in the categorial derivation trees shown.

A sentence carries the feature [dcl] if its head verb has one of the following POS tags: `VBZ` (3rd person singular present), `VBP` (non-3rd-person singular present), `VBD` (past tense).

3.5.2 Infinitival and participial VPs, gerunds

In the Treebank, participial phrases, gerunds, imperatives and infinitival verb phrases are annotated as sentences with a * null subject (which can be co-indexed with another `NP` in the sentence, depending on the construction), whereas auxiliaries and modals take a `VP` argument:

```
(29)  a. (NP (NP the policy)
             (S (NP-SBJ (-NONE- *))
                (VP (TO to)
                    (VP (VB seduce)
                        (NP socialist nations)
                        (PP-CLR into the capitalist sphere)))))
      b. (S (NP-SBJ-1 The banks)
            (VP (VBD stopped)
                (S (NP-SBJ (-NONE- *-1))
                   (VP (VBG promoting)
                       (NP the packages))))))
      c. (S (NP-SBJ Both sides)
            (VP (VBP are)
                (VP (VBG taking)
                    (NP action))))
```

Any `S` with a null subject receives the category $\mathsf{S\backslash NP}$ (with appropriate verbal feature), which is also the category assigned to VP arguments. The feature [b] is used for bare infinitives (`VB`). It is also used for imperatives and the subjunctive, since they cannot be distinguished from bare infinitives. Present participles carry the feature [ng].

3.5.3 Passive

The surface subject of a passive sentence is co-indexed with a * null element which appears in the direct object position after the past participle, for example:

```
(S (NP-SBJ-1 John)
   (VP (VBD was)
        (VP (VBN hit)
            (NP (-NONE- *-1))
            (PP (IN by)
                (NP-LGS a ball)))))
```

In this case, the null element does not indicate an argument which should be reflected in the category of the participial. Instead, the correct lexical categories are (S[dcl]\NP)/(S[pss]\NP) for *was* and S[pss]\NP for *hit*.

The algorithm uses the presence of the * null element to distinguish past participles in passive verb phrases from past participles in active verb phrases such as the following example:

```
(30) (S (NP-SBJ-1 John)
        (VP (VBZ has)
            (VP (VBN hit)
                (NP the ball))))
```

In this case, *hit* has the category (S[pt]\NP)/NP.

We analyze the *by*-PP in passive verb phrases as an adjunct rather than as an argument of the passive participle. The reason for this is that the *by*-PP is optional, so the category (S[pss]\NP)/PP[by] does not have to be acquired for all passive verbs.

In the case of verbs like *pay for*, which subcategorize for a PP, the null element appears within the PP:

```
(S (NP-SBJ-30 (PRP$ Its)
     (NN budget))
   (VP (VBZ is)
       (VP (VBN paid)
           (PP-CLR (IN for)
                   (NP (-NONE- *-30)))
           (PP (IN by)
                   (NP-LGS (PRP you)))))
           (. .)))
```

In this example, the correct lexical categories are (S[dcl]\NP)/(S[pss]\NP) for *is*, (S[pss]\NP)/(PP/NP) for *paid*, and PP/NP for *for*.

Note that the preposition has its ordinary category PP/NP, and that the past participle subcategorizes for the preposition alone, instead of the saturated PP. This means that in passive verb phrases with passive traces in PPs in object position, the passive trace must be taken into account as an argument to the preposition, but it must also be percolated up to the PP level in order to assign the correct category to the past participle.

3.5.4 Control and raising

In the Treebank, raising and subject control both have a co-indexed *-trace in the subject position of the embedded clause, for instance:

```
(31) a. (S (NP-SBJ-1 Mr. Stronach)
           (VP (VBZ wants)
               (S (NP-SBJ (-NONE- *-1))
                  (VP (TO to)
                      (VP (VB resume)
                          (NP a more influential role))))))
```

b.
```
(S (NP-SBJ-1 Every Japanese under 40)
   (VP (VBZ seems)
       (S (NP-SBJ (-NONE- *-1))
          (VP (TO to)
              (VP (VB be)
                  (ADJP-PRD fluent in Beatles lyrics))))))
```

Since an S with an empty subject NP has category S\NP, we obtain the correct lexical category (S[dcl]\NP)/(S[to]\NP) for both *seems* and *wants*.

In the case of object control (32a), the controlled object appears as a separate argument to the verb and is co-indexed with a *-trace in subject position of the complement. Object raising 32b is analyzed as a small clause in which the verb takes a sentential complement:

(32) a.
```
(S (NP-TMP Last week)
   (, ,)
   (NP-SBJ housing lobbies)
   (VP (VBD persuaded)
       (NP-1 Congress)
       (S (NP-SBJ (-NONE- *-1))
          (VP (TO to)
              (VP raise the ceiling to $124,875))))
```

b.
```
(S (NP-SBJ Czechoslovakia)
   (ADVP-TMP still)
   (VP (VBZ wants)
       (S (NP-SBJ-1 the dam)
          (VP (TO to)
              (VP (VB be)
                  (VP (VBN built)
                      (NP (-NONE- *-1))))
```

However, as explained in more detail in section 3.5.5, the CCG account of these constructions (Steedman, 1996) assumes that both elements of the small clause are arguments of the verb, and we modify the tree so that we obtain the same lexical category (((S[dcl]\NP)/(S[to]\NP))/NP) for both verbs.

3.5.5 Small clauses

The Treebank adopts a small clause analysis for constructions such as the following:

(33) a.
```
(S (NP-SBJ The country)
    (VP (VBZ wants)
        (S (NP-SBJ-2  half the debt)
           (VP (VBN forgiven)
               (NP (-NONE- *-2))))))
```

b.
```
(S (NP-SBJ that volume)
   (VP (VBZ makes)
       (S  (NP-SBJ it)
           (NP-PRD (NP the largest supplier))
                (PP of original TV programming)
                (PP-LOC in Europe))))
```

If these verbs occur in the passive, they are analyzed as taking a small clause, with a passive NP null element as subject (see section 3.5.3):

```
(34)  (S (NP-SBJ-1 The refund pool)
         (VP (MD may) (RB not)
             (VP (VB be)
                (VP (VBN held)
                    (S (NP-SBJ (-NONE- *-1))
                      (NP-PRD (NN hostage)))))))
```

Steedman (1996) argues against this kind of small clause analysis on the basis of extractions like *"what does the country want forgiven"*, which suggest that these cases should rather be treated as involving two complements. We therefore eliminate the small clause, and transform the trees such that the verb takes the children of the small clause as complements. This corresponds to the following analyses:

```
(35)  a. (S (NP-SBJ the country)
            (VP (VBZ wants)
                (NP half the debt)
                (VP (VBN forgiven)
                    (NP (-NONE- *-2)))))
      b. (S (NP-SBJ that volume)
            (VP (VBZ makes)
                (NP it)
                (NP-PRD (NP the largest supplier)
                        (PP of original TV programming)
                        (PP-LOC in Europe))))
      c. (S (NP-SBJ-1 The refund pool)
            (VP (MD may) (RB not)
                (VP (VB be)
                    (VP (VBN held)
                        (NP (-NONE- *-1))
                        (NP-PRD hostage)))))
```

The other case where small clauses are used in the Treebank includes absolute *with* constructions, which are analyzed as adverbial SBAR:

```
(36)  a. (S (SBAR-ADV (IN With)
                      (S (NP-SBJ the limit)
                         (PP-PRD in effect)))
            (, ,)
            (NP-SBJ members)
            (VP would be able to execute trades at the limit price)
            (. .))
      b. (S (SBAR-ADV (IN Though)
                      (S (NP-SBJ (-NONE- *-1))
                         (ADJP-PRD (JJ modest))))
                 (, ,)
                 (NP-SBJ-1 the change)
                 (VP (VBZ reaches)
                     (PP-LOC-CLR  beyond the oil patch)
                     (, ,)
                     (ADVP too))
                 (. .)))
```

We use the same approach for these cases, and assume that the subordinating conjunction (*with* or *though*, in these examples), takes the individual constituents in the small clause as complements. In the examples above, this gives the following lexical categories:

(37) a. *with* ⊢ ((S/S)/PP)/NP

b. *though* ⊢ (S/S)/(S[adj]\NP)

3.5.6 Yes-no questions

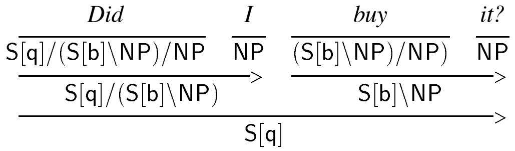
The Treebank gives yes-no questions (labelled `SQ`) a flat structure, in which the inverted auxiliary/copula is a sister of both the subject `NP` and the `VP` or predicative `NP`:

(38) a.
```
(SQ (VBZ Did)
    (NP-SBJ I)
    (VP (VB buy)
        (NP it)
    (. ?))
```

b.
```
(SQ (VBZ Is)
    (NP-SBJ this)
    (NP-PRD the future of chamber music)
    (. ?)))
```

This flat structure corresponds to the categorial analysis of yes-no questions given eg. in Carpenter (1992). Here is the CCG derivation corresponding to example 38a:

(39)



3.5.7 Inversion

The Treebank analyzes as `SINV` a number of constructions that do not follow the normal SVO pattern of English:

- verb fronting
- predicative fronting
- locative inversion
- negative inversion
- elliptical inversion
- conditional inversion
- direct speech inversion

Verb fronting The Treebank analyses predicate fronting and verb fronting in terms of movement, similar to topicalization:

```
(SINV (VP-TPC-1 (VBG Following)
                (NP the feminist and population-control lead))
      (VP (VBZ has)
          (VP (VBN been)
              (VP (-NONE- *T*-1))))
      (NP-SBJ a generally bovine press)
      (. .))
```

However, the noun phrase labelled as `NP-SBJ` here is actually in accusative case:

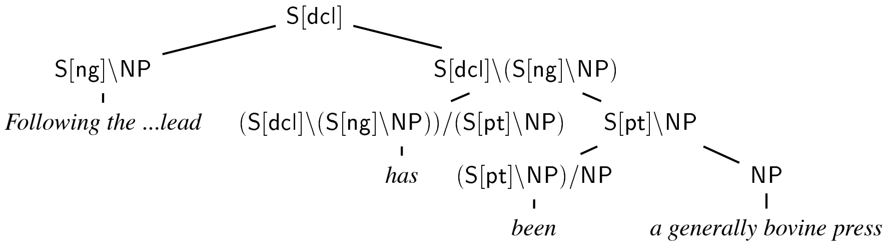
(40) a. **Following the lead has been I.*

b. *Following the lead has been me*.

Therefore, we assume that the NP is an object, and the verb phrase subject. The trees are changed so that the noun phrase appears as an object to the innermost verb. The VP-trace is removed:

```
(SINV (VP-TPC-1 (VBG Following)
                (NP the feminist and population-control lead))
      (VP (VBZ has)
          (VP (VBN been)
              (NP a generally bovine press)))
      (. .))
```

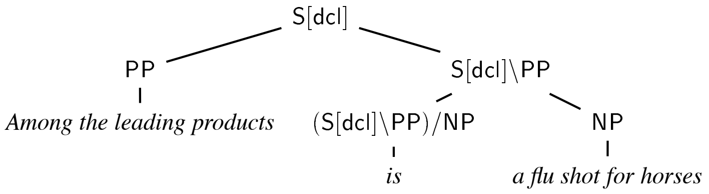
Here is the CCG derivation:



Predicative inversion Predicative inversion around the copula is also analyzed as topicalization within an `SINV`:

```
(SINV (PP-LOC-PRD-TPC-1 (IN Among)
                        (NP the leading products)))
      (VP (VBZ is)
          (PP-LOC-PRD (-NONE- *T*-1)))
      (NP-SBJ (NP a flu shot)
              (PP for horses))
      (. .))
```

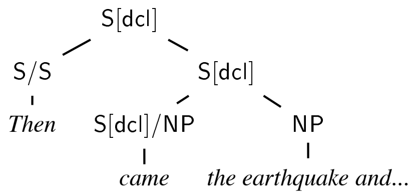
Both the fronted adverb and the subject are arguments of the verb. Since this construction occurs mainly in present tense or simple past, a modification of the tree like in the case of verb fronting was not implemented:



Locative inversion With certain verbs of movement the subject can appear to the right of the verb if there is a modifier to the left of the verb:

```
(SINV (ADVP-TMP Then)
      (VP came)
      (NP-SBJ (NP the earthquake))
              (CC and)
              (NP (NP a damaging delay)
                  (PP of 11 days)))
      (. .))
```

In those cases, both the adverb and subject should be arguments of the verb. This would require a modification to the adjunct-complement distinction which has not yet been implemented. Therefore, *came* receives category S[dcl]/NP in the above sentence:



Negative inversion In negative inversion, the negative element is often part of a sentential modifier, and hence difficult to identify:

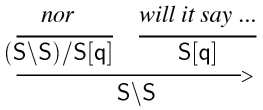
```
(SINV (ADVP-TMP Never once)
      (VBD did)
      (NP-SBJ (PRP she))
      (VP (VP gasp for air)
          (CC or)
          (VP mop her brow))
      (. .))
```

The only case where negative inversion can be identified easily is after *nor*:

```
(S (S (NP-SBJ Michael)
      (VP won't confirm the identities of any Easy Egg customers))
   (, ,)
   (CC nor)
   (SINV (MD will)
         (NP-SBJ (PRP it))
         (VP (VB say)
             (NP much of anything else)))
   (. .)))
```

The word order following the negative item is like a yes-no question. This could be captured in CCG by letting the negative item subcategorize for a yes-no question:

(41)



This analysis was only implemented for negative inversion with *nor*, since in other cases, the negative item is difficult to detect given the Treebank markup.

Elliptical inversion Certain words, such as *so*, *than*, *as*, take an inverted elliptical sentence as argument. The resulting constituent is either a declarative sentence or an adjunct:

(42) *I attend, and so does a television crew from New York City.*

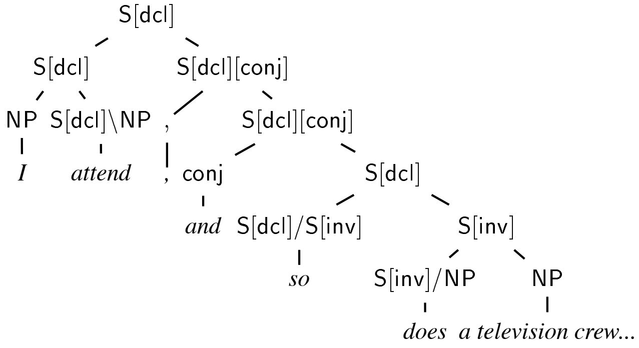
The Treebank annotation is flat:

```
(SINV (ADVP-PRD-TPC-1 (RB so))
      (VP (VBZ does)
          (ADVP-PRD (-NONE- *T*-1)))
      (NP-SBJ a television crew from New York City))
```

However, we change this to include a special kind of inverted sentence:

```
(SINV (ADVP-PRD-TPC-1 (RB so))
      (SINVERTED (VP (VBZ does)
                     (ADVP-PRD (-NONE- *T*-1)))
                 (NP-SBJ a television crew from New York City)))
```

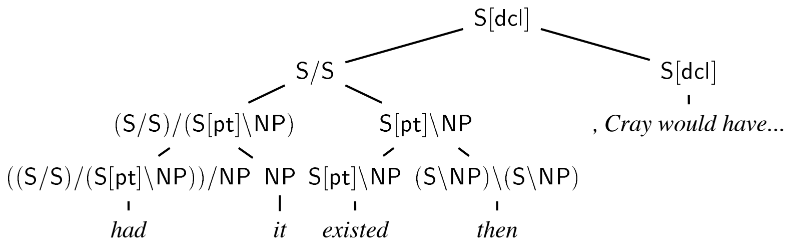
This inverted elliptical sentence consists of a do-form, auxiliary or modal, followed by the subject. A special feature [inv] on the CCG categories is used to distinguish those constructions from ordinary declaratives:



Conditional inversion Certain auxiliaries and modals (*had*, *should*, etc.) can introduce an inverted sentence which acts as a conditional clause:

```
(S (PP On the other hand)
   (, ,)
   (SBAR-ADV (SINV (VBD had)
                   (NP-SBJ (PRP it))
                   (VP (VBN existed)
                       (ADVP-TMP (RB then)))))
        (, ,)
        (NP-SBJ Cray Computer)
        (VP (MD would)
            (VP (VB have)
                (VP (VBN incurred)
                    (NP a $20.5 million loss))))
        (. .)))
```

Since this construction is triggered by auxiliaries, we treat the auxiliary as head of the adjunct, leading to the following derivation:

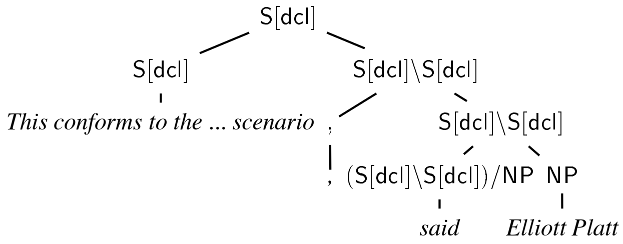


Recall that the features in our grammar do not distinguish conditional from indicative mood; therefore we cannot encode the fact that this construction requires the conditional in the main clause.

Direct speech inversion The Treebank analyses sentences where direct speech occurs before the verb as `SINV`:

```
(SINV (‘‘ ‘‘)
      (S-TPC-1 This conforms to the ‘soft landing’ scenario)
      (’’ ’’)
      (VP (VBD said)
           (S (-NONE- *T*-1)))
      (NP-SBJ Elliott Platt)
      (. .)))
```

However, since this is clearly a lexical phenomenon that can only occur with verbs of direct speech, the corresponding CCG analysis treats this word order as being determined in the lexicon, rather than being triggered by a topicalization rule:



In fact, verbs which take direct speech as argument can have at least three categories, one for the untopicalized case, and two for the sentence-topicalized cases with and without subject inversion:

(43) a. *Elliott Platt said: “This conforms to the ‘soft landing’ scenario”*
b. *“This conforms to the ‘soft landing’ scenario”, Elliott Platt said.*
c. *“This conforms to the ‘soft landing’ scenario”, said Elliott Platt.*

As explained below (section 3.12), quotation marks are cut out by the translation algorithm.

3.5.8 Ellipsis

The Treebank uses the null element *?* as placeholder “for a missing predicate or a piece thereof” (Marcus *et al.*, 1993). *?* is used for VP ellipsis, and can also occur in conjunction with a VP pro-form *do*, or in comparatives:

(44) a.
```
(S (NP-SBJ No one)
        (VP (MD can)
            (VP (VB say)
                (SBAR (-NONE- *?*))))
        (. .))
```
b.
```
(S (NP-SBJ The total of 18 deaths)
   (VP (VBD was)
       (ADJP-PRD (ADJP far higher))
            (SBAR (IN than)
                  (S (NP-SBJ (-NONE- *))
                     (VP (VBN expected)
                         (S (-NONE- *?*))))))))
```
c.
```
(S (S (NP-SBJ You)
      (RB either)
      (VP (VBP believe)
          (SBAR Seymour can do it again)))
   (CC or)
   (S (NP-SBJ you)
      (VP (VBP do)
          (RB n't)
          (VP (-NONE- *?*)))))
```

Although the *?* null element indicates a semantic argument of the head of the VP under which it appears (e.g. of *expected* or *do* in the examples above), we do not reflect this argument in the syntactic category of the heads. We follow the analysis of VP ellipsis under conjunction given in Steedman (2000), which argues that both conjuncts in examples such as 44c are complete sentences. Therefore, the syntactic category of *do* is $S\backslash NP$, not $(S\backslash NP)/VP$. See section 3.5.7 for our treatment of elliptical constructions after words such as *as* or *than*.

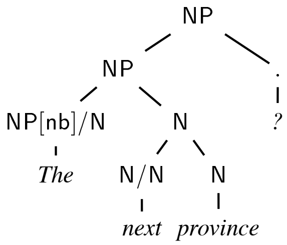
3.5.9 Fragments in the Treebank

The Treebank labels any constituent for which no proper analysis could be given as FRAG (for fragment). This can be an entire tree, or part of another constituent:

(45) a.
```
(FRAG (NP The next province) (. ?))
```
b.
```
(SBARQ (WRB how) (RP about) (FRAG (NP the New Guinea Fund)) (. ?)
```
c.
```
(TOP (FRAG (FRAG (IN If)
                 (RB not)
                 (NP-LOC (NNP Chicago))
                 (, ,)
                 (RB then)
                 (PP-LOC (IN in) (NP New York)))
           (: ;)
           (FRAG (IN if)
                 (RB not)
                 (NP-LOC (the U.S.))
                 (, ,)
                 (RB then)
                 (NP-LOC overseas)))
           (. .)))
```

These constituents are often difficult to analyze, and the annotation is not very consistent. Appendix B lists some heuristics that were used to infer additional structure.

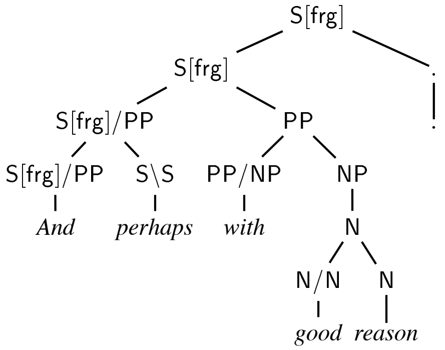
If an node is labelled FRAG, and there is only one daughter (plus, if this node is the root of the entire tree, optionally, a punctuation mark), we treat the tree as if it was labelled with the label of its daughter:



If the first daughter of a FRAG is a conjunction, we analyze it as head. The entire constituent is a fragment (S[frg]):

```
(TOP (FRAG (CC And)
           (ADVP (RB perhaps))
           (PP (IN with)
               (NP (JJ good)
                   (NN reason)))
           (. .)))
```

Here is the CCG translation:



3.6 Basic noun phrase structure

3.6.1 Noun phrases and nouns

The Treebank assumes a flat internal structure with no separate noun level:

(46) `(NP (DT the) (NNP Dutch) (VBG publishing) (NN group))`

We distinguishe noun phrases (NP) from nouns, (N). Determiners, such as *the*, are functions from nouns to noun phrases: *the* ⊢ NP[nb]/N[3]. Other prenominal modifiers are functions from nouns to nouns, eg. *Dutch* ⊢ N/N. Nouns are not marked for number. The grammar includes a unary projection from NP to N, so that verb or other elements do not specify the bareness of their noun phrase arguments. Pronouns (*this*, *you*) and quantifying noun phrases (*something*, *anything*, *nothing*, *somebody*, etc.) are NPs.

[3] For historical reasons, determiners carry a "non-bare" feature [nb]

3.6.2 Compound nouns

Compound nouns in the Treebank have in general no internal structure:

```
(NP (JJR lower) (NN rate) (NNS increases))
(NP (JJ only) (JJ French) (NN history)  (NNS questions))
```

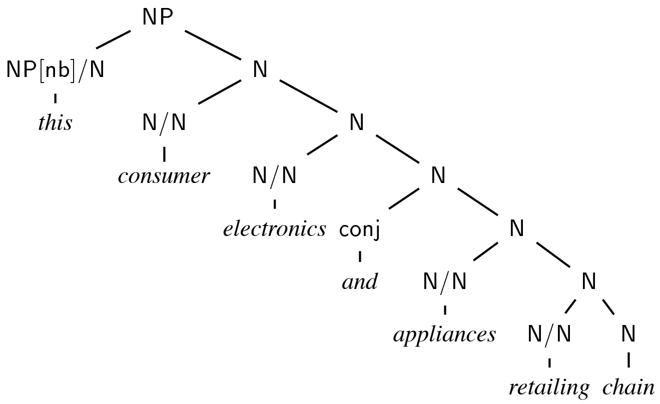
In order to obtain the correct analysis, manual re-annotation would be required, which was not deemed feasible within the current project. Therefore, compound nouns are simply translated into strictly right-branching trees. This is particularly problematic for conjunctions within compound nouns:

```
(NP (DT this)
    (NN consumer)
    (NNS electronics)
    (CC and)
    (NNS appliances)
    (NN retailing)
    (NN chain))
```

We include the following non-standard rule in our grammar:

(47) conj N ⇒ N

This rule allows us to translate the above tree as follows:



3.6.3 Appositives

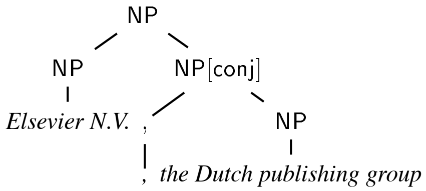
Appositives (48) are not indicated as such in the Treebank:

```
(48)  (NP (NP Elsevier N.V.)
          (, ,)
          (NP the Dutch publishing group))

      (NP-SBJ-1 (NP his son)
                (, ,)
                (NP Zwelakhe)
                (, ,)
                (NP a newspaper editor)
                (, ,))
```

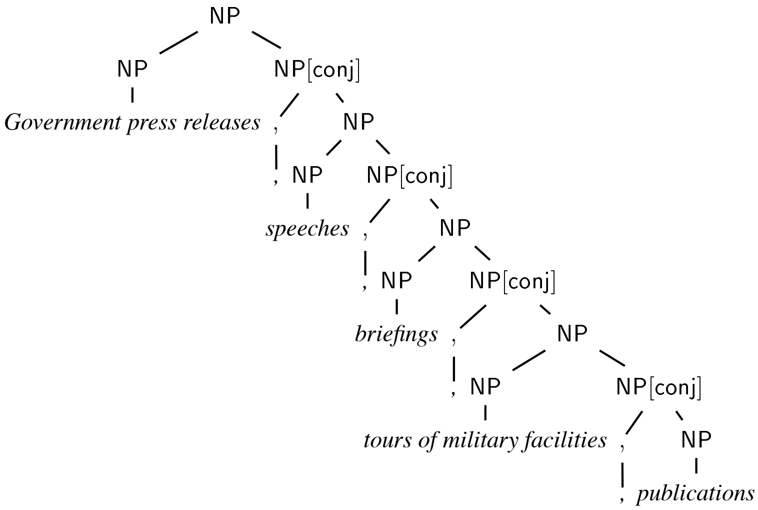
Their markup is indistinguishable from that of NP coordinations such as example (49)

(49)
```
(NP-SBJ (NP Government press releases)
        (, ,)
        (NP speeches)
        (, ,)
        (NP briefings)
        (, ,)
        (NP tours of military facilities)
        (, ,)
        (NP publications))
```

Therefore, our CCG does not distinguish between NP appositives and NP coordination, even though appositives should really be analyzed as modifiers:



Here is the "true" coordinate construction:



This leads to a reduction of ambiguity in the grammar, but is semantically not desirable.

3.6.4 Possessive *'s*

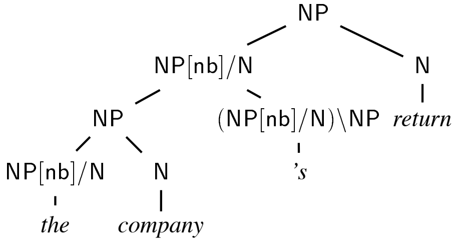
Possessive NPs in the Treebank have a flat structure in which the possessive *'s* or *'* is the last daughter:

```
(NP (NP (DT the) (NN company) (POS 's))
    (NN return))
```

In CCG, the possessive *'s* and *'* are analyzed as functors from NPs to determiners. In order to obtain this analysis, we insert a new constituent `POSSDT`, which consists of the innermost `NP` and the possessive *'s* itself, so that the final structure (before assigning categories) is as follows:

```
(NP (POSSDT (NP the company)
            (POS 's))
    (NOUN return))
```

Within a POSSDT, the POS is head, and the NP its argument; otherwise a POSSDT is like an ordinary determiner.



3.6.5 Quantifier phrases

The Treebank assumes a flat internal structure for QPs ("quantifier phrases"):

```
(QP (IN between) (CD 3) (NN %) (CC and) (CD 5) (NN %))
```

We use a number of heuristics to identify the internal structure of these constituents, for instance to detect conjuncts and prepositions. The above example is then re-bracketed as

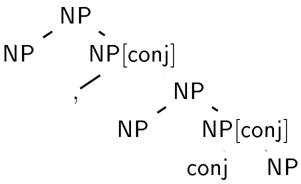
```
(QP (IN between)
    (QP (QP (CD 3) (NN %))
        (CC and)
        (QP (CD 5) (NN %)))
```

See appendix B.7 for details.

3.7 Other constructions

3.7.1 Coordination

Note that in the case of coordination (or lists), there is more than one head child, and we need to modify the translation algorithm accordingly. Coordinate constructions (or lists) are transformed into strictly right-branching binary trees. We assume that in those cases, all head children have the same category (see section 3.7.2 for a discussion of unlike coordinate phrases UCP). The inserted dummy nodes receive the same category as both conjuncts, but additionally carry a feature [conj]. A node with category X[conj] always has its head daughter (with category X) on the right, and the left daughter is either a conjunction (conj), or a punctuation mark, such as a comma or semicolon:



Therefore, coordination is captured in CCGbank by the following binary rule schemata:

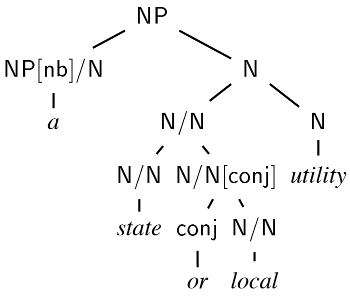
(50) a. conj X ⇒ X[conj]
b. , X ⇒ X[conj]
c. X X[conj] ⇒ X

3.7.2 ”Unlike” coordinate phrases

The Treebank annotates as UCP (“unlike coordinate phrase”) coordinate constructions where the conjuncts do not belong to the same syntactic category, eg:

```
(NP (DT a)
    (UCP (NN state)
         (CC or)
         (JJ local))
    (NN utility))
```

A UCP as head of a modifier is not a problem, because the CCG categories of modifiers only depend on the CCG category of the head:



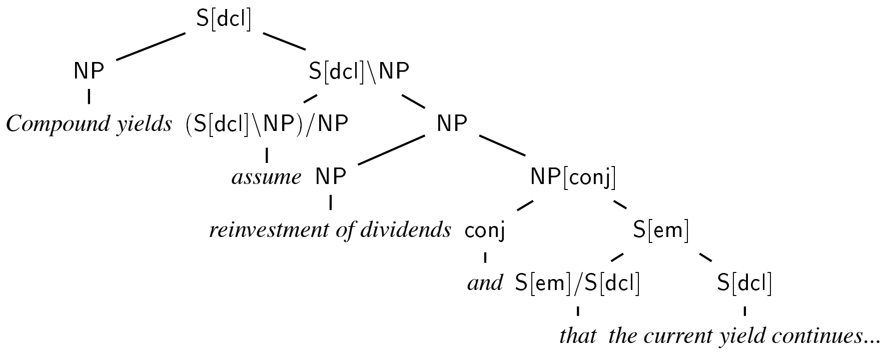
Such constructions only pose a problem for the translation algorithm if they occur as arguments, since the CCG category of arguments is determined by their Treebank label:

```
(S (NP-SBJ Compound yields)
   (VP (VBP assume)
       (UCP (NP reinvestment of dividends)
            (CC and)
            (SBAR (IN that)
                  (S (NP-SBJ the current yield)
                     (VP continues for a year)))))
   (. .))
```

In order to account for such cases, we modify the grammar slightly, and add special coordination rules of the following form:

(51) conj Y ⇒ X[conj]

This enables us to analyze the previous sentence:

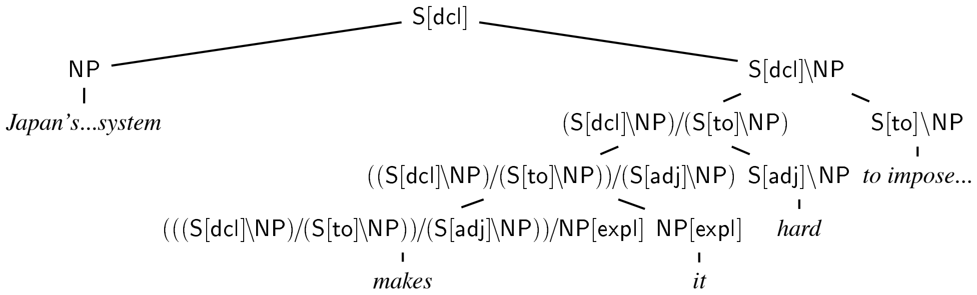


3.7.3 Expletive *it* and *there*

If *"it"* is annotated with an expletive trace `*EXPL*` or is the subject of a cleft sentence (see section 3.9.7), its category is NP[expl], and the verb subcategorizes for a NP[expl].

```
(TOP (S (NP-SBJ Japan 's management system)
        (VP (VBZ makes)
            (S (NP-SBJ (NP (PRP it))
                       (S (-NONE- *EXP*-1)))
               (ADJP-PRD (JJ hard))
               (S-1 (NP-SBJ (-NONE- *))
                    (VP to impose a single system corporatewide))))
        (. .)))
```

The `*EXPL*` trace is cut out, resulting in the following CCG derivation:[4]



Similarly, we use the POS-tag `EX` used for expletive *there* to obtain the feature NP[thr]. Lexical categories that take NP[expl] or NP[thr] as arguments do not co-index these with other arguments.

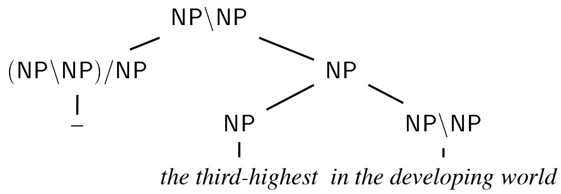
3.7.4 Parentheticals

The label `PRN` indicates a parenthetical element. A large number of these constituents constitute of a dash followed by one constituent or a constituent enclosed in parentheses, eg:

```
(PRN (: --)
     (NP (NP the third-highest)
         (PP-LOC (IN in)
                 (NP the developing world))))
```

[4] An alternative analysis might treat the to-infinitive as an argument of the adjective; however, in the Treebank analysis, the to-infinitive is a sister, not a daughter of the adjectival phrase. Therefore the present analysis is obtained.

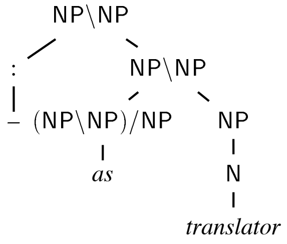
We treat opening parentheses, dashes or colons as functors which take the following constituent and yield a modifier:



This is a slightly over-simplified analysis, since it does not take into account that dashes are balanced (see eg. (Nunberg, 1990)). However, if the enclosed constituent is of a type that could be a modifier within the local tree under which the PRN appears, the enclosed constituent is the head of the parenthetical and receives the same category as if it did not appear underneath a PRN:

```
(PRN (: -)
     (PP (IN as)
         (NP (NN translator))))
```

Here, the PP is a modifier category; hence, the corresponding CCG derivation is as follows:



The dash is treated like an ordinary punctuation mark, and its category is determined from its part-of-speech tag (:).

3.7.5 Extraposition of appositives

Appositive noun phrases can be extraposed out of a sentence or verb phrase. However, in written English, a comma is required before or after the appositive:

(52) a. *No dummies,* the drivers pointed out they still had space.

b. Factory inventories fell 0.1% in September*, the first decline since February 1987*.

We analyze these constructions with special binary type changing rules that take into account that these appositives can only occur adjacent to commas:

$$\begin{array}{llll} \mathsf{NP} & , & \Rightarrow & \mathsf{S/S} \\ , & \mathsf{NP} & \Rightarrow & \mathsf{S\backslash S} \\ , & \mathsf{NP} & \Rightarrow & \mathsf{(S\backslash NP)\backslash(S\backslash NP)} \end{array}$$

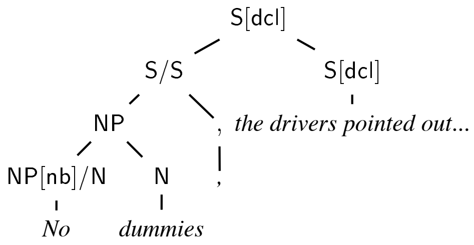
The Treebank analysis does not usually group the noun phrase and comma together, eg.:

```
(S (S-ADV (NP-SBJ (-NONE- *-1))
          (NP-PRD No dummies))
   (, ,)
   (NP-SBJ-1 the drivers)
   (VP pointed out they still had space ...).
```

Therefore, we insert a new constituent XNP which comprises the NP and the comma:

```
(S (XNP (S-ADV (NP-SBJ (-NONE- *-1))
         (NP-PRD No dummies))
   (, ,))
   (NP-SBJ-1 the drivers)
   (VP pointed out ...)).
```

This yields the following CCG analysis:



How these rules are acquired from the Treebank is explained in detail in appendix B.

3.7.6 Multi-word expressions

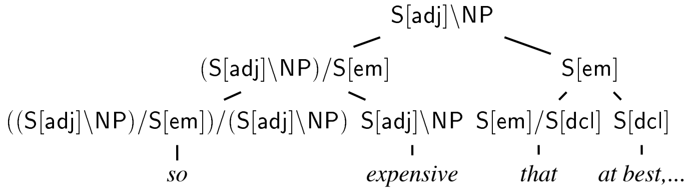
Multi-word expressions pose a particular problem for a lexicalist framework.With a few exceptions, explained below, an analysis for multi-word expressions is currently not attempted. This includes phrasal verbs, which are difficult to spot in the Treebank since particles can be found as PRT, ADVP-CLR and ADVP. Therefore, phrasal verbs do not subcategorize for particles in our grammar. An analysis was attempted for some common expressions that are either very frequent or where the multi-word expression has a different subcategorization behaviour from the head word of the expression.

because of...,instead of... We analyze *because* as the head of this expression. The prepositional phrase headed by *of* that follows it is its argument. When *instead of* appears within a PP, it is analyzed in the same way as *because of.*

as if..., as though Two special categories, S[as] and S[poss] are used for this construction. SBARs whose specifier is *if* and *though* have category S[poss] as a complement. SBARs whose specifier is *as* and which contain another SBAR with category S[poss] have category S[as] as complement.

so X that... In adjective phrases of the form *"so ADJ that..."*, we analyze *so* as a modifier of the adjective which also takes the embedded declarative as argument:

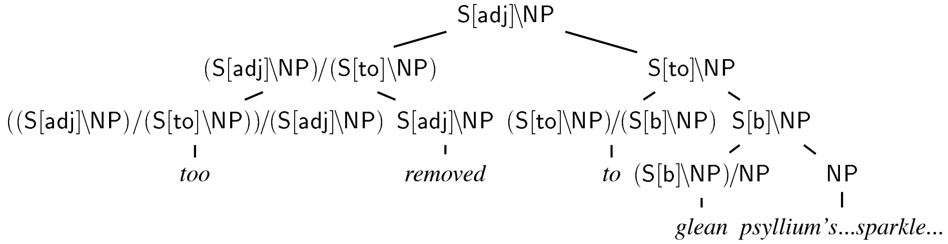
```
(ADJP-PRD (ADJP (RB so)
                (JJ expensive))
          (SBAR (IN that)
                (S (, ,)
                   (PP at best)
                   (, ,)
                   (NP-SBJ only the largest exchanges)
                   (VP can afford it))))
```



too ADJ to Similarly, constructions of the form *"too ADJ to"*, where the *to*-VP is a sister of *too* is changed to an analysis where *too* takes the *to*-VP as argument:

```
(ADJP (RB too)
      (VBN removed)
      (S (NP-SBJ (-NONE- *))
         (VP to glean psyllium 's new sparkle in the West)))
```

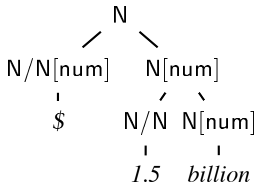
This is the resulting derivation:



This modification presumes that if the to-VP is an argument of the adjective (as in *"happy to do something."*), the to-VP is embedded within an ADJP, which in turn is modified by *"too"*.

at least/most X In this construction, we analyze *at* as taking a superlative adjective (with category S[asup]\NP) as argument.

Monetary expressions Amounts of currency (eg. *$ 1.5 billion*, *$ 3.85*) are among the most frequent type of multi-word expression in the Wall Street Journal. However, their internal structure is completely regular: it is always a currency symbol followed by a sequence of numbers (tagged CD), and syntactically, they behave like one item. In earlier versions of CCGbank, we replaced these expressions by a string DOLLARS. In the current release, we assume that the *$*-sign takes the numeral *1.5 billion* as argument. We assign a special category N[num] to *1.5 billion*, and assume that *billion* is the head, which is modified by *1.5*:



Dates In expressions such as *"Oct. 30, 1989"*, we assume that the name of the month (*Oct.*) takes the day of the month (*30*) as argument. We assign the category N[num] to *30*.

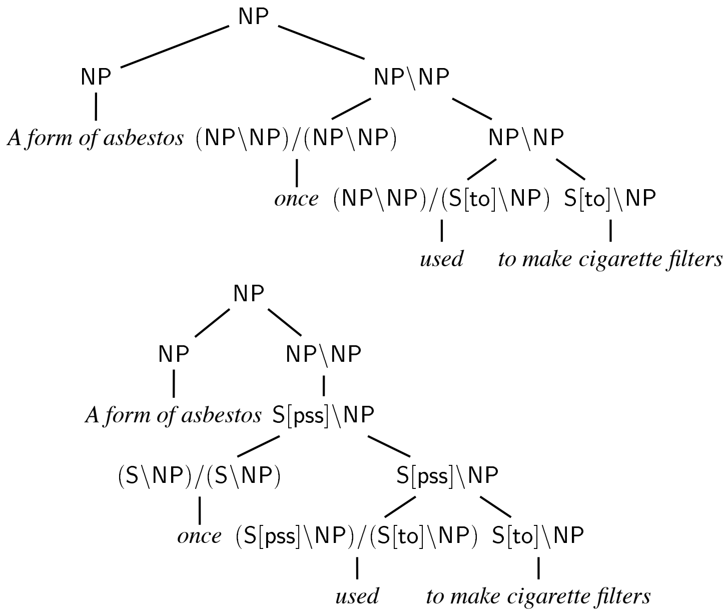


Figure 3.1: The effect of type-changing rules on lexical categories of adjuncts

3.8 Type-changing rules for clausal adjuncts

Figure 3.1 illustrates how the basic algorithm described in section 3.3 leads to a proliferation of adjunct categories. For example, a past participle such as *used* receives different categories depending on whether it occurs in a reduced relative or a main verb phrase. As a consequence, modifiers of *used* will also receive different categories depending on what occurrence of *used* they modify. This is undesirable, since we are only guaranteed to acquire a complete lexicon if we have seen participles (and all their possible modifiers) in all their possible surface positions. Similar regularities have been recognized and given a categorial analysis by Carpenter (1992), who advocates lexical rules to account for the use of predicatives as adjuncts.

It seems more economical to us to put such type-changing rules in the grammar. Such an approach has been taken by Aone and Wittenburg (1990) (also in a categorial framework) to encode morphological rules, and for reduced relatives and other syntactic constructions. Aone and Wittenburg show that these type-changing rules can be derived from zero morphemes in the grammar. Carpenter (1991, 1992) demonstrates that, in general, the inclusion of lexical rules can lead to a shift in generative power from context-free to recursively enumerable. However, this does not hold true if these lexical rules cannot operate on their own output and hence generate an infinite set of category types. Like Aone and Wittenburg, we only consider a finite number of instantiations of these type-changing rules, namely those which arise when we extend the category assignment procedure in the following way: for any sentential or verb phrase modifier (an adjunct with label S,SBAR with null complementizer, or VP) to which the original algorithm assigns category $X|X$, apply the following type-changing rule (given in bottom-up notation) in reverse:

(53) $S\$ \Rightarrow X|X$

where $S\$$ is the category that this constituent obtains if it is treated like a head node by the basic algorithm. $S\$$ has the appropriate verbal features, and can be $S\backslash NP$ or S/NP. Some of the most common type-changing rules are:

(54) a. $S[\mathrm{pss}]\backslash NP \Rightarrow NP\backslash NP$

"workers [exposed to it]"

b. S[adj]\NP ⇒ NP\NP
"a forum [likely to bring attention to the problem]"

c. S[ng]\NP ⇒ NP\NP
"signboards [advertising imported cigarettes]"

d. S[ng]\NP ⇒ (S\NP)\(S\NP)
"become chairman, [succeeding Ian Butler]"

e. S[dcl]/NP ⇒ NP\NP
"the millions of dollars [it generates]"

In written English, certain types of NP-extraposition require a comma before or after the extraposed noun phrase:

(55) *Factories booked \$236.74 billion in orders in September, [$_{NP}$ nearly the same as the \$236.79 billion in August]*

We make use of this fact in the following binary type-changing rules:

$$\begin{array}{llll} \mathsf{NP} & , & \Rightarrow & \mathsf{S/S} \\ , & \mathsf{NP} & \Rightarrow & \mathsf{S\backslash S} \\ , & \mathsf{NP} & \Rightarrow & \mathsf{(S\backslash NP)\backslash(S\backslash NP)} \end{array}$$

These rules are used whenever a NP-adjunct without functional tags, such as -TMP, appears at the periphery of a sentence or verb phrase and is immediately followed or preceded by a comma.

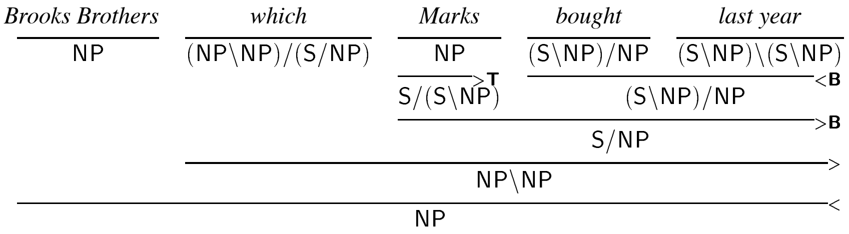
3.9 Long-range dependencies through extraction

The Treebank represents wh-questions, relative clauses, topicalization of complements, and *tough* movement in terms of movement. The "moved" constituent is co-indexed with a trace (*T*) which is inserted at the extraction site:

```
(NP-SBJ (NP Brooks Brothers))
        (, ,)
        (SBAR (WHNP-1 (WDT which))
              (S (NP-SBJ NNP Marks))
                 (VP (VBD bought)
                     (NP (-NONE- *T*-1))
                     (NP-TMP last year)))))
        (, ,))
```

CCG has a different, but similarly uniform treatment of these constructions. What in transformational terms is described as the moved constituent is analyzed in CCG as a functor over a sentence missing a complement. For instance, the relative pronoun in the following examples has the category (NP\NP)/(S/NP), while the verb *bought* maintains its respective canonical categories (S\NP)/NP :

(56)



CCG allows the subject noun phrase and the incomplete verb phrase to combine via type-raising and forward composition to form a constituent with the category S/NP, which can in turn be taken as an argument of the relative pronoun. As the relative clause itself is a noun phrase modifier, the relative pronoun has the category (NP\NP)/(S/NP). This treatment of "movement" in terms of functors over "incomplete" constituents allows CCG to keep the same category for the verb even when its arguments are extracted.

The *T* traces in the Treebank help us in two ways to obtain the correct categorial analysis: firstly, their presence indicates a complement which needs to be taken into account in order to assign the correct category to the verb, and secondly, we can use a mechanism very similar to slash-feature passing in GPSG (Gazdar *et al.*, 1985) to obtain the correct categories for the wh-word and the incomplete sentence.

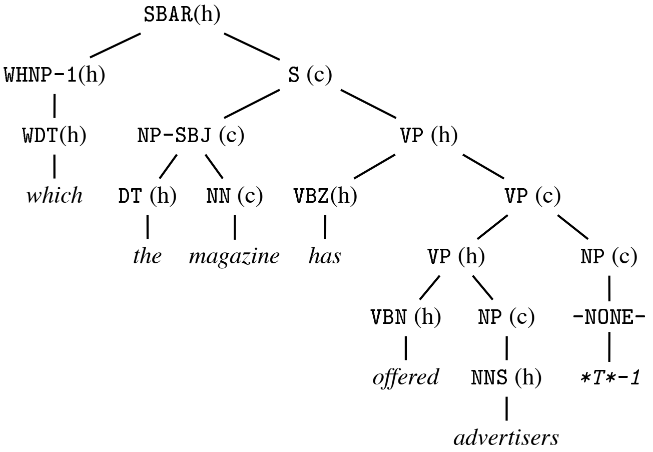
The following sections show our treatment of various constructions that involve long range dependencies arising through extraction, such as relative clauses, wh-questions, *tough* movement, topicalization, pied piping, subject extraction from embedded sentences and clefts. We use the example of relative clauses to explain in detail how the algorithm deals with *T* traces.

3.9.1 Relative clauses

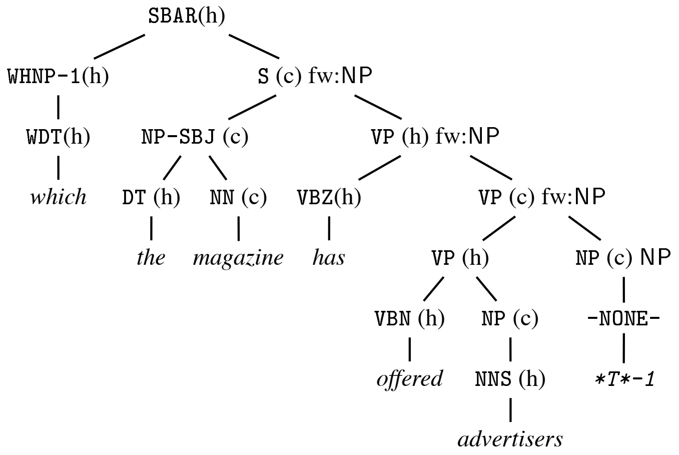
Here is the Treebank analysis of a relative clause:

```
(SBAR (WHNP-1 (WDT which))
  (S (NP-SBJ (DT the) (NN magazine))
     (VP (VBZ has)
         (VP (VBN offered)
            (NP (NNS advertisers))
            (NP (-NONE- *T*-1)))))))
```

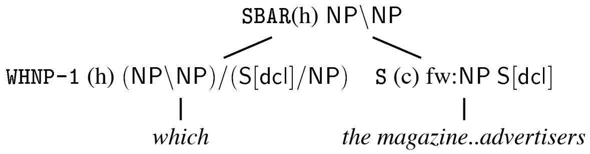
This is the same tree binarized and marked up with the constituent type:



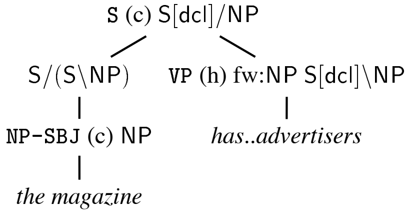
The NP-node with a *T* null element underneath it is a complement trace. The category of a complement trace (here NP) is determined (from its label) before the recursive category assignment described above. This category is then percolated up the head path of the parent of the trace to the next clausal projection. Depending on the position of the trace node within its local tree, this category is marked as a forward (fw) or backward (bw) argument:



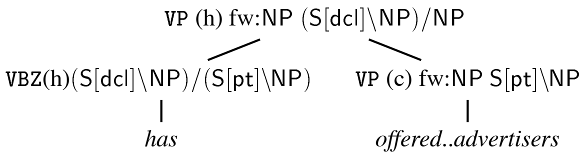
The S node is a complement, and receives category S[dcl]. However, since the S node also has a (forward) trace NP which is not percolated further to its parent, the head daughter (WHNP) subcategorizes for the unsaturated S[dcl]/NP. Assuming that the category of the SBAR parent is NP\NP, the WHNP has category (NP\NP)/(S[dcl]/NP):



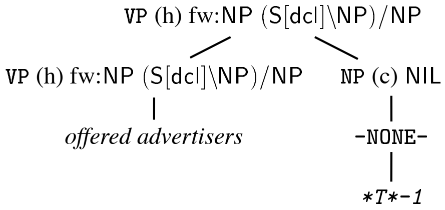
In the next step, the children of the S-node are assigned categories. The NP-SBJ is a backward complement with category NP, but since the S-node has a forward trace, it is type-raised to S/(S\NP). (Like adjunct categories, type-raised categories do not carry features). The forward trace is also appended to the category of the parent S node to yield S[dcl]/NP:



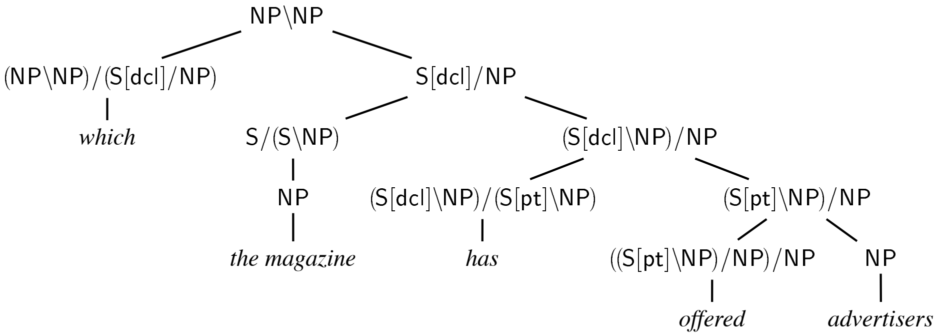
Then, categories are assigned to the children of the VP-node. The VP daughter is a complement with category S[pt]\NP. Since the forward trace is carried up to the parent VP, the head daughter VBZ subcategorizes for S[pt]\NP and hence receives category (S[dcl]\NP)/(S[pt]\NP). The forward trace is also appended to the category of the parent VP:



Going down to the children of the VP daughter, the NP trace is a forward argument. This yields the transitive verb category (S[pt]\NP)/NP for the VBN node. However, since the NP is a trace, this node will be cut out of the tree in a postprocessing stage. The category of the VP parent node is now also changed to take its forward trace into account.



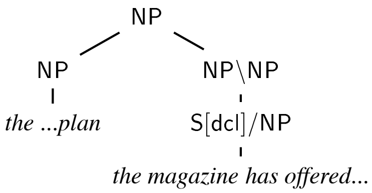
After cutting out the trace and all unary projections X ⇒ X, the subtree of the relative clause is as follows:



Reduced and zero relative clauses do not have a relative pronoun:

```
(NP-PRD (NP the second incentive plan))
        (SBAR (WHNP-1 (-NONE- 0))
              (S (NP-SBJ the magazine))
                 (VP (VBZ has)
                     (VP (VBN offered)
                         (NP advertisers)
                         (NP (-NONE- *T*-1))
                         (PP-TMP in three years))))
```

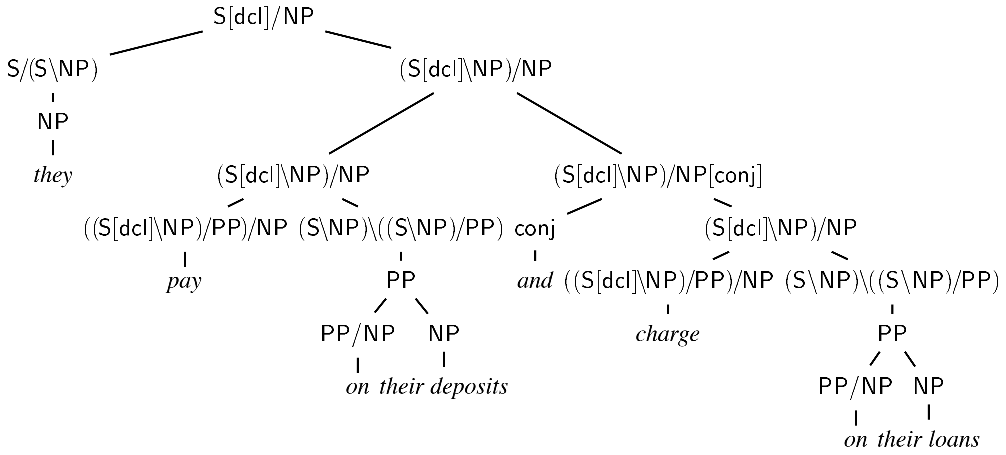
As already mentioned in section 3.8, we use unary type-changing rules (corresponding to the zero morphemes of Aone and Wittenburg (1990)) to account for this:



This algorithm works also if there is a coordinate structure within the relative clause such that there are two *T* traces:

```
(NP (NP the interest rates))
    (SBAR (WHNP-1 (-NONE- 0))
          (S (NP-SBJ (PRP they))
(VP (VP (VBP pay)
        (NP (-NONE- *T*-1))
        (PP-CLR (IN on)
                (NP their deposits))))
    (CC and)
    (VP (VB charge)
        (NP (-NONE- *T*-1))
        (PP-CLR (IN on)
                (NP their loans)))))))
```

This results in the following CCG derivation for the relative clause:

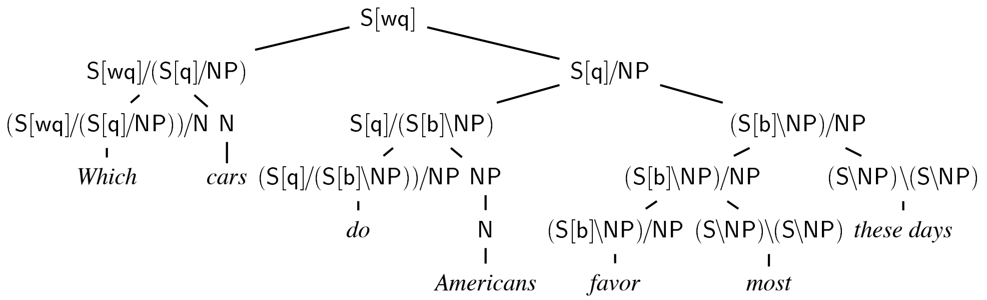


3.9.2 Wh-questions

T-traces are also used for wh-questions (labelled SBARQ):

```
(TOP (SBARQ (WHNP-1 (WDT Which)
                    (NNS cars))
            (SQ (VBP do)
                (NP-SBJ (NNP Americans))
                (VP (VB favor)
                    (NP (-NONE- *T*-1))
                    (ADVP (RBS most))
                    (NP-TMP (DT these)
                            (NNS days))))
            (. ?)))
```

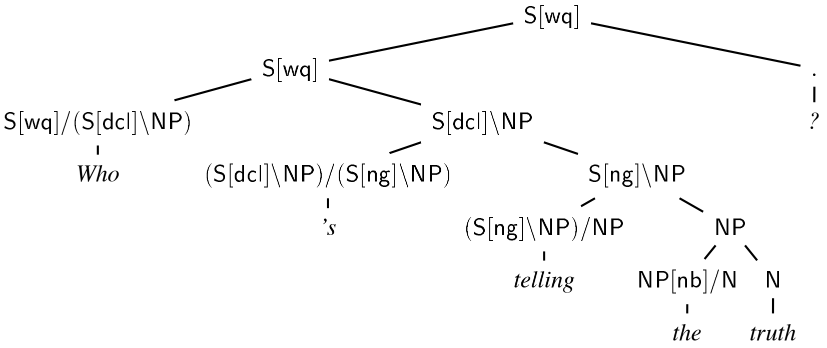
The Treebank analyses wh-questions as SBARQ, with the wh-word in specifier position. It assumes a further level of structure, SQ, which includes the rest of the question. By percolating the *T*-trace up to the SQ-level, we obtain the desired CCG analysis:



However, in subject questions, such as the following example, the following two kinds of analyses can be found:[5]

```
(57)  a. (SBARQ (WHNP-1 (WP Who))
               (SQ (VBZ 's)
                   (NP-SBJ (-NONE- *T*-1))
                   (VP (VBG telling)
                       (NP (DT the)
                           (NN truth))))
               (. ?))
      b. (SBARQ (WHNP-1 (WP Who))
               (S (NP-SBJ (-NONE- *T*-1))
                  (VP (VBZ 's)
                      (VP (VBG telling)
                          (NP (DT the)
                              (NN truth)))))
               (. ?))
```

For our purpose, a sentence with a trace in subject position is the same as a verb phrase; and the CCG analysis assumes in fact the the wh-word subcategorizes for a declarative verb phrase:



However, in order to obtain the correct CCG derivation from the first Treebank analysis, the inverted subject trace underneath the SQ has to be cut out, and the label SQ has to be changed to VP:

```
(SBARQ (WHNP-1 (WP Who))
       (VP (VBZ 's)
           (VP (VBG telling)
               (NP (DT the)
                   (NN truth))))
       (. ?))
```

[5] Erratum: In Hockenmaier *et al.* (2004), we used an example where the trace was in the wrong place.

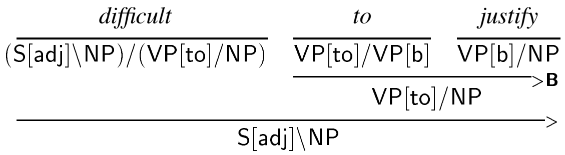
3.9.3 *Tough* movement

Tough movement is also annotated using *T*-traces:

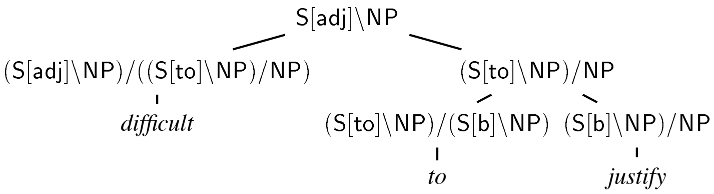
```
(S (NP-SBJ (PRP It))
   (VP (VBZ is)
       (ADJP-PRD (JJ difficult)
                 (SBAR (WHNP-1 (-NONE- 0))
                       (S (NP-SBJ (-NONE- *))
                          (VP (TO to)
                              (VP (VB justify)
                                  (NP (-NONE- *T*-1))))))))
```

The adjective phrase has the following categorial analysis (Steedman, 1996):

(58)



We obtain this analysis from the Treebank by percolating the forward NP slash category to the SBAR-level. As explained in section 3.5.2, the empty subject in infinitival verb phrases is cut out.

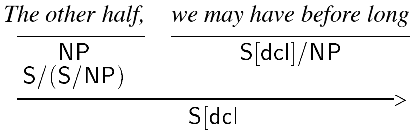


3.9.4 Topicalization

We assume that English has the following schema of a non order-preserving type-raising rule (Steedman, 1987) to account for topicalization of noun phrases, adjective phrases and prepositional phrases:

(59) $\mathsf{X} \Rightarrow \mathsf{S/(S\backslash X)}$
with $\mathsf{X} \in \{\mathsf{NP}, \mathsf{PP}, \mathsf{S[adj]\backslash NP}\}$

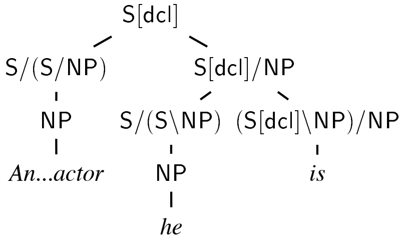
(60)



The Penn Treebank uses *T*-traces also for topicalization. If a constituent is topicalized, or fronted, it receives the tag -TPC, and is placed at the top level of the sentence. A co-indexed *T*-trace is inserted at the canonical position of that constituent:

```
(TOP (S (NP-TPC-1 An excellent environmental actor))
        (NP-SBJ (PRP he))
        (VP (VBZ is)
            (NP-PRD (-NONE- *T*-1)))
        (. .)))
```

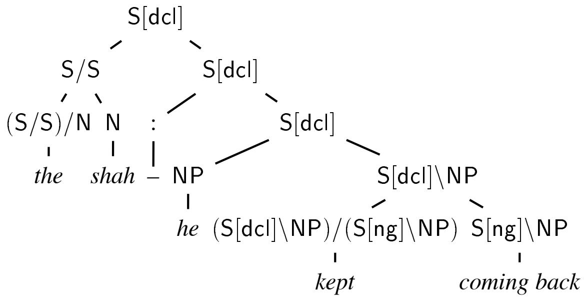
We stipulate that categories of the form S/(S\X) which can only be derived by non-order-preserving type-raising, can be assigned by the parser to any constituent of category X in sentence initial position.[6] Therefore, this category need not appear in the lexicon. Topicalised noun phrases (NP-TPC) receive the category NP, but in order to assign the correct category to the verb, an NP with tag -TPC is not considered a complement:



If there is a resumptive pronoun (that is, the fronting is an instance of left-dislocation), there is no coreference between the fronted element and the pronoun:

```
(S (NP-TPC the shah)
   (: --)
   (NP-SBJ-1 he)
   (VP (VBD kept)
       (S (NP-SBJ (-NONE- *-1))
          (VP coming back))))
```

In these cases, we obtain the correct lexical entry for the verb in the same manner, since it suffices to treat the (NP-TPC) as adjunct:



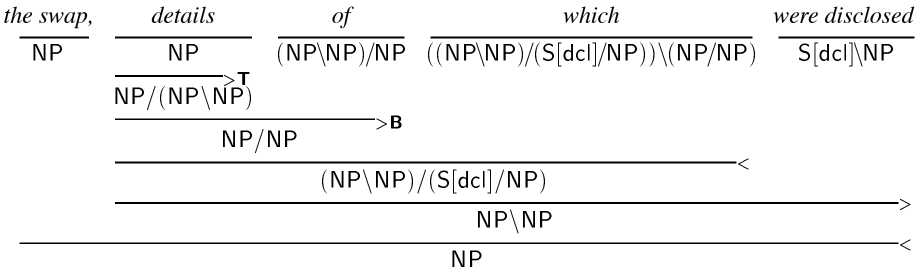
See 3.5.7 for the treatment of S-TPC and VP-TPC.

3.9.5 Pied piping

We follow the analysis of Steedman (1996) for pied piping:

[6] This assumption is not implemented in the parser or the probability model presented in the following chapters.

(61)



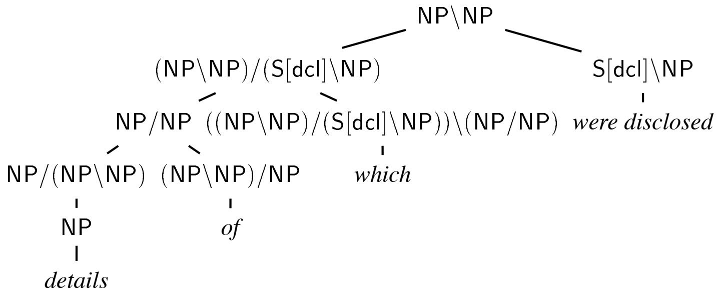
In the corresponding Treebank tree the preposition and relative pronoun form a PP which modifies the head noun.

```
(NP-SBJ (NP the swap)
        (, ,)
        (SBAR (WHNP-2 (WHNP (NNS details))
                      (WHPP (IN of) (WHNP (WDT which))))
              (S (NP-SBJ-1 (-NONE- *T*-2))
                 (VP were disclosed))))
```

However, in the CCG analysis, the preposition *"of"* combines first with the noun phrase *"details"*, and the relative pronoun *"which"* takes this constituent as argument. Therefore, a new constituent `piedPipeNP` is inserted, which consists of the noun phrase and preposition:

```
(NP-SBJ (NP the swap)
        (, ,)
        (SBAR (WHNP (piedPipeNP (WHNP (NNS details))
                                (IN of))
                    (WHNP (WDT which)))
              (S (NP-SBJ-1 (-NONE- *T*-2))
                 (VP were disclosed))))
```

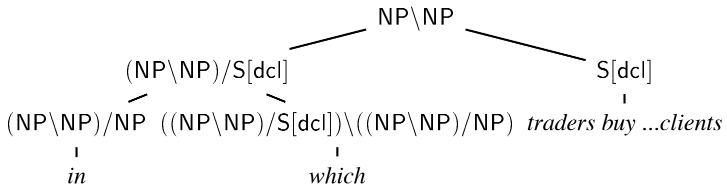
The preposition is an adjunct of the noun phrase; however, a special rule during category assignment guarantees that it is assigned the ordinary category of an NP-modifying preposition, (NP\NP)/NP, and that the noun phrase is type-raised and composed with the preposition. This constituent is then treated as an argument of the relative pronoun.



Similarly, there are cases of pied piping with only the PP:

```
(NP (NP commodities markets)
    (, ,)
    (SBAR (WHPP (IN in)
                (WHNP (WDT which)))
      (S (NP-SBJ traders)
         (VP buy and sell both for their own account and for clients))))
```

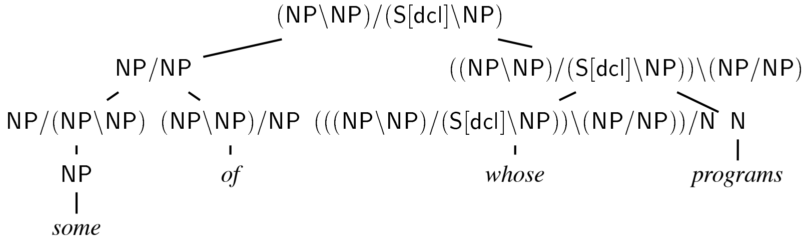
Here, we leave the tree as it is, since its constituent structure already corresponds to the CCG analysis. In this case the `WHNP` is head. The preposition is an argument and receives the NP-modifying category (NP\NP)/NP:



In the case of pied piping with *"whose"*, the relative pronoun also takes a noun as argument:

```
(NP-PRD (NP Ruth Messinger)
   (, ,)
   (NP a Manhattan city councilwoman)
   (, ,)
   (SBAR (WHNP-1 (NP (DT some))
              (WHPP (IN of)
                    (NP (WHNP (WP$ whose)
                        (NNS programs))
              (, ,)
              (ADJP such as commercial rent control)
              (, ,))))
          (S (NP-SBJ (-NONE- *T*-1))
             (VP have made their way into Mr. Dinkins 's position papers)
```

This is the CCG derivation:



3.9.6 Subject extraction from embedded sentences

Steedman (1996) assumes that the verb which takes a bare sentential complement from which a subject is extracted has category ((S\NP)/NP)/(S\NP). This category is only required when the subject is extracted.

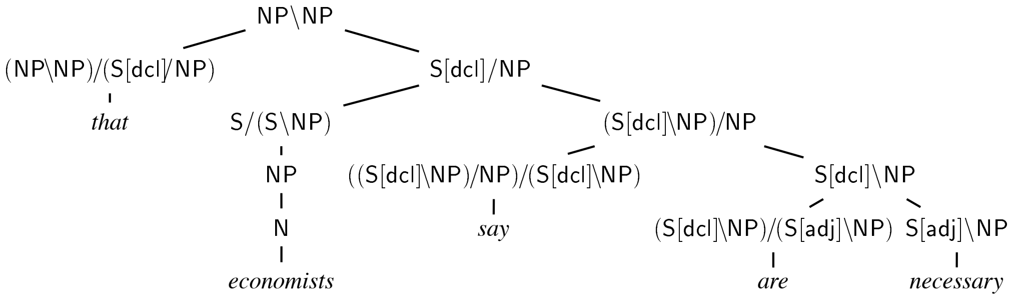
This constituent structure does not correspond to the Treebank analysis, where subject traces are in the ordinary position:

```
(NP (NP the sort)
    (PP of measures)
    (SBAR  (WHNP-1 (IN that))
           (S (NP-SBJ (NNS economists))
              (VP (VBP say)
                  (SBAR (-NONE- 0)
                        (S (NP-SBJ (-NONE- *T*-1))
                           (VP (VBP are)
                               (ADJP-PRD necessary))))))))
```

However, in order to obtain the desired analysis, we assume that the verb takes the VP and the NP argument in reversed order. Therefore, the tree is changed as follows:

```
(NP (NP the sort)
    (PP of measures)
    (SBAR  (WHNP-1 (IN that))
           (S (NP-SBJ (NNS economists))
              (VP (VBP say)
                  (VP (VBP are)
                       (ADJP-PRD (JJ necessary)))
                  (NP-SBJ (-NONE- *T*-1))))))
```

This then leads to the following CCG analysis:

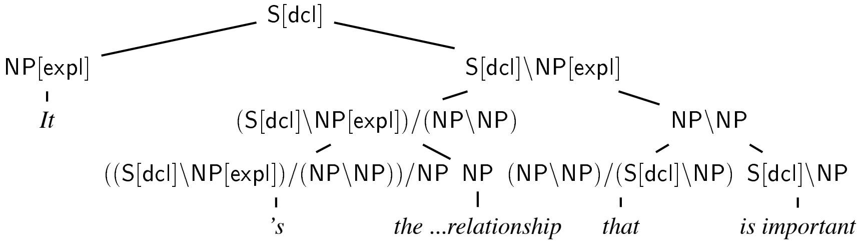


3.9.7 Clefts

Clefts are annotated as S-CLF in the Treebank:

```
(TOP (S-CLF (NP-SBJ (PRP It))
            (VP (VBZ 's)
                (NP-PRD the total relationship)
                (SBAR (WHNP-2 (WDT that))
                      (S (NP-SBJ (-NONE- *T*-2))
                         (VP (VBZ is)
                             (ADJP-PRD (JJ important))))))
            (. .)))
```

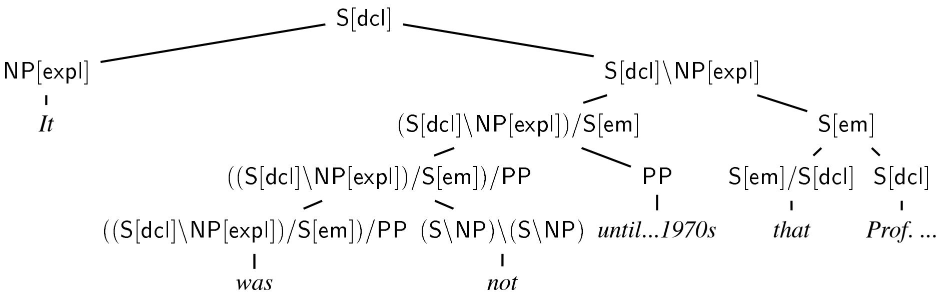
Although there is no *EXP*-trace, all subject noun phrases under S-CLF are assigned the category NP[expl]. We treat the SBAR as argument of the verb. If the focus of the cleft is a NP, as in the above example, its category is that of a noun phrase modifier NP\NP:



If the focus of the cleft is an adverb, the SBAR is simply an embedded declarative, eg:

```
(TOP (S-CLF (NP-SBJ (PRP It))
            (VP (VBD was)
                (RB not)
                (PP-PRD until the early 1970s)
                (SBAR (IN that)
                      (S (NP-SBJ-2 Prof. Whittington and two graduate students)
                         (VP began to publish ...))))
            (. .)))
```

Here is the corresponding CCG translation:

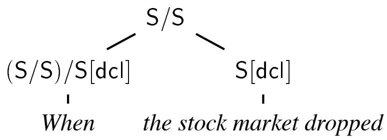


3.9.8 Extraction of adjuncts

T-traces can also stand for an adjunct (here indicated by the label ADVP-TMP):

```
(TOP (S (SBAR-TMP (WHADVP-1 (WRB When))
                  (S (NP-SBJ the stock market)
                     (VP (VBD dropped)
                         (ADVP-TMP (-NONE- *T*-1)))))
        (S (NP-SBJ the Mexico fund)
           (VP plunged about 18%))
        (. .)
```

Adjunct traces do not indicate missing complements, and are therefore simply ignored by the translation procedure.



3.9.9 Heavy NP shift

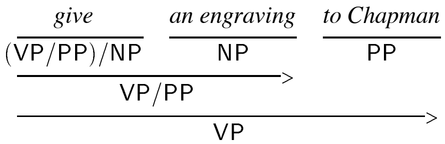
In English, some noun phrase arguments can be shifted to the end of the sentence if they become too "heavy":

(62) a. *give an engraving to Chapman*
b. *give to Chapman an engraving by Rembrandt*

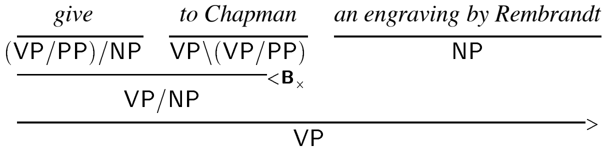
This kind of construction has been studied extensively by Ross (1967).

In order to provide an analysis according to which *give* has the same lexical category in the second case, Steedman (1996) uses backward-crossing composition:

(63) a.



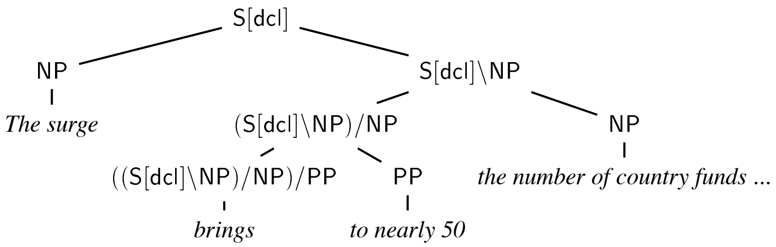
b.



In the Penn Treebank, heavy NP shift is not marked:

```
(S (NP-SBJ The surge))
   (VP (VBZ brings)
       (PP-CLR to nearly 50)
       (NP (NP the number)
           (PP (IN of)
               (NP country funds
                   that are or soon will be listed in New York or London))))
        (. .)))
```

From this tree, the following CCG translation is obtained, which does not conform to Steedman's analysis:



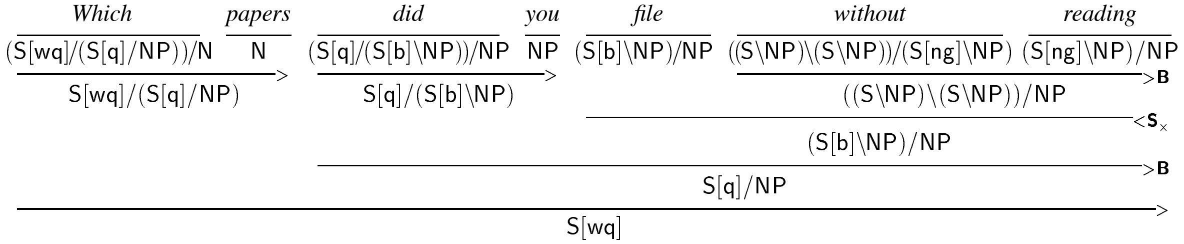
Backward crossing composition is also used in Steedman (1996, 2000) to account for certain preposition stranding phenomena in English. However, in its unrestricted form, this rule leads to overgeneralization. In order to avoid this overgeneralization, Steedman advocates the use of two features, *SHIFT* and *ANT*, on the arguments of lexical functor categories. At present, such features are not induced from the Penn Treebank. The grammar which underlies CCGbank only consists of particular rule instantiations, and might therefore not be prone to the overgeneration problem which motivated Steedman's features. We leave the question of whether such features would be necessary for a CCG that consists only of the rule instantiations found in a corpus, and how they could be obtained, open to further investigation.

3.9.10 Parasitic gaps

Parasitic gaps that arise through extraction, such as the following example, do not seem to occur in the Penn Treebank:

(64) *Which papers$_i$ did you file* t$_i$ *without reading* t$_i$*?*

In CCG, this construction is analyzed with the combinatory rule of substitution (Steedman, 1996):



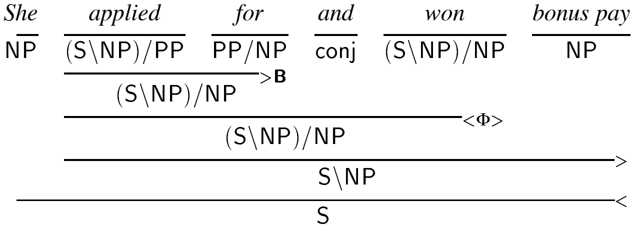
If such a construction did occur in the Penn Treebank, and both traces were in the right locations, our algorithm would be able to obtain the correct analysis. In section 3.10.2, we discuss the related case of right node raising parasitic gaps, which do occur in the Treebank. Since we obtain the correct derivation (with substitution) in these cases, the grammar that is implicit in CCGbank predicts the existence of parasitic gaps that arise through extraction.

3.10 Long-range dependencies through coordination

3.10.1 Right node raising

Right node raising of complements Right node raising constructions such as (65) can be analyzed in CCG using the same lexical categories as if the shared complement was present in both conjuncts (Steedman, 1996):

(65)



In order to assign the correct lexical categories to such sentences, we need to know where the canonical location of the shared complement is, ie. that the shared constituent is interpreted as a complement of both verbs, and that sentence (65) means the same as:

(66) *She applied for bonus pay and won bonus pay.*

The Treebank adopts an analysis of this construction in which the shared constituent is co-indexed with two *RNR*-traces that occur in the canonical position of the shared element:

```
((S (NP-SBJ She)
    (VP (VP (VBD applied)
        (PP-CLR (IN for)
           (NP (-NONE- *RNR*-1))))
      (CC and)
      (VP (VBD won)
        (NP (-NONE- *RNR*-1)))
      (NP-1  bonus pay)
      (PP-LOC (IN under) (NP the reform law)))
     (. .)))
```

In order to assign correct lexical categories to sentences with right node raising, we need to alter the translation algorithm slightly. The category assignment proceeds in three steps for sentences which contain `*RNR*`-traces:

1. When determining the constituent type of nodes: Identify all nodes which are co-indexed with `*RNR*`-traces (eg. `NP-1`). These constituents are neither heads, complements nor adjuncts, and hence will get ignored in the category assignment. `*RNR*`-traces themselves (or their maximal projections, here `NP`s) are treated like ordinary constituents, and thus they can be either heads, complements or adjuncts.
2. Assign categories to the nodes as before. Nodes which are co-indexed with `*RNR*`-traces (eg. the `NP-1` above) will be ignored because they are neither heads, complements nor adjuncts. `*RNR*`-traces themselves will receive the category of an ordinary constituent in this canonical position. If an `*RNR*`-trace is a complement, its category is percolated up to the topmost level of this coordination. This trace category is treated like a trace arising through extraction.
3. If the `*RNR*`-traces with the same index do not have the same category, this sentence cannot be processed, as the CCG analysis predicts that both constituents in canonical position have the same category.[7] Otherwise, copy the category of the `*RNR*`-traces, and assign it to the co-indexed node. Then assign categories in the usual top-down manner to the subtree beneath the co-indexed node.

Ignoring the co-indexed constituent *bonus pay* in the first pass guarantees that *applied* is assigned (S\NP)/PP, not (S\NP)/NP/PP. Considering the `*RNR*`-traces as ordinary constituents guarantees that *for* is assigned PP/NP, not PP, and *won* (S\NP)/NP, not S\NP.

Right node raising of heads In English it is also possible for two conjoined noun phrases to share the same head:

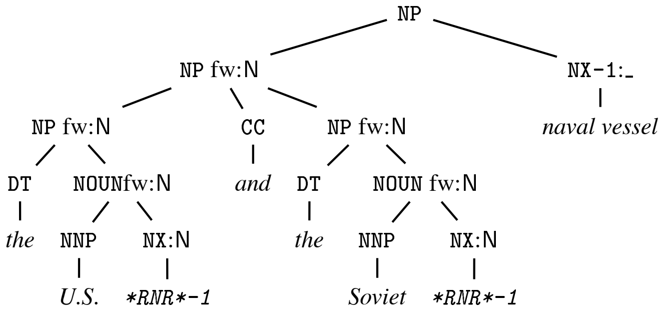
(67) *a U.S. and a Soviet naval vessel*

This is also annotated with `*RNR*`-traces.

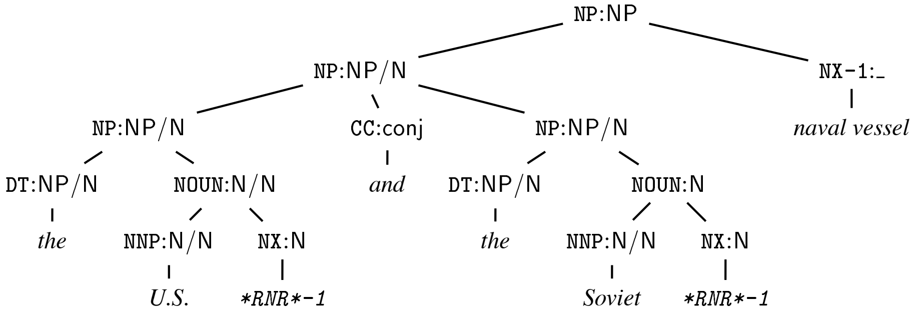
```
(NP (NP (DT a)
        (NNP U.S.)
        (NX (-NONE- *RNR*-1)))
    (CC and)
    (NP (DT a)
        (JJ Soviet)
        (NX (-NONE- *RNR*-1)))
    (NX-1 (JJ naval)
          (NN vessel)))
```

Our algorithm works just as well with this case: First, the category of the traces is determined and percolated up the tree (here shown with noun level inserted):

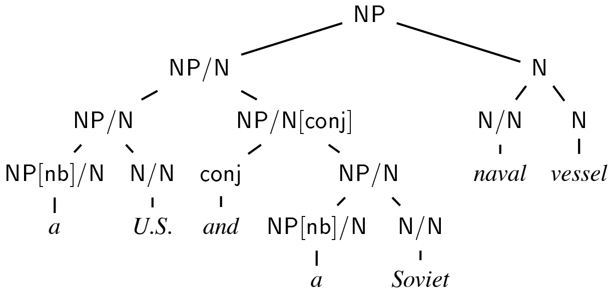
[7] This would arise if two `*RNR*`-trace complements did not have the same Treebank label, say if one was a `PP` and the other a `NP`, but in practice this does not happen. Note that in the adjunct case it is also conceivable that the categories of the two `*RNR*`-traces differ because they might be attached at different levels in the tree, but again, this does not seem to happen in practice.



During category assignment, the NX-1 is ignored. Categories are assigned to all other nodes, resulting in the following tree:



Then we assign NP to the shared constituent NX-1, and assign the corresponding categories to its daughters. Finally, the *RNR* traces are cut out:



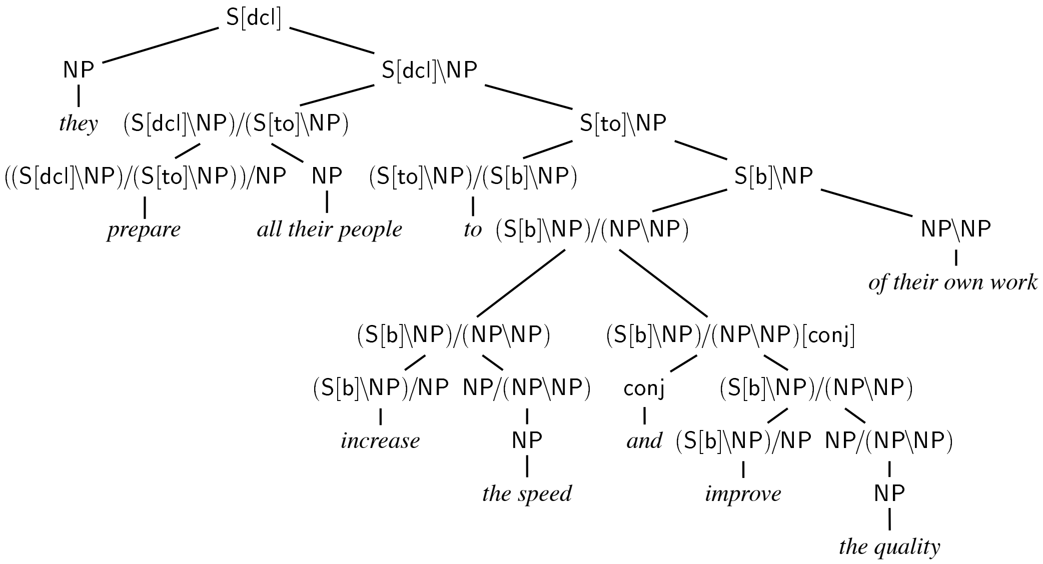
Right node raising of adjuncts When the shared constituent is an adjunct, the algorithm that we have just presented works if only the modified constituents are conjoined, although in that case it is not strictly necessary to use this two-pass procedure:

```
(NP (NP (NP president)
        (PP (-NONE- *RNR*-1)))
    (CC and)
    (NP (NP chief executive officer)
        (PP (-NONE- *RNR*-1)))
    (PP-1 of New England Electric))
```

However, the shared constituent can also modify constituents inside the conjuncts:

```
(S (CC And)
   (NP-SBJ they)
   (VP (VBP prepare)
       (S (NP-SBJ all their people)
          (VP (TO to)
              (VP (VP (VB increase)
                      (NP (NP the speed)
                          (PP (-NONE- *RNR*-1))))
                  (CC and)
                  (VP (VB improve)
                      (NP (NP the quality)
                          (PP (-NONE- *RNR*-1))))
                  (PP-1 of their own work)))))
    (. .)))
```

We stipulate that the NPs are type-raised to $\mathsf{NP/(NP\backslash NP)}$ before they are combined with the verbs by composition. This analysis requires the following modification to our algorithm: If a *RNR*-trace is an adjunct, we first determine the category of the constituent it modifies (this can only be done if this constituent is itself a complement, such as an NP, which is generally the case), which allows us to determine the category of the shared constituent (here $\mathsf{NP\backslash NP}$). The NPs which contain the *RNR*-traces are marked, so that their categories are type-raised to $\mathsf{NP/(NP\backslash NP)}$ during category assignment. The category of the shared adjunct, $\mathsf{NP\backslash NP}$, is also percolated up as a forward trace to the VPs. Finally, we obtain the correct derivation:

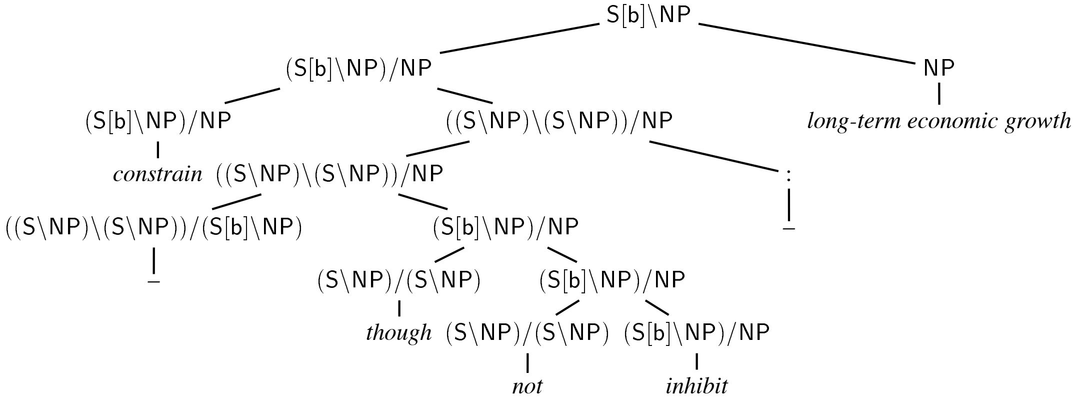


3.10.2 Right node raising parasitic gaps

Right node raising can also occur without coordination, and is then similar to parasitic gaps (section 3.9.10) that arise through extraction:

```
(S (NP-SBJ Hong Kong's uneasy relationship with China)
   (VP (MD will)
       (VP (VP (VB constrain)
               (NP (-NONE- *RNR*-1)))
           (PRN (: --)
     (IN though)
     (VP (RB not)
         (VB inhibit)
         (NP (-NONE- *RNR*-1)))
     (: --))
(NP-1 long-term economic growth))))
```

Like parasitic gaps that arise through extraction, CCG uses substitution (see section 2.3) to analyze this construction (Steedman, 1996). Our treatment of right node raising traces deals with this case correctly:



However, our algorithm is not able to obtain a CCG derivation for the following example, because substitution would only work if the *from*-PP is an argument of *go* and the *to*-PP is an adjunct. However, they both have the same function tag, -DIR:

```
(S-TPC-3 (NP-SBJ-1 They)
         (VP (MD 'll)
             (VP (VB go)
                 (PP-DIR (IN from)
                         (S-NOM (NP-SBJ (-NONE- *-1))
                                (VP (VBG being)
                                    (NP-PRD (NP one)
                                            (PP (IN of)
                                                (NP (DT the)
                                                    (ADJP most leveraged)
                                                    (NP (-NONE- *RNR*-2))))))))
                 (PP-DIR (TO to)
                         (NP (NP one)
                             (PP (IN of)
                                 (NP (DT the)
                                     (ADJP least leveraged)
                                     (NP (-NONE- *RNR*-2))))))
                 (NP-2 casino companies))))
```

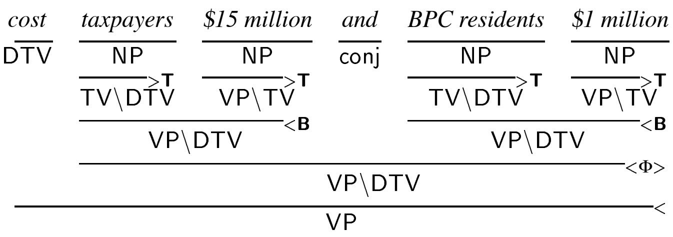
3.10.3 Argument cluster coordination

If two VPs with the same head are conjoined, the second verb can be omitted:

(68) *It could cost taxpayers $15 million and BPC residents $1 million.*

In the CCG account of this construction, *taxpayers $15 million* and *BPC residents $1 million* form constituents ("argument clusters"), which are then coordinated. Argument clusters are obtained by type-raising and composing the constituents of each argument cluster, such that the resulting category is a functor which takes a verb of the right category to its left to yield a verb phrase (cf. Steedman, 2000, chapter 7). Then the argument clusters are conjoined, and combine with the verb via function application:[8]

(69)



The Treebank encodes these constructions like a VP-coordination in which the second VP lacks a verb. Also, the daughters of the second conjunct are co-indexed with the corresponding elements in the first conjunct using the = notation (referred to in the Treebank manual (Bies *et al.*, 1995) as template gapping):

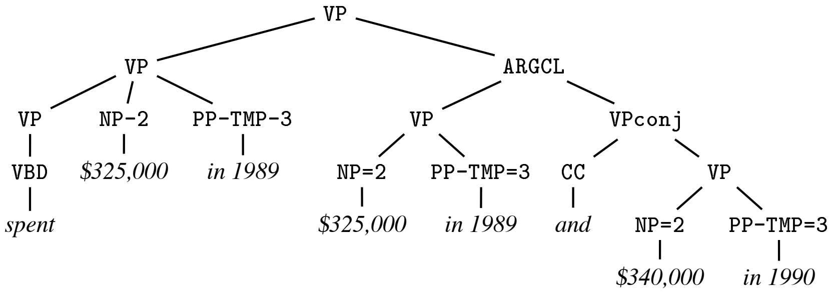
```
(S (NP-SBJ It)
   (VP (MD could)
       (VP (VP (VB cost)
               (NP-1 taxpayers)
               (NP-2 $ 15 million))
           (CC and)
           (VP (NP=1 BPC residents)
               (NP=2 $ 1 million))))
        (. .))
```

If the second VP contains constituents which do not correspond to constituents in the first VP, a null element (marked *NOT*) with the same label is inserted in the appropriate place in the first VP. This null element is co-indexed with the corresponding constituent in the second VP:

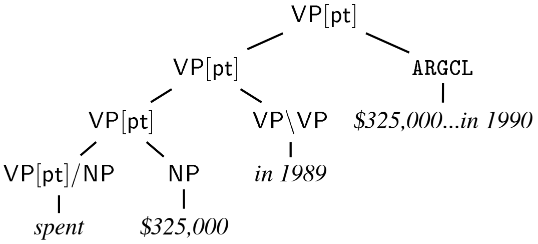
```
(S-ADV (NP-SBJ (-NONE- *-1))
       (VP (VP (VBG increasing)
               (PP-DIR-2 to 2.5 %)
               (PP-TMP-3 in February 1991)
               (ADVP-TMP-4 (-NONE- *NOT*)))
           (, ,)
           (CC and)
           (VP (PP-DIR=2 to 3 %)
               (PP-TMP=3 at six month intervals)
               (ADVP-TMP=4 thereafter))))
```

Since the Treebank constituent structure does not correspond to the CCG analysis, we need to transform the tree before we can translate it. We obtain the CCG constituent structure by creating a new node ARGCL consisting of a VP consisting of copies of the co-indexed elements of the first conjunct, the conjunction and the second conjunct (again using VP to abbreviate S\NP):

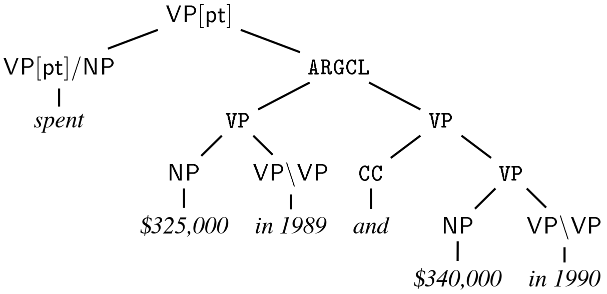
[8] We use the following abbreviations: VP for S\NP, TV for transitive (S\NP)/NP and DTV for ditransitive ((S\NP)/NP/NP)



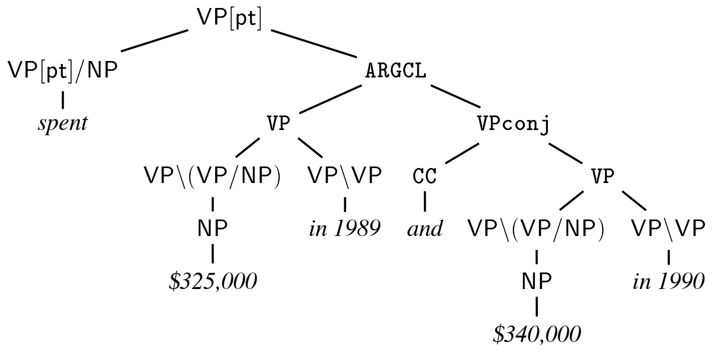
Category assignment now proceeds in two phases: first we assign categories in the normal fashion, ignoring the ARGCL tree:



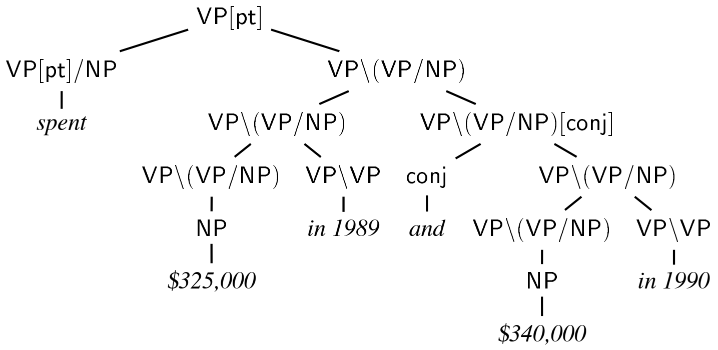
Then the constituents which are co-indexed with the elements in the first tree are assigned the same categories as their antecedents, and all nodes in the first conjunct apart from the verb are cut out:



Then category assignment proceeds underneath the co-indexed nodes (not shown here), as well as above them. Category assignment within an argument cluster is a bottom-up, right-to-left process. The leftmost node (PP-TMP=3) is an adjunct and does not need to be type-raised. However, the object noun phrase is backward-typeraised to VP\(VP/NP) (instantiating the T in the type-raising rule with the category of the parent of the argument cluster). If there was another object to the left of this noun phrase, with category Y, its type-raised category would be (VP/NP)\((VP/NP)/Y), instantiating the T with the argument category of the previous complement.



Then category assignment proceeds, bottom-up:

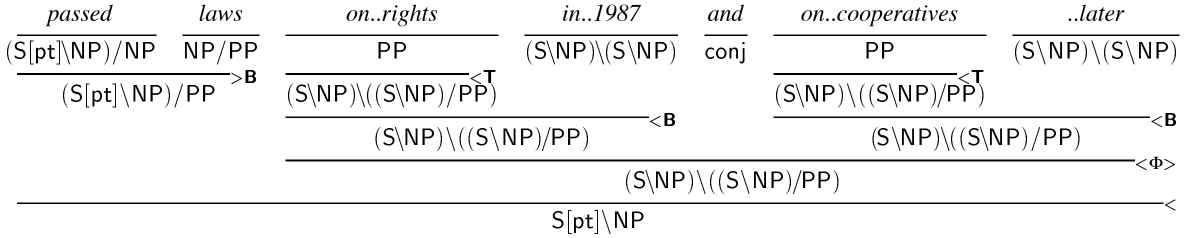


NOT-null elements, which are inserted in the first conjunct for elements in the second conjunct that do not have a counterpart in the first conjunct, are treated like ordinary constituents and are then later cut out.

This algorithm does not work for all sentences that are annotated with template-gapping ('=') in the Treebank. In particular, it does not work for sentential gapping (discussed below). It also does not work for cases such as the following, where the tree that contains the antecedents in the first conjunct is not isomorphic to the analysis of the second conjunct:

```
(SBAR-TMP (WHADVP-4 (WRB When))
     (S (NP-SBJ the Supreme Soviet)
        (VP (VP (VBD passed)
                (NP (NP laws)
                    (PP-2 on workers' rights))
                (PP-TMP-3 in May 1987))
            (CC and)
            (VP (PP=2 on self-managing cooperatives)
                (ADVP-TMP=3 a year later))
            (ADVP-TMP (-NONE- *T*-4)))))
```

CCG is perfectly capable of handling such cases. For example, here is the derivation of the above example:



Unfortunately, there is a lot of variability in the Treebank annotations, and we did not attempt to modify the algorithm to deal with each of these cases. This is a shame, because we believe that CCG's analyses of these constructions are one of its main advantages. Future issues of the treebank could consider providing a more consistent annotation of gapping, along lines such as the following:

```
(VP  (VP passed (NP laws *X*-1) *X*-2)
      (GAP (GAP (PP-1 on worker's rights)
                (PP-TMP-2 in May)
      (CC and)
      (GAP (GAP (PP-1 on cooperatives)
                (ADVP-TMP-2 a year later)))
```

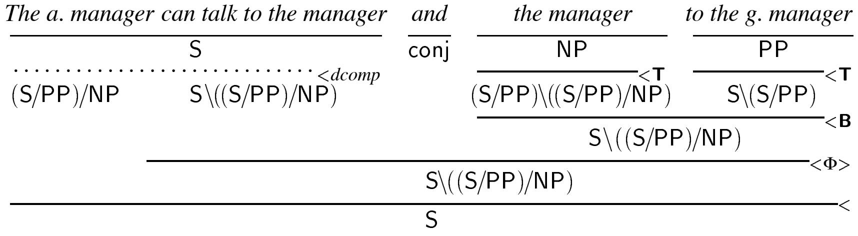
3.10.4 Gapping

The Treebank uses a similar annotation to argument cluster coordination for sentential gapping:

```
(S (S (NP-SBJ-1 Only the assistant manager)
      (VP (MD can)
          (VP (VB talk)
              (PP-CLR-2 (IN to)
                  (NP the manager)))))
   (CC and)
   (S (NP-SBJ=1 the manager)
      (PP-CLR=2 (TO to)
          (NP the general manager))))
```

This construction cannot be handled with the standard combinatory rules of CCG that are assumed for English. Instead, Steedman (2000) proposes an analysis of gapping that uses decomposition, which is a rule that is not based on combinatory logic. Decomposition allows a constituent to be split apart into two subparts, and is used to yield an analysis of gapping that is very similar to that of argument cluster coordination:

(70)



However, this analysis is problematic for our current purposes. Since the derivation is not a tree anymore, and since the decomposed constituents do not correspond to actual constituents in the surface string, this derivation is difficult to represent in a treebank.

Therefore, these sentences are currently not translated to CCG. However, we could use a set of special coordination rules of the form

(71) $\mathsf{S\ conj\ S\backslash X \Rightarrow S}$

where $\mathsf{S\backslash X}$ is the category of the second conjunct, eg. $\mathsf{S\backslash((S/PP)/NP)}$.

3.11 Other null elements in the Treebank

Besides the null elements discussed above, the Treebank contains further kinds of null elements, all of which we ignore when extracting a lexicon from the Treebank.

ICH ("Insert constituent here") is used for extraposition of modifiers. When there is intervening material between a modifier and the constituent it modifies, and if the intervening material causes a difference in attachment height, a *ICH* null element is inserted as adjunct to the modified constituent:

```
(72)  a. (S (NP-SBJ (NP Areas of the factory)
                    (SBAR (-NONE- *ICH*-2)))
            (VP (VBD were)
                (ADJP-PRD particularly dusty)
                (SBAR-2 where the crocidolite was used))
                (. .))
      b. (S (NP-SBJ There)
            (VP (VBZ 's)
                (NP-PRD (NP (DT an)
                            (NN understanding)
                            (SBAR (-NONE- *ICH*-1)))
                   (PP on the part of the US)
                   (SBAR-1 that Japan has to expand its functions in Asia))))
```

Like in the case of ellipsis, this is a case of a semantic dependency which should not be reflected in the syntactic category. Note that a constituent which is co-indexed with an *ICH* null element is not a complement. We therefore treat all constituents which are co-indexed with an *ICH* null element as adjuncts. In example (72a), for instance, *were* has category $\mathsf{(S[dcl]\backslash NP)/(S[adj]\backslash NP)}$, not $\mathsf{((S[dcl]\backslash NP)/S[emb])/(S[adj]\backslash NP)}$.

The null element *PPA* ("Permanent Predictable Ambiguity") is used for genuine attachment ambiguities:

```
(S (NP-SBJ He)
   (ADVP-TMP already)
   (VP (VBZ has)
       (VP (VBN finagled)
           (NP (NP a $ 2 billion loan)
               (PP (-NONE- *PPA*-1)))
           (PP-CLR-1 from the Japanese government))
        (. .))
```

Since the Treebank manual states that the actual constituent should be attached at the more likely attachement site, we chose to ignore any *PPA* null element.

Another null element, *U*, is used to "mark the interpreted position of a unit symbol" (Marcus *et al.*, 1993). This null element is ignored. See section 3.7.6 for the treatment of monetary expressions.

```
(NP (QP ($ $) (CD 1.5) (CD billion))
    (-NONE- *U*)))
```

3.12 Preprocessing the Treebank

The previous sections presented the general translation procedure. Some necessary preprocessing steps, such as the insertion of a noun level and our reanalysis of small clauses, were mentioned. However, other problems remain. For instance, usually `PP` daughters of `VP` are adjuncts, but the functional tag `-CLR` on a `PP` daughter can be used to indicate that there is a closer relationship between the PP and the verb. For instance, a `-CLR` tag on a PP can indicate that this is the second object of a ditransitive verb. However, it is well known that this tag has not been used in a very consistent manner, and there is an estimated POS tagging error rate of 3% (Ratnaparkhi, 1996). The translation algorithm is sensitive to these errors and inconsistencies: POS tagging errors can lead to incorrect categories or to incorrect features on verbal categories (eg. when a past participle is wrongly tagged as past tense); the omission or addition of functional tags causes errors in the complement-adjunct distinction; and certain types of coordinate constructions are not recognized as such if the bracketing is not correct. Additionally, the algorithm presented in section 3.3 requires the constituent structure of the phrase-structure tree before binarization to conform to the desired CCG analysis. Some systematic changes that are necessary have already been mentioned, but there are other cases that require modification. For instance, if the flat tree of a coordinate construction contains any adjuncts or arguments to the conjuncts, a separate level has to be inserted before binarization can proceed. This is true for constituents which are co-indexed with a right node raising trace (`*RNR*`), but there are also other cases which are either not explicitly marked as right node raising constructions, or where adjuncts to one of the conjuncts appear as sisters rather than daughters of the constituent they modify. Some heuristics were developed to correct POS tag errors that are likely to lead to errors in the translation process. For instance if a simple past tense form occurs in a verb phrase which itself is the daughter of a verb phrase whose head is an inflected verb, it is highly likely that it should be a past participle instead. Using the verb form itself and the surrounding context, we attempt to correct such errors automatically. Originally, quotation marks (“ and ”) also caused a number of problems for the translation algorithm, since they can appear in various positions in the tree and surface string. We therefore decided to eliminate them in the preprocessing step.

3.13 Generating the predicate-argument structure

In CCG, every syntactic derivation has a corresponding semantic interpretation. In order to obtain this interpretation, the lexicon has to pair words and their syntactic categories with an appropriate semantic type. We did not attempt to furnish our lexicon with fully-fledged predicate-argument structures or logical forms. Instead, we chose to use word-word dependencies which approximate the true semantic interpretation. Although simplistic in nature, these word-word dependencies include the non-local dependencies that arise through extraction, right node raising, argument cluster coordination, as well as control and raising. In order to capture these non-local dependencies, we co-index arguments of certain lexical functor categories, such as those of relative pronouns or control verbs. This mechanism is explained in section 2.5.6. A complete list of the lexical entries in sections 02-21 which use this co-indexation mechanism is given in appendix C. We believe that in practice this mechanism is largely correct, even though it is based on the (fundamentally flawed) assumption that all lexical categories that have the same syntactic type project the same dependencies. That this assumption is incorrect is most obvious in the case of control verbs: both *promise* and *persuade* have the syntactic category ((S\NP)/(S[to]\NP))/NP, yet for *persuade*, the subject of the *to*-VP should be co-indexed with the object NP, whereas for *promise*, it should be co-indexed with the subject. It may be possible to use the indices on the pro-null elements (`*-1`) in the Penn Treebank to recover this information; we leave this to future research.

3.14 Summary – the complete algorithm

Here is the complete translation algorithm, including the preprocessing step described in section 3.12 and the modifications necessary to deal with traces and argument clusters:

```
foreach tree $\tau$:
    preprocessTree($\tau$);
    preprocessArgumentClusters($\tau$);
    determineConstituentType($\tau$);
    makeBinary($\tau$);
    percolateTraces($\tau$);
    assignCategories($\tau$);
    treatArgumentClusters($\tau$);
    cutTracesAndUnaryRules($\tau$);
    verifiyDerivation($\tau$);
    getPredicateArgumentStructure($\tau$);
```

preprocessTree: Correct tagging errors, ensure the constituent structure conforms to the CCG analysis for noun phrases, coordinate constructions and small clauses, and eliminate quotes.

preprocessArgumentClusters: Create the constituent structure that corresponds to an argument cluster in argument cluster coordination. This argument cluster contains a copy of the first conjunct. Within this copy, constituents are co-indexed with the original constituents in the first conjunct.

determineConstituentType: For each node, determine its constituent type. We distinguish the following constituent types: heads, complements, adjuncts, conjunctions, constituents that are co-indexed with a `*RNR*`-trace, spurious traces, and argument clusters.

makeBinary: Binarize the tree.

percolateTraces: Determine the category of `*T*` and `*RNR*` traces in complement position, and percolate them up to the appropriate level in the tree.

assignCategories: Assign categories to nodes in the tree, starting at the root node. Nodes that are co-indexed with `*RNR*` traces are ignored at first, and then receive the category of the corresponding traces. Argument clusters are also ignored in this step.

treatArgumentClusters: Determine the category of constituents within an argument cluster that are co-indexed with elements in a first conjunct. Assign categories to the nodes within these constituent in the ordinary top-down fashion. Use type-raising and composition to assign categories to the intermediate nodes within the argument cluster.

cutTracesAndUnaryRules: At this point, the tree contains certain constituents that are not part of the CCG derivation, such as traces and the copy of the first conjunct within an argument cluster. These are all cut out. Resulting unary projections of the form $\mathsf{X} \Rightarrow \mathsf{X}$ are eliminated.

verifyDerivation: There is a small fraction of trees for which the algorithm does not produce a valid CCG derivation. These trees are discarded in this step. In most cases, this is due to argument cluster coordination ("template gapping") that is not annotated in a way that our algorithm can deal with.

getPredicateArgumentStructure: Use the co-indexation mechanism described in section 2.5.6 to project non-local dependencies, and generate the word-word dependencies that constitute the underlying predicate-argument structure.

As described above, we do not translate trees that contain gapping constructions (indicated by `NP-SBJ=...`). The last step in the algorithm verifies that the translation is indeed a valid CCG derivation.

3.15 Related work

3.15.1 An alternative algorithm

Watkinson and Manandhar (2001) present an alternative algorithm for the extraction of AB categorial lexicons from the Penn Treebank. However, they do not offer a way of dealing with the various null elements in the Treebank, which means that they can only process a small subset of the sentences in the corpus.[9] Furthermore, unlike ours, their algorithm does not correspond to a reverse derivation, and therefore it is unclear how the correctness of their translation can be guaranteed unless categories assigned in the initial step can later be modified.

In particular, without such a correction, it would be possible for their method to assign lexical categories to a sentence which cannot be combined to derive a sentential category. Their algorithm proceeds in four stages:

1. Map some POS tags to categories.
2. Look at the surrounding subtree to map other POS tags to categories
3. Annotate subtrees with head, complement and adjunct information using heuristics similar to Collins (1999).
4. Assign categories to the remaining words in a bottom-up fashion.

In the first step, Watkinson and Manandhar map some part-of-speech tags deterministically to CG categories. The example they give is `DT` → $\mathsf{NP/N}$. However, this analysis is only correct for determiners appearing in noun phrases in complement position. For instance, it is not the correct analysis for determiners in temporal NPs.[10] Consider the `NP-TMP` in the following example:

(73)
```
(S (NP-SBJ South Korea)
   (VP (VBZ has)
       (VP (VBN recorded)
           (NP a trade surplus)
           (NP-TMP (DT this)
                   (NN year))))
       (. .)))
```

Here is the derivation of the embedded verb phrase:

(74)
$$\begin{array}{cccc}
\textit{recorded} & \textit{a trade surplus} & \textit{this} & \textit{year}\\
\hline
\mathsf{(S[pt]\backslash NP)/NP} & \mathsf{NP} & \mathsf{((S\backslash NP)\backslash(S\backslash NP))/N} & \mathsf{N}\\
\cline{1-2}\cline{3-4}
\multicolumn{2}{c}{\mathsf{S[pt]\backslash NP}\quad >} & \multicolumn{2}{c}{\mathsf{(S\backslash NP)\backslash(S\backslash NP)}\quad >}\\
\hline
\multicolumn{4}{c}{\mathsf{S[pt]\backslash NP}\quad <}
\end{array}$$

[9] A search with `tgrep` showed that out of the 49,298 sentences in the Treebank, 34,318 contain a null element matching the regular expression /*/

[10] It is also not the correct analysis for `NP`s which only consist of one `DT` daughter, such as the following: `(NP (DT those))`, `(NP (DT some))`, `(NP (DT all))`

In step 3, they use very similar heuristics to ours (both are based on Collins (1999)) to identify heads, adjuncts and complements. Thus, the NP-TMP would be identified as an adjunct, and either the analysis given in the first step would have to be modified, or the categories cannot combine:

(75)
$$\begin{array}{cccc}
\textit{recorded} & \textit{a trade surplus} & \textit{this} & \textit{year} \\
\cline{1-1}\cline{2-2}\cline{3-3}\cline{4-4}
(\mathsf{S[pt]\backslash NP})/\mathsf{NP} & \mathsf{NP} & \mathsf{NP/N} & \mathsf{N} \\
\cline{1-2}\cline{3-4}
\multicolumn{2}{c}{\mathsf{S[pt]\backslash NP} \quad {>}} & \multicolumn{2}{c}{\mathsf{NP} \quad {>}}
\end{array}$$

Assuming that such cases can be detected and are corrected, it is not clear that their bottom-up translation procedure yields different results from our top-down method if the same heuristics are used to identify heads and distinguish between complements and adjuncts. In step 4, they assign variables to the lexical categories of words which have not been assigned categories yet, then traverse the whole tree bottom-up and instantiate these categories using the head/complement/adjunct information already available to instantiate these variables. However, some of this information will only be available at the top of the (sub)tree, and will thus presumably be percolated down the tree through the variables. In such cases, the resulting categories should be the same.

As Watkinson and Manandhar use AB categorial grammar, which only has function application, it is also not clear how they could extend their algorithm to deal with the *T* and *RNR* traces in the Treebank. Furthermore, they could not strip off the outer arguments of the head category when determining the categories of adjuncts, because AB categorial grammar does not allow composition. We would expect this to lead to a larger, less compact lexicon.

3.15.2 Related work using other grammar formalisms

The algorithm presented here is similar to algorithms which extract Lexicalized Tree-Adjoining Grammars from the Penn Treebank (Xia *et al.*, 2000; Xia, 1999; Chen and Vijay-Shanker, 2000; Chiang, 2000, 2004). All of these algorithms rely crucially on head-finding procedures and heuristics to distinguish complements from adjuncts; therefore different implementations of the same algorithm can yield very different lexicons (see Chen and Vijay-Shanker (2000) for the impact of different complement/adjunct heuristics). Cahill *et al.* (2002) present an algorithm to annotate the Penn Treebank with Lexical Functional Grammar F-structures.

3.16 Summary

This chapter presented the algorithm which creates the syntactic derivations in CCGbank by translating the phrase-structure trees in the Penn Treebank and outlined the kinds of changes that need to be performed on the trees before this algorithm can applied in order to obtain correct CCG analyses. The head-finding rules and heuristics that were used to distinguish complements from adjuncts are listed in appendix A. Appendix B contains a detailed list of the preprocessing steps. The next chapter analyses the coverage of the translation algorithm and the resulting grammar and lexicon.

Chapter 4

Statistics of the CCGbank grammar and lexicon

Here we first examine briefly the coverage of the translation algorithm on the entire Penn Treebank. Then we examine the CCG grammar and lexicon that are obtained from CCGbank. Although the grammar of CCG is usually thought of as consisting only of the combinatory rule schemata, we are interested here in the instantiations of these rules, because statistical parsers such as Hockenmaier and Steedman (2002b) or Clark and Curran (2004) are trained on these instantiations. We report our results on sections 02-21, the standard training set for Penn Treebank parsers, and use section 00 to evaluate coverage of the training set on unseen data.

4.1 Coverage of the translation algorithm

CCGbank contains 48,934 (99.44%) of the 49,208 sentences in the entire Penn Treebank (sections 00-24). For the remaining 274 sentences, the translation algorithm failed to provide a CCG derivation. 173 of these cases are due to gapping constructions. There are 107 instances of sentential gapping, where the second conjunct consists of a subject NP and arguments of the verb. As explained in section 3.10.4, we did not attempt to adapt the algorithm to this construction, which Steedman (2000) analyzes with the rule of decomposition. An additional 49 instances of (non-sentential) gapping could not be processed by the translation algorithm. One example that cannot be processed by our algorithm is discussed in section 3.10.3. For 117 out of the missing 274 sentences the output of the algorithm was not a valid CCG derivation; this includes 17 cases of gapping.

A fundamental assumption behind the automatic translation of syntactially annotated corpora into different grammatical formalisms such as CCG, TAG, HPSG or LFG is that the analyses that underly the original annotation can be mapped directly (or, at least, without too much additional work) into the desired analyses in the target formalism. This is only the case if all constructions that are treated in a similar manner in the original corpus are also treated in a similar manner in the target formalism. As our research and the work of others (Xia (1999), Chiang (2000), Chen and Vijay-Shanker (2000), Cahill *et al.* (2002)) on the Penn Treebank has shown, such a correspondence exists in most cases. Gapping seems to be one kind of construction where this correspondence does not seem to hold.

4.2 The lexicon

The lexicon, which pairs words with their lexical categories, is crucial in CCG. From CCGbank, we can obtain a lexicon that can be used by any CCG parser, including non-statistical ones.

Number of entries A lexicon extracted from sections 02-21 has 74,669 entries for 44,210 word types (or 929,552 word tokens). The expected number of lexical categories per token in sections 02-21 is 19.19. This is because some high-frequency function words also have a large number of categories (see table 4.1).

Word	#Lexical Categories	Word frequency
as	130	4237
is	109	6893
to	98	22056
than	90	1600
in	79	15085
–	67	2001
’s	67	9249
for	66	7912
at	63	4313
was	61	3875
of	59	22782
that	55	7951
-LRB-	52	1140
not	50	1288
are	48	3662
with	47	4214
so	47	620
if	47	808
on	46	5112
from	46	4437

Table 4.1: The 20 tokens with the highest number of lexical categories (sections 02-21)

Category frequency f			Number of categories
100,000	$\leq f <$	220,000	2
10,000	$\leq f <$	100,000	13
1,000	$\leq f <$	10,000	49
100	$\leq f <$	1,000	108
10	$\leq f <$	100	253
5	$\leq f <$	10	131
2	$\leq f <$	5	291
0	$< f \leq$	1	440

Table 4.2: The frequency distributions of lexical categories (section 02-21)

Number and growth of lexical category types There are 1286 lexical category types. 847 of these appear more than once, 680 appear more than twice, and 556 appear five times or more. The frequency distribution of lexical categories is shown in table 4.2. Figure 4.1 examines the growth of lexical category types in sections 02-21. New categories that appear only once keep appearing, but the numbers seem to have converged for categories that appear at least twice or more. After having processed 58% of the training data, all categories that appear at least five times have been seen. In fact, 95% of the categories that appear at least five times have been seen after 27% of the data have been processed. Inspection of the category types that appear only once suggests that many arise from noise in the annotation and should not be used by the parser. However, a large number of the categories that occur only once are linguistically important, since they include pied piping constructions or verbs which take expletive arguments, etc.

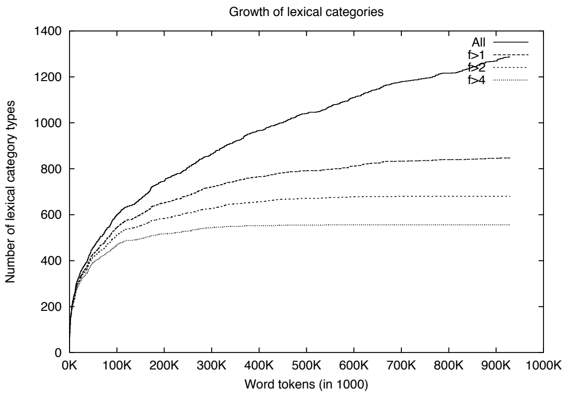


Figure 4.1: The growth of lexical category types (sections 02-21)

Lexical coverage How well does the lexicon cover unseen data? The lexicon from sections 02-21 contains the correct entries for 94.0% (42,707 out of 45,422) of the tokens in section 00. 96.2% of the tokens that appear in section 00 also appear in sections 02-21. 952 (35.1%) of the unseen entries have the category N, and 791 (29.1%) have category N/N.

4.3 The grammar

Size and growth of the grammar The grammar in sections 02-21 has 3262 specific rule instantiations. Of these, 1146 appear only once, and 2027 appear less than five times. However, these low-frequency rules are not necessarily due to noise, since many are instantiations of type-raising, coordination or punctuation rules. Table 4.3 lists the most frequent rule instantiations. The distribution of rule frequencies is given in table 4.4. The growth of rule instantiations is shown in figure 4.2. As in the case of lexical categories, new types keep appearing. Each new lexical category requires at least one new rule, therefore this correlation is hardly surprising. Like lexical categories, the number of rules that appear at least twice has converged.

Grammatical coverage Out of the 51,984 individual rule instantiations (corresponding to 844 different rule types) in section 00, 99.9% (51,932) appear in sections 02-21. Out of the 52 missing rule instantiations (corresponding to 38 rule types, because one rule[1] appears 13 times in one sentence), six involve coordination, and three punctuation. One missing rule is an instance of substitution. Two missing rules are instances of type-raised arguments that combine with a verb.

[1] (NP\NP)/(NP\NP) → ((NP\NP)/(NP\NP))/N[num] N[num]

Rule	Frequency
147622	N → N/N N
115516	NP → N
91536	NP → NP[nb]/N N
64404	NP → NP NP\NP
56909	S[dcl] → NP S[dcl]\NP
43291	NP\NP → (NP\NP)/NP NP
37386	TOP → S[dcl]
35423	S[dcl] → S[dcl] .
22184	(S\NP)\(S\NP) → ((S\NP)\(S\NP))/NP NP
16969	PP → PP/NP NP
16585	S[dcl]\NP → S[dcl]\NP (S\NP)\(S\NP)
15686	S[dcl]\NP → (S[dcl]\NP)/NP NP
15293	NP → NP NP[conj]
13847	S[dcl] → S/S S[dcl]
12669	S[b]\NP → (S[b]\NP)/NP NP
12519	S[to]\NP → (S[to]\NP)/(S[b]\NP) S[b]\NP
10546	NP → NP ,
10504	S[dcl]\NP → (S[dcl]\NP)/(S[b]\NP) S[b]\NP
9283	S[dcl] → , S[dcl]
8184	NP[nb]/N → NP (NP[nb]/N)\NP

Table 4.3: The 20 most frequent rule instantiations in CCGbank, section 02-21

Rule frequency *f*			Number of rules
100,0000	$\leq f <$	150,000	2
10,000	$\leq f <$	100,000	16
1,000	$\leq f <$	10,000	75
100	$\leq f <$	1,000	219
10	$\leq f <$	100	604
5	$\leq f <$	10	319
2	$\leq f <$	5	882
0	$< f \leq$	1	1146

Table 4.4: The distribution of rule frequencies in CCGbank, section 02-21

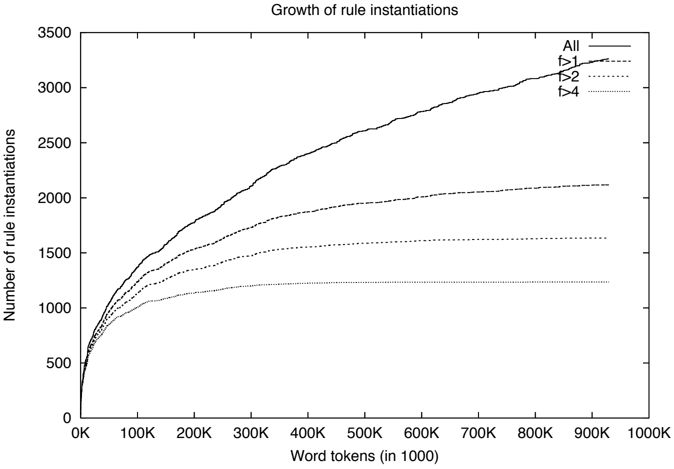


Figure 4.2: The growth of rule instantiations(Section 02-21)

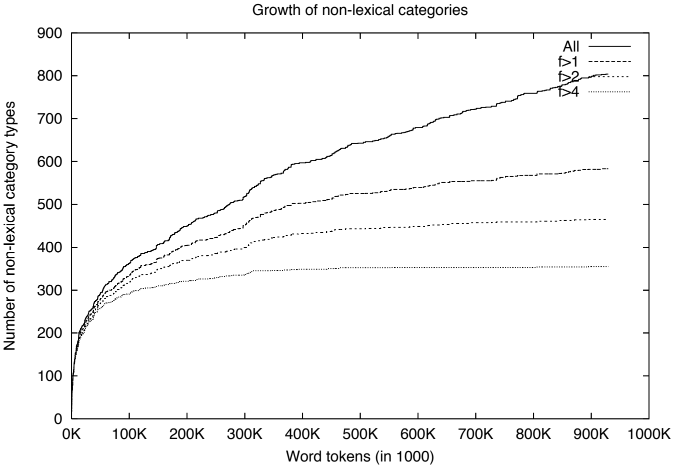


Figure 4.3: The growth of non-lexical category types (Section 02-21)

Chapter 5

Conclusion

CCGbank is a corpus of canonical CCG derivations that is obtained from the Penn Treebank by an automatic translation process. It has enabled the creation of robust, accurate, wide-coverage CCG parsers such as Hockenmaier and Steedman (2002b), Clark *et al.* (2002), Hockenmaier (2003b) and Clark and Curran (2004), which return not only syntactic derivations, but, unlike other wide-coverage Penn Treebank parsers such as Collins (1999) or Charniak (2000), also word-word dependencies which represent the underlying deep predicate-argument structures. Although these predicate-argument structures are only an approximation of the true semantic interpretation, and despite the problems highlighted in section 2.5.8, they may nevertheless prove useful for tasks such as summarization and question-answering (Clark *et al.*, 2004). Because of CCG's transparent syntax-semantics interface, CCGbank can also be seen as a first step towards the creation of a treebank annotated with semantic interpretations, and Bos *et al.* (2004) and Bos (2005) have demonstrated that the output of a parser trained on CCGbank can in fact be translated successfully into Kamp and Reyle (1993)'s Discourse-Representation Theory structures.

A fundamental assumption behind the automatic translation of syntactially annotated corpora into different grammatical formalisms such as CCG, TAG, HPSG or LFG is that the analyses that underly the original annotation can be mapped directly (or, at least, without too much additional work) into the desired analyses in the target formalism. This can only hold if all constructions that are treated in a similar manner in the original corpus are also treated in a similar manner in the target formalism. Our research and the work of others (eg. Xia (1999), Chiang (2000), Chen and Vijay-Shanker (2000), Cahill *et al.* (2002)) on the Penn Treebank has shown that such a correspondence exists in most cases. Although the output of current Penn Treebank parsers is linguistically impoverished, the Penn Treebank annotation itself is not, and it was precisely the additional information present in the null elements and function tags that are ignored by other parsers that made the creation of CCGbank possible.

However, as explained in this report, a number of obstacles remain to the translation of Penn Treebank trees to a linguistically richer formalism such as CCG. The flat noun phrase structure is one of them: although it is possible to introduce a separate noun level, compound nouns still have a flat internal structure, which is semantically undesirable, and leads us to assume a strictly right-branching (but often incorrect) analysis. The Treebank markup also makes appositives difficult to distinguish from noun phrase lists. Moreover, postnominal modifiers are always attached at the NP-level which is also undesirable from a semantic point of view. At the verb phrase and sentence level, there are other problems. For instance, the acquired CCG lexicon does not have a proper analysis of phrasal verbs. The distinction between complements and adjuncts is similarly difficult to draw, especially for constituents annotated with the `CLR`-tag, which is known to be used fairly inconsistently across the corpus. Kinyon and Prolo (2002) describe a different set of heuristics for distinguishing complements from adjuncts than those used here, and it would be interesting to compare the two approaches. In a future version of CCGbank, it might be possible to make use of the complement-adjunct distinction that is implicit in the semantic role annotation of the Penn Treebank that is currently being developed within the Proposition Bank project (Palmer *et al.*, 2005).

There are certain other constructions, such as gapping, where the correspondence between the Penn Treebank analyses and ours (and possibly others) breaks down. However, these constructions are so varied, and yet so rare, that manual annotation may be better suited to their analysis. Other linguistic phenomena, such as fragmental utterances, or multi-word expressions, lack an adequate syntactic analysis in the source corpus. We cannot fault the creators of the Penn Treebank for this: in many cases, the correct analyses for these constructions are unclear.

We believe that CCGbank is an important first step towards the creation of a corpus that contains logical forms. However, as outlined above, more work is required to obtain the desired syntactic derivations. We are also painfully aware that our current method of generating interpretations or predicate-argument structures is flawed, since it assumes that all lexical categories that have the same syntactic type use the same co-indexation mechanism to percolate dependencies between their arguments. This is clearly not the case, because subject control verbs such as *promise* have the same syntactic category as object control verbs like *persuade*. Future research should address this problem, although we have found our current co-indexation to be largely correct.

Despite these shortcomings, the grammar that is implicit in CCGbank covers a wide range of syntactic and semantic phenomena, and we hope that CCGbank will be a valuable resource, not just for statistical parsing, but also for other applications.

Appendix A

Identifying heads, complements and adjuncts

A.1 Head-finding rules

The head-finding rules are adapted from the rules developed by Collins (1999) and Magerman (1994). They are given for each non-terminal label, and are to be read as follows: going from left to right (←), or from right to left (→), the first constituent with label XP is head. Otherwise, search for the next constituent in the list. A ← or → next to the nonterminal label indicates the search direction for all head finding rules for this nonterminal label. If none of the constituents listed is found, the leftmost (←) or rightmost (→) constituent is chosen. In general, constituents which carry a functional tag are not identified as heads, unless this is indicated by an "all" next to the rule.

ADJP (←) SO*, (←) ADJP, (←) DOLLAR, (→) VBN, (→) VBG, (→) JJ, (→) JJR, (→) JJS, (→) NN, (→) NNS, (→) NNP, (←) QP, (←) DT, (→) CD, (→) RB, (→) ADVP, (→) RBR, (←) IN, (←) VBD, (←) VBP, (←) VB, (←) NP, (←) FW, (←) RBS, (←) SBAR, (←) PDT

ADVP If the rightmost daughter is RBR, JJ or JJR, it is head. Otherwise: (→) RB, ADVP, JJ, RBR, RBS, IN, JJR, JJS, FW, TO, CD, JJR, JJ, IN, WHADJP (all), NP, JJS, NN, RP

NAC ←: DT, NN, NNS, NNP, NNPS, NP, NAC, EX, DOLLAR, CD, QP, PRB, VBG, JJ, JJS, JJR, ADJP, FW

S-ADV ←: MD, VP, .*-PRD

S ←: MD, VBP, VBD, VBZ, TO, VP, "-PRD", ADJP, S, SBAR, SINV, UCP, INTJ, NP

SINV If the first daughter is ADVP with head word *"so"*, *"So"*, or *"SO"*, it is head. Otherwise ←: VBZ, VBD, VBP, VB, MD, VP, S, SINV, ADJP< NP

SQ ←: VBZ, VBD, VBP, VB, MD, VP, SQ, SBARQ

SBAR If there are two daughters and the first one is WHNP with trace, and the second daughter is a S with a to-infinitival VP, the S is head. Otherwise, ←:
IN, WHNP, WHPP, WHADVP, WHADJP, WDT, RB, MD, DT, X, PP, PRN, UCP, VBD, VB, S, SQ, SINV, SBAR, FRAG

SBARQ ←: WHNP, WHADVP, WHADJP, WHPP, WP (FRAGS as SBARQ), SBARQ

PP If the first daughter is PRN, it is the head, otherwise →: IN, TO, VBG, VBN, VB, RP, PP, FW Else the leftmost child is head.

VP ←: TO, VBD, VBN, MD, VBZ, VB, VBG, VBP, VP, JJ, S, SYM, NN, NNS, NP

QP ←: QP, ASTAG, NP, NOUN, NN, NNS, RBR, UP, NP, DOLLAR, SYM, CD Else, the rightmost child is head.

WHNP ←: WDT, WP, WPS, WHADJP, WHADVP, WHNP, IN, WRB

WHADJP ←: WRB, WHADVP, WP, RB

WHADVP →: WRB, IN

WHPP →: WHNP

CONJP →: CC, JJ, IN, RB, CONJP

UCP The leftmost child is head

FRAG If the first child is CC or CONJP, and the second child is not SBAR, SBARQ, RB, take the first child. otherwise, ←:
WP, WHADVP, RB, SBAR,IN, S, NP, ADVP, PP

PRN If the first child is ":" or LRB, and the last child is not ":", then the first child is head, unless the second child is SBAR, PP, PRN, IN, or if there are only two children.

If the first child is PRN, it is is head. Otherwise, ←:
PRNS, S, NP, VP, PP, SBAR, PP, UCP, ADJP, ADVP, RB PRNS is the label that is inserted for S, or SINV under PRN to trigger the unary type-changing rule.

INTJ The leftmost child is head

X ←: DT, otherwise take the leftmost child

RRC →: VP, NP, ADVP, ADJP, PP

PRT (→): RP

LST (→): LS, COLON

NP, NPnonBare **or** NX**:**

- Within a NP or NPnonBare: going from left to right, find the first NP that does not have an adjunct tag.
- Otherwise: if the first child is a determiner or quantifier, the first child is head. Determiner or quantifier labels are:
CD, DT, PDR, POSSDT, QP, WDT, WPD, WRB, WP, JJ, POSS The only exception to this rule is that if the second child is a "non-bare" NP, it is head. This arises if the first child is *both*, *neither*, *such a*, etc.
- Otherwise: if the last child is POS, it is head.
- Otherwise: Going from right to left, the first constituent that has a noun label is head. Noun labels are the following:
NN, NNP, NNPS, NNS, NX (not co-indexed with a *RNR*-trace), POS, JJR, VBG, VB, VBZ, VBN, JJS, QP
- Otherwise, going from left to right, search for a NP or WHNP (this is for NPs with NP adjuncts).
- Otherwise, search for a $ or ADJP from right to left.
- Otherwise, search for a CD from left to right.
- Otherwise, search from the right for a JJ, RB, QP, DT
- Otherwise, search from the left for a ADVP, FW, INTJ, POSS
- Otherwise, take the leftmost node.

In parallel coordinations (coordinate constructions of the form (XP XP CC XP)), go from left to right, and identify the first constituent with the same label as the parent as head.

Head-finding rules for additional nonterminals The translation procedure adds a number of additional nonterminals and POS tags in order to deal with specific constructions. These serve the purpose of recognizing specific constructions. For example, ATP is the constituent that contains the words *at least/most*; SO is the POS tag assigned to *so* in constructions of the form *so ADJ that*. DOLLAR is the POS tag assigned to the string DOLLARS which replaces monetary expressions. The constituent XNP contains a NP and a comma, and is required to trigger the binary type-changing rules that account for extraposition.

PRNS (the label that is inserted for S, or SINV under PRN to trigger the unary type-changing rule) ←: VP, VB

ATP (for *"(at least/most) X"*) (←) IN

INP (*"(in order) to do something"* (→): IN

pipedPipeNP The leftmost child is head

THANP (the constituent that is inserted in QPs such as *more than three times*, and that includes *than* and everything that follows it): The rightmost child is head

POSSDT (The constituent that includes the possessive *"'s"* and the preceding noun phrase): The rightmost child (ie. the possessive *"'s"*) is head

SOADJ (←): ADJP Otherwise: (→): VBN, VBG, JJ, JJR, JJS, NN, NNS Otherwise: (←) JJ, JJR. Else, the leftmost child is head.

DTP (a constituent consisting of a predeterminer and a determiner) (→): DT

XNP rightmost (all)

A.2 Complement-adjunct distinction

If the node has a complement tag (-SBJ, -CLR, -DTV, -TPC, -PRD), it is a complement, unless it is an ADVP-CLR which expands to a null element, or it also has an adjunct tag in addition to a -CLR tag. PP-TPC nodes are only complements under SINV, or if they carry a -PRD tag. Nodes with -PRD tag that are children of UCPs are not complements. UCPs themselves are treated as if their label was the label of their first child. Nodes with adjunct tags (-ADV, -VOC, -BNF, -DIR, -LOC, -MNR, -TMP (with the exception of PP-TMP nodes under ADJP) and -PRP) cannot be complements. NP-TPC nodes that are not co-indexed (eg. NP-TPC-1) are also treated like adjuncts, since there is a resumptive pronoun or other element.

For each constituent type, this is a list of the constituents that are recognized as complements when they appear as children of this constituent:

ADJP: PP unless it is left to the head or headed by IN *"than"*, S, SBAR (not if the complementizer is *"than"*, *"as"*, *"so"*, *"which"*), NP, SOADJ

ADVP: PP, unless it is left to the head or headed by IN with head word *"than"*, NP, SOADJ, SBAR (not *"than"*, *"as"*, *"so"*, *"which"*, *"before"* or WHADVP, and not if there is an intervening punctuation mark)

NP: NOUN, NPnonBare

NX: NOUN, NPnonBare

NAC: NOUN, NPnonBare

NOUN: SBAR

S: S with numerical index, but not co-indexed with a *RNR*-trace; VP if the head of S is MD

SINV: NP, S (and not a parallel conjunction), SBARQ, VP

SQ: VP, NP, WP, S

SBAR: NN (*"in order to..."*), S, SQ, VP, SINV, SBARQ (FRAG is changed to SBARQ)

SBARQ: SQ, WHNP, SINV, S, SBARQ, NP, VP

PP: NP

VP: NP, VP (unless a parallel conjunction), SBARQ, S, SQ, ADJP, PPs with *T*-trace

S **under** VP an S under a VP parent is a complement if it is not preceded by a comma and another S.

SBAR **under** VP An SBAR is a complement if it is:

- not preceded by a comma
- preceded by a comma, and a relative clause with *"which"*
- preceded by a comma, and if there is no other intervening complement between the comma and the verb (this is wrong for intransitive verbs).
- preceded by a comma, and if there is another comma somewhere in the yield between the verb and the immediately preceding comma.

WHNP: NOUN

WHADJP: JJ, ADVP, ADJP

WHADVP: JJ, ADJP, ADVP (unless is the first child), RB (unless it is the first child or headed by *"not"*)

WHPP: IN, TO, piedPipeNP

UCP: treat like it had the same label as its first child

FRAG: NP, PP (if left from head)

PRN: RRB, COLON (if the head is LRB or COLON)

X: ADJP, JJR

Complement-adjunct distinction for additional constituents Additional constituents that are inserted during translation have their own complement-adjunct rules.

POSSDT: NP, NOUN

SOADJ: PP, unless it is left to the head or headed by IN with head word *"than"*, NP, SOADJ

ATP: JJS, RBS, DT

Appendix B

Changes to the Treebank

This chapter specifies how the constituents Treebank are translated. For each nonterminal label, the preprocessing steps, head-finding rules and complement-adjunct rules are given. In general, only preprocessing changes that apply to more than one tree are listed.

B.1 Correcting tagging errors

The following heuristics attempt to correct some systematic tagging errors. Only tagging errors that led to errors in category assignment were attempted to correct.

Under `ADJP`: if a `NN`, `VBN`, or `VBG` appears after a conjunction, it is a second head in a coordinate construction – change it to `JJ`

***"whether"* as `CC`; any `CC` as first child of `SBAR`:** change to `IN` (preposition). Subordinating conjunctions in CCG are like modifiers, not like coordinating conjunctions.

Infinitives tagged as `VBP`: change to `VB`, eg. within inverted questions

Infinitives within *"do/would"* questions: if the first child of an `SQ` is any form of *"do"* or *"would"*, and the first child of the `VP` underneath the `SQ` is not a `VB`, `MD`, `VBZ` or `-NONE-`, it is changed to `VB`.

***"and"* is always `CC`**

If the last child of an `NP` is `JJ`, it is changed to `NN`. This tagging error leads to errors with the head-finding rules, hence it is changed.

`VBP` **mistagged as** `VB` under `SQ`, if the first or second child is `VB`, it is changed to `VBP`.

Mistagged `NNP`s: occasionally words in capital letters are tagged `NNP` even though they are not, eg. *"Does"*, *"Should"*. Under `ADVP`, if the adverb has only one child, and it is tagged as `NNP`, change this to `RB`. This is a tagging error, and we do not want unknown `NNP`s to have adverbial categories.

Unrecognized prepositions and subordinating conjunctions: if the first child of a `PP` is tagged `NN`, it is changed to `IN`.

Possessive *'s* as `VBZ`: if the possessive *"'s"* under `POSSDT` is labelled `VBZ`, it is changed to `POS`

`VBZ` **as possessive *"'s"*:** under a `VP`, possessive *"'s"* is labelled `VBZ`.

Simple past tense (VBD) **mistagged as past participles** (VBN) If a VBN is the head of a VP and its grandparent is a S, we assume it should not be a past participle but a simple past tense form (VBD). Under SINV, if the first child is VBN, it is changed to VBD.

Bare infinitives (VB) **mistagged as** VBP**,** VBD **or** VBN**:** in to-infinitives – if they are children of a VP which itself is the child of a VP whose head is TO.

-ing forms tagged as NN**,** NNP**,** JJ**,** RB If an "ing"-form appears within a VP, and is tagged as NN, NNP, JJ or RB, it is changed to VBG.

-ed forms tagged as NN**,** NNP**,** JJ If a form eding in "ed" appears within a VP and is tagged as NN, NNP or JJ, it is changed to past tense VBD if the parent of the VP is S, and to past participle VBN if the parent of the VP is a VP or if the VP is passive. (This only captures mistagged regular verbs.)

Other forms tagged as NN**,** NNP **or** JJ Other verbs are also somtimes mistagged as NN, NNP or JJ. They are changed to VBZ if the parent of the VP is a sentence, yielding a declarative sentence, to VB if the parent of the VP is a VP headed by a do-form, and to VBN otherwise.

RB **as first child of** VP**:** changed to VB/VBG/VBN (depending on ending) if there is no other head in the VP.

VBZ **mistagged as** NNS **underneath** VP if an NNS is the first or second child of a VP, or the third child and immediately preceded by a conjunction, it is a VBZ.

VB **mistagged as** VBP**:** if a VBP is a child of a VP that is a child of an SINV, the VBP is changed to VB.

VBN **mistagged as** VBD**:** if a VBD is a child of a VP that is a child of an SINV, the VBD is changed to VBN.

VB **mistagged as** NN**:** if a NN is a child of a VP that is a child of an SINV, the NN is changed to VB.

***"that"* underneath a** NP **is** DT (not IN).

RB **as head of** VP**s embedded in** TO-VP**s** (there should be other cases too, but this one is easiest to detect). Are changed to VB

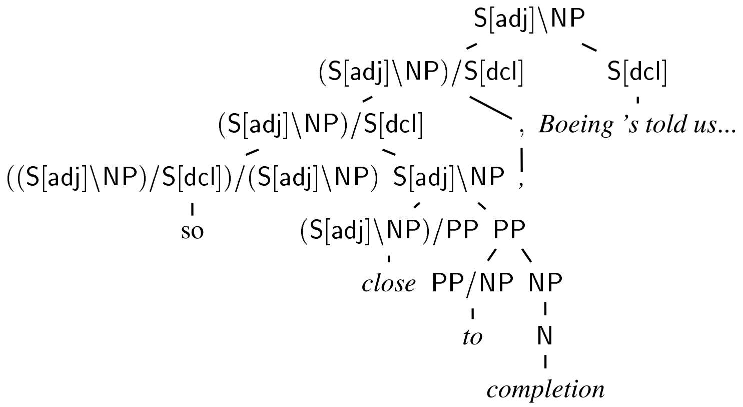
NNS **appearing as first or second child in a** VP Are changed to VBZ (3rd person singular verb)

B.2 Preprocessing ADJPs

B.2.1 *"So"* + adjective; *"as"*+ adjective

- *"so"* ADJP...: If there is a PP and an SBAR, we want the PP to be an argument or modifier of the adjective itself, and the SBAR to be an argument of the *"so"*:

```
(ADJP-PRD (ADJP (RB so)
                (RB close))
          (PP (TO to)
              (NP (NN completion)))
          (, ,)
          (SBAR (-NONE- 0)
                (S (NP-SBJ (NNP Boeing))
                   (VP (VBZ 's)
                       (VP (VBN told)
                           (NP (PRP us))
(SBAR (-NONE- 0)
      (S (NP-SBJ (EX there))
         (VP (MD wo)
             (RB n’t)
             (VP (VB be)
                 (NP-PRD (DT a)
                         (NN problem))))))))))
```



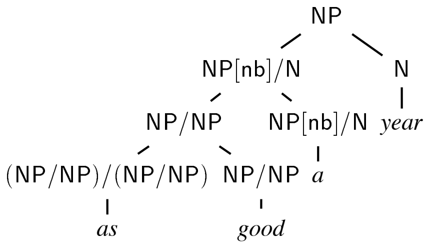
- Within a noun phrase, an adjective phrase with *“so”* or *“as”* can modify a determiner, eg *“so personal an issue”*, *“as good a year”* (Note that this is different from *“so many people”*). The Treebank gives this construction a flat analysis:

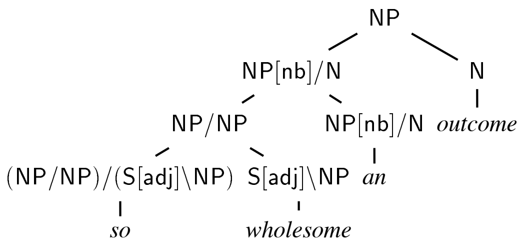
  ```
  (NP (ADJP (RB so)
            (JJ personal))
      (DT an)
      (NN issue))
  ```

  However, since we distinguish determiners and nouns, we analyze this construction as follows:

  ```
  (NP (DTP (ADJP (RB so)
                 (JJ personal))
           (DT an))
      (Noun (NN issue)))
  ```

  Here the `ADJP` modifies the determiner:



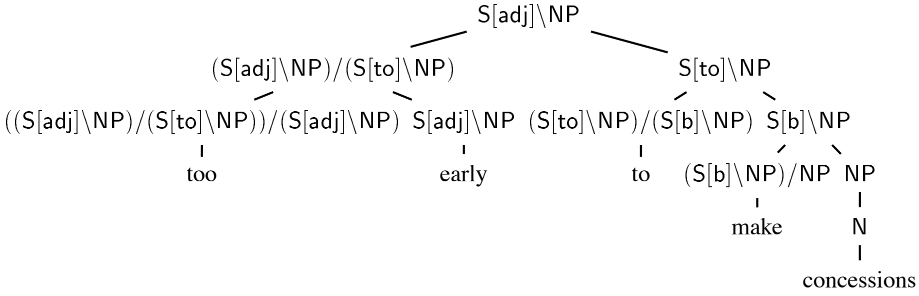


B.2.2 *"Too JJ toVP"*

This construction is treated in a similar way to the "so JJ" cases. The flat Treebank analysis (see below) is modified so that *too* takes the adjective and the to-VP as argument:

```
(S (NP-SBJ the first round)
   (VP (VBD was)
       (ADJP-PRD (RB too)
                 (JJ early)
                 (S (NP-SBJ (-NONE- *))
                    (VP (TO to)
                        (VP (VB make)
                            (NP (NNS concessions)))))
```

Here is the CCG translation:



B.2.3 Other changes

- If an ADJP expands to a conjunction of ADJPs, the following heuristics were implemented to deal with adjuncts of these conjuncts that appear at the same level as the coordination itself: If a PRN child immediately precedes an ADJP, it is re-analyzed as a child of its ADJP sibling. Anything that appears after the last ADJP is re-analyzed as a child of this ADJP.

- ADJP-PRD under NP is changed to ADJP. This is an appositive, and should be treated as an adjunct, not a complement.

B.3 Preprocessing ADVPs, WHADVPs and WHADJPs

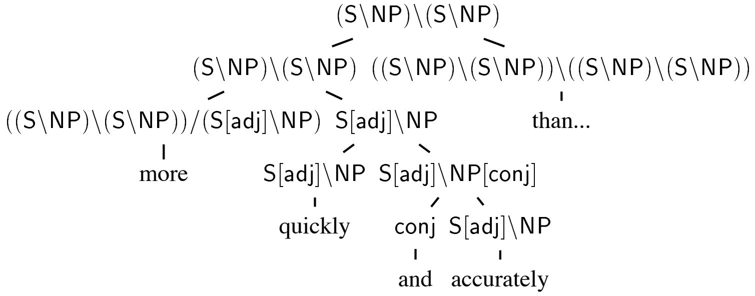
B.3.1 Inserting coordinate structures

The Treebank gives coordinate ADVPs a flat structure:

```
(ADVP (DT either)
      (RB legally)
      (CC or)
      (RB illegally))
```

In order to correctly identify the heads in these cases, a new ADVP which consists of the conjunction and the two constituents surrounding it is inserted. This also disambiguates constructions such as the following in a manner that assigns the modifier wide scope over both conjuncts (which is correct in this example, but might not always be the right analysis):

```
(ADVP  (ADVP (RBR more)
             (RB quickly)
             (CC and)
             (RB accurately))
       (SBAR (IN than)
             (S  ...)))
```



B.3.2 Other changes

- Incorrect complement tags: ADVP under NP are not complements; therefore PRD tags are deleted.
- If an ADVP consists of a determiner (DT) followed by a nominal tag (NN), its label is changed to NP-TMP, and it is preprocessed in the same manner as NPs. This guarantees that nouns are inserted.
- The label of ADVPs that consist of three children where the second child is "to" is changed to QP-ADV, and they are preprocessed like QPs, eg:

  ```
  (ADVP (CD 847)
        (TO to)
        (CD 644))
  ```

- The label of ADVP-PRD-TPCs that appear under S and whose first daughter is *"so"* or *"So"* is changed to ADVP.
- *"(ADJP (ADVP more than) JJ)"*: here we assume that *"than"* is head and takes *"more"* as argument.

B.4 Preprocessing Ss

B.4.1 Type-changing rules

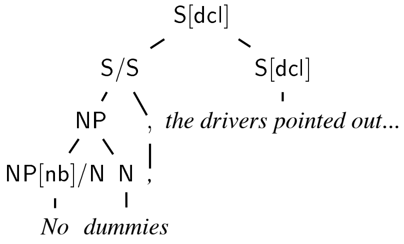
- A binary type-changing rule for sentences that are followed by a comma is triggered if the first element of a sentence is a S-ADV small clause with empty NP-SBJ and NP-PRD:

```
(S (S-ADV (NP-SBJ (-NONE- *-1))
          (NP-PRD No dummies))
   (, ,)
   (NP-SBJ-1 the drivers)
   (VP pointed out they still had space ...).
```

This is changed to:

```
(S (XNP (S-ADV (NP-SBJ (-NONE- *-1))
          (NP-PRD No dummies))
   (, ,))
   (NP-SBJ-1 the drivers)
   (VP pointed out ...)).
```

This yields the following CCG analysis:



- There are two similar binary type-changing rules for NP adjuncts that appear either at the end of a sentence and are preceded by a commas or that appear at the beginning of a sentence and are followed by a comma.
- Sentential subjects S-NOM-SBJ are treated like NPs, and we assume a type-changing rule that can change an NP to a sentence.

B.4.2 Changing the bracketing

- Under an S, constituents such as ADVP, PP, TO that appear adjacent to a VP are moved under the VP, not at S-level. If the VP is surrounded by commas, the modifier and the commas are all appear under the VP.
- (S NP-SBJ VP CC S): An S spanning the NP-SBJ and VP in the first conjunct is inserted.
- In coordinate construction, premodifiers of the second or further conjunct that appear as sisters of the conjunct are moved underneath that conjunct. There are similar preprocessing steps for other constituents, eg. fir VPs and NPs.
- If an NP-SBJ or other consistuent with -SBJ tag in an S is surrounded by parentheses that appear at the S level, the parentheses are moved underneath that constituent:

```
(S (-LRB- -LRB-)
   (NP-SBJ (DT the)
           (NN departure))
   (-RRB- -RRB-)
   (VP ...)
```

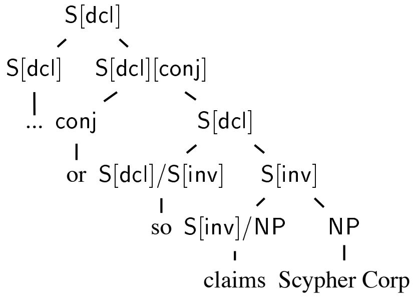
This is changed to:

```
(S (NP-SBJ (-LRB- -LRB-)
           (NP-SBJ (DT the)
                   (NN departure))
           (-RRB- -RRB-))
   (VP ...))
```

- If there is an SBAR after a VP in clefts (S with a -CLF tag), the SBAR is moved underneath the VP and its label is changed to SBAR-REL, so that the cleft analysis comes out correctly in the translation.
- *so* in SINV: If an S has an SINV daughter whose first daughter is *“so”*, we analyze the *“so”* as taking an inverted sentence as argument, and returning a declarative sentence:

```
( (S (S PORTING POTABLES just got easier))))
   (, ,)
   (CC or)
   (SINV (ADVP (RB so))
         (VP (VBZ claims))
         (NP-SBJ (NP Scypher Corp.))
   (. .)) )
```

In order to obtain this analysis, an SINV is inserted which contains the VP and NP-SBJ, the original SINV is relabeled SBAR, resulting in the following CCG analysis:

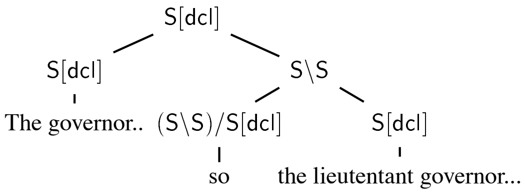


B.4.3 Other changes

- (S S IN S): If an S has a daughter with label IN which is follwed by an S, an ‘‘SBAR-ADV’’ node is inserted, and the IN and S are made daughters of this ‘‘SBAR-ADV’’. In most of these cases, the IN is *so* or *for*:

```
(S (S The governor couldn’t make it)
   (, ,)
   (IN so)
   (S the lieutenant governor welcomed the special guests)
   (. .))
```

This results in the following CCG analysis:



- S-TPC with a null subject is changed to S-ADV.
- In coordinate S, PRNS, SBAR, SINV, VP:[1] If PRN, "..." or "–" follow the first conjunct, they are moved underneath the first conjunct.
- If there is a child IN immediately followed by a S, both of them should go under a SBAR-ADV:

```
(S (S The governor could n't make it)
   (, ,)
   (IN so)
   (S the lieutenant governor welcomed the special guests)
   (. .))
```

- If there are two children marked with the SBJ tag, only the second one is the real subject, so the first SBJ tag is deleted.

B.5 Preprocessing NPs

Most changes to NPs have to do with the insertion of a noun level into base (non-recursive) NPs, the reanalysis of possessive *"'s"*, and the placement of modifiers in coordinate constructions.

B.5.1 Reanalysis of NP structure

- The possessive *'s* and ' are analyzed as functors from NPs to determiners: a new constituent (labelled POSSDT) is inserted, which consists of the NP and the possessive *'s*.
- A "non-bare" NP is inserted in an NP after the first child if the first child is a PDT (*all, such, both, half* etc.), a WP (*what else*, *what other means*), or a JJ with head word *such*), or if the first child is a DT, and the new token to be added is a NP:

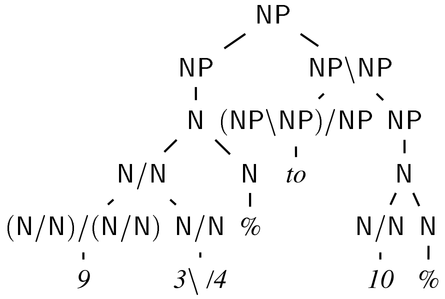
```
(NP (DT both)
    (NP (DT the) (NN quake))
    (CC and)
    (NP (NNP Hugo)))

(NP (DT all)
    (NP (DT the) (NN agency) (POS 's))
    (NNS policies))
```

- In coordinate construction, S-NOM should be recognized as conjunct, and a unary projection NP → S-NOM is inserted.

[1] this is not just S!

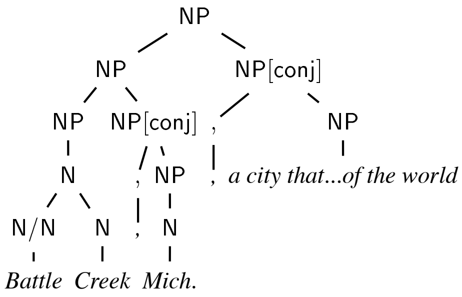
- In dates (NP-TMP) of the form *"Feb. 18"*, the month takes the number as complement (which is assigned the category N[num]).
- If NP → NP TO NP, we obtain the same analysis as for QPs

```
(NP (NP (QP (CD 8) (CD 1\/2))
        (NN %))
    (TO to)
    (NP (QP (CD 8) (CD 3\/8))
        (NN %)))
```



- If there are two adjacent NP children, the first of which has a comma as last child, the comma is lifted up to the top level in order to obtain the appositive analysis for the second NP:

```
(NP (NP (NP Battle Creek)
        (, ,)
        (NP (NNP Mich.))
        (, ,))
    (NP a city that calls itself the breakfast capital of the world))
```



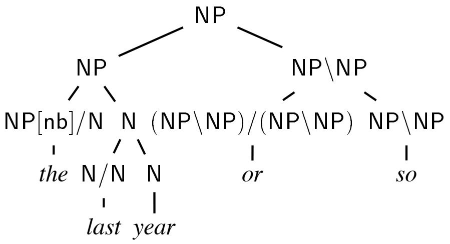
- If there are any clausal postmodifiers within a base NP, insert another NP consisting of the base NP only. This avoids the postmodifier becoming a modifier of the noun. For example, if the last child of an NP is a QP, the QP is a postmodifier (*or more*, *or so*, *out of X*)

```
(NP (DT the)
    (JJ last)
    (NN year)
    (QP (CC or)
        (RB so)))
```

This tree is changed in the following way before the noun level is introduced:

```
(NP (NP (DT the)
        (JJ last)
        (NN year))
    (QP (CC or)
        (RB so)))
```

Here is the CCG reanalysis:



We do not make this change if the QP is a purely numerical expression such as the following:

```
(NP (DT the)
    (VBG remaining)
    (QP (CD 2.2)
        (CD million)))

(NP (NN age)
    (QP (CD 59)
        (CD 1\/2)))
```

Similarly, there are cases where appositions are not recognized because there is an adjunct such as a PP, SBAR, PRN after the first NP:

```
(NP (NP (DT the) (NNP Center))
    (PP (IN for) (NP (NNP Security) (NNP Policy)))
    (, ,)
    (NP (DT a) (JJ conservative) (NNP Washington) (NN think-tank))
```

In these cases, we insert a NP which includes the NP and its adjunct.

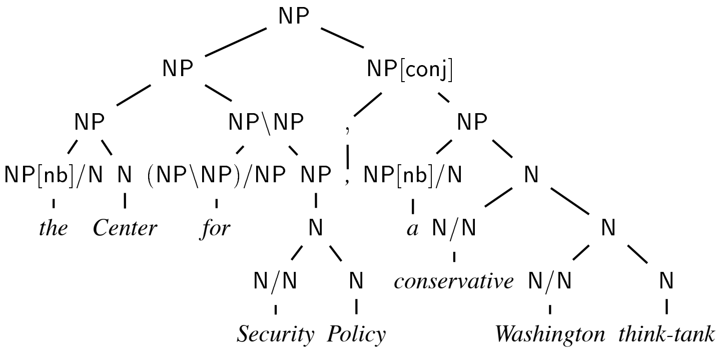
Also, if there is a RB followed by a NP within a NP, an NP which includes the RB and NP, is inserted.

In coordinate NPs, if there is a postmodifier adjacent to one of the conjunct NPs (except the last one). attach it to this NP. Postmodifiers are PP, PRN, SBAR, RRC, ADJP. These postmodifiers must not be coindexed with a *RNR* trace.

```
(NP (NP (DT the)
        (NNP Center))
    (PP (IN for)
        (NP (NNP Security)
            (NNP Policy)))
    (, ,)
    (NP (DT a)
        (JJ conservative)
        (NNP Washington)
        (NN think-tank)))
```

This becomes

```
(NP (NP (NP (DT the)
            (NNP Center))
        (PP (IN for)
            (NP (NNP Security)
                (NNP Policy))))
    (, ,)
    (NP (DT a)
        (JJ conservative)
        (NNP Washington)
        (NN think-tank)))
```



A similar preprocessing step deals with RB and ADVP immediately preceding a conjunct NP.

- If the last child of a NP with more than two children is a QP, it is usually an adjunct to the whole NP such as *or more*, *or so*, *out of X*, and we insert a separate NP level that includes everything up to the QP:

```
(NP (DT the)
    (JJ last)
    (NN year)
    (QP (CC or) (RB so)))
```

```
(NP (CD one)
    (JJ new)
    (NN ringer)
    (QP (IN out) (IN of) (CD 10)))
```

The new structure is then:

```
(NP (NP (DT the) (JJ last) (NN year))
    (QP (CC or) (RB so)))
```

```
(NP (NP (CD one) (JJ new) (NN ringer))
    (QP (IN out) (IN of) (CD 10)))
```

We do not make this change if the QP is a purely numerical expression such as the following:

```
(NP (DT a)
    (JJ fixed)
    (QP (CD 107) (CD 3\/4)))
(NP (DT the)
    (VBG remaining)
    (QP (CD 2.2) (CD million)))
```

Similarly, there are cases where appositions are not recognized because there is an adjunct such as a `PP, SBAR, PRN` after the first NP:

```
(NP (NP (DT the) (NNP Center))
    (PP (IN for) (NP (NNP Security) (NNP Policy)))
    (, ,)
    (NP (DT a) (JJ conservative) (NNP Washington) (NN think-tank))
```

In these cases, we insert a `NP` which includes the `NP` and its adjunct.

If there is a `RB` followed by a `NP` within a `NP`, an `NP` which includes the `RB` and `NP`, is inserted.

These changes mean that only `NP`s appear under `NP`s if there is a conjunction, or an adjunct such as a relative clause or `PP` at the same level. There are still some cases of `NP`s under nouns, eg:

```
(NP (DT the)
    (NP (NNP Oct.) (CD 19) (, ,) (CD 1987))
    (NN market)
    (NN collapse))
```

These are treated as adjuncts.

B.5.2 Multiple NPs and modifiers

Similarly, if there are appositives and other modifiers such as relative clauses (`SBAR`), reduced relative clauses (`RRC`, `VP`) or `PP`s modifying the same `NP`, additional structure has to be imposed to obtain the correct analysis. There are three types of cases:

- `NP → NP, NP, MODIFIER (,)`
  The second NP is an appositive, and the modifier modifies the first NP. This becomes
  `NP → (NP, NP) , MODIFIER (,)`
- `NP → NP , NP MODIFIER`
  The modifier modifies the second NP. This becomes
  `NP → NP , (NP MODIFIER)`
- `NP → NP, MODIFIER , NP`
  Since appositives are analyzed like coordinate lists, merge the first NP and the modifier to an NP:
  `NP → (NP, MODIFIER) , NP`

B.5.3 Insertion of noun level:

There are a few cases where annotators inserted an NP as a noun level, but these are extremely rare, and we leave them as they are:

```
(NP (DT a)
    (NP (NP (JJ small)
            (, ,)
            (JJ tight)
            (JJ facial)
            (NN shot))
        (PP (IN of)
            (NP (NP (NNP David)
                    (NNP Dinkins))
                (, ,)
                (NP (NP (JJ Democratic)
                        (NN candidate))
                    (PP (IN for)
                        (NP (NP (NN mayor))
                            (PP (IN of)
                                (NP (NNP New)
                                    (NNP York)
                                    (NNP City)))))))))
(NP (DT the)
    (NP (JJ social)
        (NNS studies)
        (NN section)))
(NP (DT a)
    (NP (NNS cold-cuts)
        (NN buffet)))
```

NPnonBare**:** this is inserted in any NP whose first child is a PDT.

NPs with determiners/quantifiers: Insert a noun into a NP, NAC or NX after DT, POSS, POSSDT, QP, WDT, WRB, WP$ if they are not followed by a conjunction, a punctuation mark or a NPnonBare, or appear within a parallel conjunction ((XP XP CC XP) or parallel list ((XP XP , XP).

Inserting nouns into mass nouns, proper nouns, bare plurals: If a noun phrase does not have a determiner, and is not a personal or demonstrative pronoun, we insert a noun level immediately under the NP, so that the entire NP is also a noun. This applies to bare plurals, mass nouns and proper nouns.

Possessives: If a POSSDT (see possessives above) is followed by an NP, change the NP to a noun.

See below for the treatment of pied piping.

B.5.4 Other changes

- The Treebank analyzes some NPs that are modified by two SBARs as being modified by one SBAR which consists of these two SBAR. In order to obtain the correct CCG analysis, these cases are changed so that the NP is modified firectly by the two SBARs.

B.6 Preprocessing VPs

B.6.1 Coordinate VPs

- Sometimes, VP coordinations are not bracketed properly, so that the main verb of the first VP appears at the same level as the VP conjuncts, eg:

```
(VP (VBZ is)
    (VP restricted to his home much of the day)
    (CC and)
    (VP isn't allowed to work as a journalist))
```

In these cases, a VP is inserted which contains the verb and the following VP:

```
(VP (VP (VBZ is)
        (VP restricted to his home much of the day))
    (CC and)
    (VP isn't allowed to work as a journalist))
```

This can also happen with the verb of the second conjunct:

```
(VP (VP are very short-term)
    (CC and)
    (VBP are)
    (VP going to create high turnover))
```

This is changed in a similar way.

- Another type of bracketing error in coordinate VPs attaches the main verb of the first conjunct outside the entire conjunction, eg:

```
(VP (VBZ has)
    (VP (VP expanded its sale force to about 20 people from about 15)
        (CC and)
        (VP (VBZ hopes)
            (S to expand its sales ...))))
```

This error can be recognized by the fact that the verb in the second conjunct is tagged VBZ (other cases include VBP). This is changed to the correct bracketing:

```
(VP (VP (VP (VBZ has)
            (VP expanded its sale force to about 20 people from about 15)
        (CC and)
        (VP (VBZ hopes)
            (S to expand its sales ...))))
```

Similarly, coordination of modal verbs is sometimes not bracketed properly:

```
(VP (MD can)
    (CC and)
    (MD should)
    (VP be instituted throughout the industry))
```

We insert a VP that contains both modals. Since this VP preceeds the other VP, it is recognized as the head.

- VP coordination/lists:

```
VP --> VP, VP, ADV VP, VP
```

is rebracketed as

```
VP --> VP, VP, (ADV/RB VP), VP
```

Similarly,

```
VP --> ADVP/RB VP, VP...
```

is changed to

```
VP --> (ADVP/RB VP), VP...
```

even though there is a genuine ambiguity.

B.6.2 Other changes

- Sometimes, NPs in VPs aren't bracketed properly:

```
(VP (VB buy)
    (ADJP (RB closely)
          (VBN held))
    (NNS concerns))
```

- PP conjunctions within VP are sometimes not bracketed properly:

```
(VP (VB switch)
    (NP his retirement accounts)
    (PP-DIR out of three stock funds)
    (CC and)
    (PP-DIR into a money market fund))
```

In order to recognize the PP coordination, an extra PP is inserted:

```
(VP (VB switch)
    (NP his retirement accounts)
    (PP (PP-DIR out of three stock funds)
        (CC and)
        (PP-DIR into a money market fund))
```

- Ellipsis null elements (*?*) are cut out.

- If the last two children of a VP are ", NP" and there is another VP at the same level, the NP is out of construction:

```
(VP (VP (VBD fell)
        (NP-EXT  0.1 %)
        (PP-TMP in September))
    (, ,)
    (NP (NP the first decline)
        (PP-TMP since February 1987)))
```

We introduce a separate level XNP that includes the comma and the NP, which then triggers a binary type changing rule $(\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP}) \rightarrow ,\mathsf{NP}$

```
(VP (VP (VBD fell)
        (NP-EXT  0.1 %)
        (PP-TMP in September))
    (XNP (, ,)
         (NP (NP the first decline)
             (PP-TMP since February 1987)))
```

Sometimes the comma appears as last child of the VP child. This is also changed to the XNP structure.

B.7 Preprocessing QPs

The Treebank assumes a flat internal structure for QPs ("quantifier phrases").

Coordination in QPs If there is a conjunction within the QP, and the conjunction is not the first child, we assume that everything to the left and everything to the right of the conjunction is a QP. If there is a preposition (IN) before the conjunction (*between...and...*), then the first conjunct is only what comes between the preposition and the conjunction. Examples:

```
(QP (DT all) (CC but) ($ $) (CD 40,000))
(QP (IN between) (CD 1) (NN %) (CC and) (CD 3) (NN %))
```

Prepositions in QPs If there is one of the following prepositions within the QP, then everything that comes after the preposition is rebracketed as a NP, and a PP is inserted which consists of the preposition and the NP: *of, out, as, in, than, to*. Examples:

```
(QP (RBR more) (IN than) (CD 30))
(QP (RB Only) (CD five) (IN of) (DT the) (CD 40))
(QP (JJ as) (RB much) (IN as) (CD 15))
(QP (RB Only) (CD one) (IN in) (CD four))
```

These NPs are further preprocessed to insert nouns etc. The prepositions *to* and *in* are analyzed like conjunctions which take the left and right noun phrases as arguments. Their category is the category that is assigned to the QP.

***"as much as..."*:** The first *"as"* subcategorizes for the *"much"* and the following preposition.

***"well over..."*, etc.** If the first child of a QP is an adverb (RB), and the second is a preposition (IN, but not *from*), then the adverb modifies the preposition. Example:

```
(QP (RB well) (IN over) (CD 9))
```

***"nearly a third"*, etc:** If the first child of a QP is an adverb (RB), and the second is a preposition (IN, but not *from*), then the adverb modifies the entire QP. Examples:

```
(QP (RB nearly) (DT a) (JJ third))
```

B.8 Preprocessing RRCs

RRC is a label assigned to reduced relative clauses. However, a search with tgrep finds only 55 RRCs in the entire Treebank, since most reduced relative clauses are annotated differently. For instance, reduced relatives consisting of past or present participles should be annotated VP (Bies *et al.*, 1995, p. 230), eg.:

```
(NP (NP government figures)
   (VP (VBD released)
        (NP (-NONE- *) )
        (NP-TMP (NNP Wednesday) ))))))
```

Some instances of RRC expanding to VP can be found:

```
(NP (NP (NNS workers))
    (RRC (VP (VBN exposed)
             (NP (-NONE- *))
             (PP-CLR to it)
             (ADVP-TMP more than 30 years ago))))
```

These are re-labelled as VP.

There are also RRCs with a ADJP child:

```
 (NP (NP (NNS stores))
    (RRC (ADJP open)
         (NP-TMP more than one year)
```

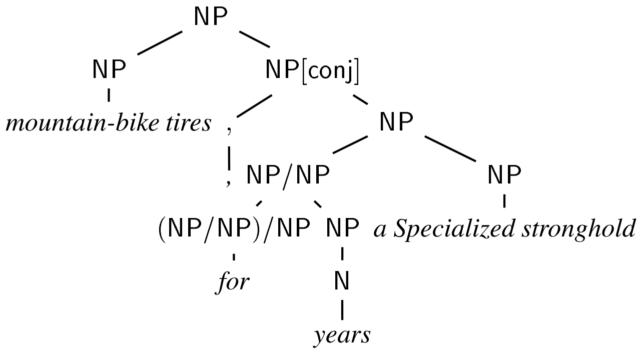
These are also re-labelled as ADJP, given that the same construction can appear elsewhere as ADJP:

```
(NP (NP (NNS funds))
    (ADJP (JJ open)
          (PP only to institutions)))
```

Appositives consisting of a sentential modifiers and a noun phrases, such as the following examples, are also annotated as RRC:

```
(NP (NP mountain-bike tires)
    (, ,)
    (RRC (PP-TMP for years)
         (NP (DT a Specialized stronghold))))
```

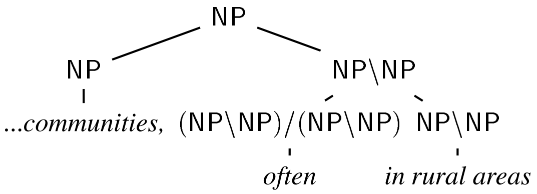
Given that in our grammar, appositives are analyzed like NP-coordination, we simply relabel these instances of RRC as NP; the sentential modifier is then treated like a premodifier of the second noun phrase:



Note that this leads to overgeneration in the grammar, since only appositive noun phrases can be modified in this manner.

Otherwise, if a RRC contains a PP child, it is re-labelled as PP. These are constructions consisting of a PP and a modifier, eg:

```
(NP (NP an estimated 92 communities))
    (, ,)
    (RRC (ADVP-TMP often)
         (PP-LOC (IN in)
                 (NP rural areas)))
```



B.9 Preprocessing SINVs

Elliptical inversion As described in section 3.5.7, we analyze elliptical inversion after certain function words like *so, than, as* by letting the function word subcategorize for an inverted, elliptical sentence. *Than* and *as* are analyzed as specifiers of SBAR:

```
(SBAR-ADV (IN as)
          (SINV (VBZ does)
                (NP-SBJ President Bush)
                (VP (-NONE- *?*))))
```

so appears as within an SINV:

```
(SINV (ADVP-TPC-1 (RB so))
      (VP (VBZ does)
          (ADVP (-NONE- *T*-1)))
      (NP-SBJ (NP almost everyone else))
              (PP-LOC in the book))
```

This analysis is modified so that *so* is also a specifier of an SBAR.

Verb fronting: As described in section 3.5.7, we analyze the subject NP as an object of the verb; therefore we replace the VP-trace with the NP-SBJ:

```
(SINV (VP-TPC-1 (VBG Following)
                (NP the feminist and population-control lead))
      (VP (VBZ has)
          (VP (VBN been)
              (VP (-NONE- *T*-1))))
      (NP-SBJ a generally bovine press)
      (. .))
```

This example becomes:

```
(SINV (VP-TPC-1 (VBG Following)
                (NP the feminist and population-control lead))
      (VP (VBZ has)
          (VP (VBN been)
              (NP a generally bovine press)))
      (. .))
```

B.10 Preprocessing SBARs

***"(Six months) before/after S"*:** The temporal NP receives a complement tag and is therefore analyzed as an argument of the subordinating conjunction.

Free relatives: Free relatives are analyzed as SBAR-NOM and receive category NP.

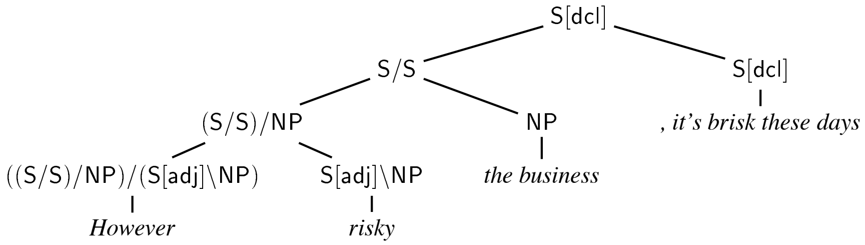
***so that*, *as though*, *as if* etc:** If an SBAR has two adjacent subordinating conjunctions as children (tagged as IN) next to a sentence S, another SBAR is inserted which consists of the second subordinating conjunction and the S.

***what about*, *what if*:** If an SBAR has a WP child that is immediately followed by IN, a new constituent is inserted which constists of the IN and its right sister. If the right sister is an NP (*what about...*), this new constituent is a PP-complement, if it is a S (*what if...*), the new constituent is a SBAR-complement.

***however/whatever*:** In SBARs with complementizer *whatever* or *however* that take a small clause argument where the second element is a trace, the trace is deleted:

```
(S (SBAR-ADV (WHADJP-1 (WRB However)
                       (JJ risky))
          (S (NP-SBJ the business)
             (ADJP-PRD (-NONE- *T*-1))))
   (, ,)
   (S it's brisk these days))
```

The CCG derivation is:



B.11 Preprocessing PPs

PPs under WHNPs: PPs under WHNP modify the noun under the first WHNP or NP, not the WHNP itself, eg:

```
(WHNP-1 (WHNP (WP what)
              (NN kind))
        (PP of initiative))))
```

This is rebracketed so that the PP modifies the noun *kind*. The following example is rebracketed so that the PP modifies the noun *opposition.*

```
(WHNP-3 (NP (WP$ whose)
            (NN opposition))
        (PP to foreign airline investment))))
```

PPs under NP do not carry the PRD tag.

If a PP has a WHNP child, it is relabelled as WHPP.

***"because of"*, *"instead of"*:** a PP with complement tag is inserted, which consists of the preposition *of* and the adjacent NP.

Within a PP, if a relative clause (an SBAR with WHNP in specifier position) appears which is adjacent to an NP, a new NP is inserted which consists of the NP and the relative clause.

B.12 Preprocessing UCPs

UCPs in small clauses: if a UCP-PRD appears within a small clause (S) where the subject NP is empty, cut out the empty subject and the S-node.

UCPs with more than three daughters: Most UCPs have three daughters: the two conjuncts and the conjunction. We use the following preprocessing steps in order to recognize the conjuncts in UCPs with more than two daughters:

- If the first daughter is DT, IN, or has the headword *either*, a new node is inserted that contains all but the first daughter. The label of this new node is determined by the label of the second daughter of the UCP: it is NOUN if the second daughter is a nominal POS-tag (NN etc.)

```
(UCP (DT both)
     (NN portfolio)
     (CC and)
     (JJ direct))
```

- We assume that ADVP, RB and PRN that follow immediately after a comma or conjunction and do not immediately precede a comma modifiy the element that follows after them, eg:

```
(UCP-PRD (PP in violation of Article II)
         (, ,)
         (CC and)
         (RB thus)
         (ADJP void and severable))
```

This is rebracketed as

```
(UCP-PRD (PP in violation of Article II)
         (, ,)
         (CC and)
         (ADJP (RB thus)
               (ADJP void and severable)))
```

If the first daughter of a UCP is RB with headword "not" or a CONJP, we also rebracket this in the same manner.

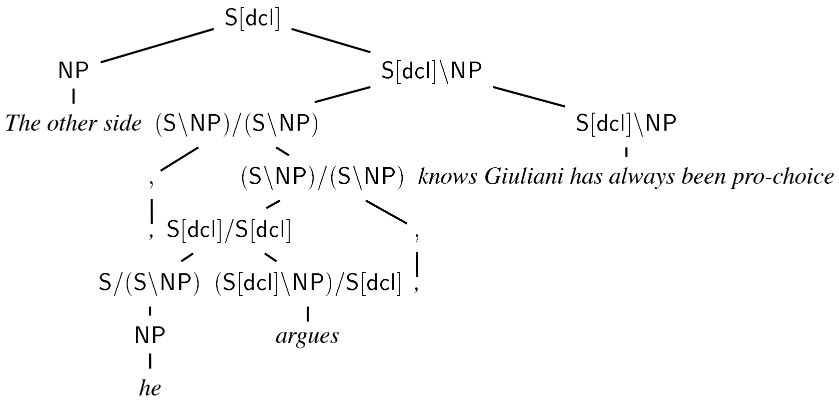
B.13 Preprocessing PRNs

Trees rooted in PRN: If a complete tree is rooted in PRN, its label is changed to the label of its first child, or to it second child if the first child is an opening parenthesis.

Sentences as PRNs: In direct speech, parentheticals indicate the speaker, eg.:

```
(PRN (, ,)
     (S (NP-SBJ (PRP he))
        (VP (VBZ argues)
            (SBAR (-NONE- 0)
                  (S (-NONE- *T*-1)))))
     (, ,))
```

This construction is analyzed with a unary type-changing rule:



If a colon is followed by a PRN, insert the colon into the parenthetical, and reanalyze the parenthetical.

PRNs that appear under coordinate constructions and that are adjacent to constituents that are conjuncts (have the same label as the parent node) are re-analyzed as children of this conjunct.

B.14 Preprocessing FRAGs

NP-SBJ **traces:** NP-SBJ null elements under FRAG are eliminated.

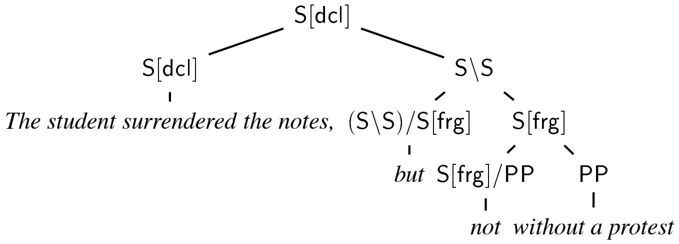
***"not"* is head:** in a FRAG, an adverb (RB) with head word *"not"* is head, unless it is preceded by WRB, WHADVP, NP:

```
(FRAG (RB Certainly) (RB not) (NP the lawyers) (. .))
(FRAG (CC And) (WHADVP why) (RB not) (. ?))
```

FRAGs that contain a wh-word are SBAR (if the FRAG is a child of a VP), or SBARQ.

If a conjunction (CC) is immediately followed by a FRAG, the conjunction is re-analyzed so that it takes the fragment as complement:

```
(TOP (S (S (NP-SBJ The student)
           (VP surrendered the notes))
        (, ,)
        (CC but)
        (FRAG (RB not)
              (PP without a protest))
        (. .)))
```



B.15 Preprocessing Xs

If the first child is NP, NN, NNP or SYM, then the entire constituent is relabelled as NP. If the headword of the first child is , relabel the X as X-ADV, so that it is recognized as an adjunct. If the first child is IN, TO or PP, the constituent is relabelled as PP. If the first child is DT and it is the only child, or if the last child is a NN or NNS, relabel the constituent as NP. Otherwise, if the parent of the node is SBAR and this node is its first child, relabel the X as S. Else, relabel it as FRAG.

B.16 Changes involving null elements

***T*-traces in subjects of *to*-VPs:** In some instances where the object of an object control verb such as *expect* is extracted, the trace appears as the subject of an S that contains the *to*-VP:

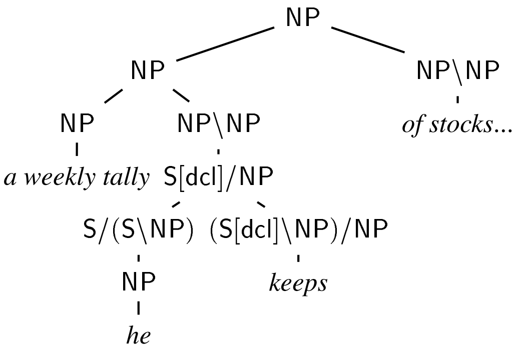
```
(NP (NP 70 businesses)
    (SBAR (WHNP-1 (IN that))
          (S (NP-SBJ (PRP it))
             (VP (VBD did)
                 (RB n't)
                 (VP (VB expect)
                     (S (NP-SBJ (-NONE- *T*-1))
                        (VP to produce 10% operating margins)))))))
```

In order for our trace-finding algorithm to work correctly, the NP-SBJ that contains the trace and the*to*-VP are moved up to the VP-level, and the S node is deleted, yielding the lexical category ((S[b]\NP)/(S[to]\NP))/NP for *expect.*

***T*-traces with adjuncts:** There are a few examples of NPs that expand to an NP with *T* trace as well as a modifier such as PP or SBAR.

```
(S (NP-SBJ One reason for his gloom)
   (VP (VBZ is)
       (NP-PRD (NP a weekly tally)
          (SBAR (WHNP-1 (-NONE- 0))
     (S (NP-SBJ (PRP he))
        (VP (VBZ keeps)
            (NP (NP (-NONE- *T*-1))
                (PP of stocks within a point of hitting new highs or lows)))))))
(. .)))
```

If, as in this example, the trace indicates object extraction in a relative clause, its modifier is lifted up to be an adjunct of the `NP` to which the relative clause is attached. The CCG derivation is as follows:



There are also cases of passive traces that are modified, eg.:

```
(VP (VBZ has)
    (VP (VBN been)
        (VP (VBN built)
            (NP (NP (-NONE- *-1))
                (SBAR-PRP (WHNP-2 (-NONE- 0))
                   (S (NP-SBJ (-NONE- *T*-2))
                      (VP to carry it through the transition period)))))))
```

Here the modifier (`SBAR-PRP`) is lifted up to the lowest `VP` level.

`*T*` traces within VPs containing a PP which contains a passive null element (`*`) or a "logical subject" (indicated by the `LGS` tag), are changed to `*` if the VP has no other child that expands to a `*` null element.

Eliminating `*?*` null elements (ellipsis): `*?*` null elements are ignored by the translation algorithm. If a constituent has only one child which is a `*?*` null element, it is eliminated. However, sometimes they are heads of constituents that have other, non-null elements, eg.:

```
(VP (VBP do)
    (VP (-NONE- *?*)
        (PP-LOC in South Carolina)))
```

In this case, the non-null elements (here the `PP-LOC`) are moved up one level in the tree, and the constituent which now contains only the null element is eliminated:

```
(VP (VBP do)
    (PP-LOC in South Carolina))
```

B.17 Other changes

”at least/most X” If *at* is immediately adjacent to a superlative adjective or adverb (eg. *least* or *most*), it subcategorizes for this adjective, which receives category S[asup]\NP.

Parentheses, dashes: for any constituent, if it has a sequence of three children of the form ‘‘LRB/dash XP RRB/dash’’, and it is not the case the XP and its parent are both NPs, let the parentheses or dashes be children of the XP.

As explained in section 3.7.4, we analyze parentheticals that consist of a dash followed by a constituent that is not normally an adjunct so that the dash and the adjacent constituent are an adjunct where the dash is “head” and the adjacent constituent its “argument”. We also follow this analysis for dashes and colons under S and VP.

Appendix C

Non-local dependencies projected by the lexicon

Here is a list of lexical categories that use the co-indexation mechanism described in section 2.5.7 to project long-range dependencies. Also given are the words that are found to have these lexical categories, followed by the frequency with which this lexical entry occurs in the corpus. This data is generated from sections 02-21 only. We classify the categories broadly into whether they project control and raising dependencies (this includes auxiliaries and modals) or extraction dependencies that arise in relative clauses and *tough*-movement. These lists include a number of spurious, but mostly low-frequency, categories, which are mostly due to annotation errors. All lexical categories of the same type have the same co-indexation, even though we believe that this ought to be done on a word-by-word basis.

C.1 Extraction dependencies

Relative pronouns:

- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[b]}/\mathsf{NP}_i)$ *that (1), which (1)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *– (2), -LRB- (2), : (12), as (1), besides (1), except (1), of (1), than (5), that (8), which (1), who (9)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}/\mathsf{NP}_i)$ *That (1), as (3), that (299), what (4), whatever (1), which (121), who (7), whom (18)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *- (2), – (1), -LRB- (1), : (3), That (1), WHO (2), What (1), as (2), brand (1), on (1), than (4), that (1957), what (2), which (1600), whichever (2), who (1484), whose (1)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *- (1), – (11), -LRB- (20), as (31), if (3), than (9), when (2)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}/\mathsf{NP}_i)$ *that (1)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[q]}/\mathsf{NP}_i)$ *Who (1)*
- $(\mathsf{NP}_i\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *– (1), -LCB- (1), : (5), How (2), as (2), how (1)*

Relative pronouns which subcategorize for a noun:

- $(((\mathsf{NP}\backslash\mathsf{NP})\backslash(\mathsf{NP}\backslash\mathsf{NP}))/(\mathsf{S[dcl]}\backslash\mathsf{N}_i))/\mathsf{N}_i$ *whose (1)*

- $((NP\backslash NP)/(S[dcl]/N_i))/N_i$ *whose (3)*
- $((NP\backslash NP)/(S[dcl]\backslash N_i))/N_i$ *which (2), whose (156)*
- $((S[qem]/S[wq])/(S[dcl]/N_i))/N_i$ *how (1)*
- $((S\backslash NP)/(S[dcl]\backslash N_i))/N_i$ *whose (1)*
- $((S/S)/(S[dcl]/N_i))/N_i$ *Whatever (2)*
- $((S/S)/(S[to]\backslash N_i))/N_i$ *In (3)*
- $((S\backslash S)/(S[to]\backslash N_i))/N_i$ *in (1)*
- $(N/(S[dcl]\backslash N_i))/N_i$ *which (1)*
- $(NP/(S[dcl]/N_i))/N_i$ *What (1), double (1), what (9), whatever (7)*
- $(NP/(S[dcl]\backslash N_i))/N_i$ *how (1), what (7), whatever (3), which (1)*
- $(NP/(S[to]\backslash N_i))/N_i$ *a (1), the (1)*
- $(S/(S[to]\backslash N_i))/N_i$ *in (3)*
- $(S[dcl]/(S[q]/N_i))/N_i$ *what (1)*
- $(S[qem]/(S[dcl]/N_i))/N_i$ *a (1), an (2), how (24), what (6), which (3), whichever (1), whose (3)*
- $(S[qem]/(S[dcl]\backslash N_i))/N_i$ *how (6), what (2), which (13), whichever (1)*
- $(S[wq]/(S[dcl]/N_i))/N_i$ *What (1), what (1)*
- $(S[wq]/(S[dcl]\backslash N_i))/N_i$ *How (1), Which (2)*
- $(S[wq]/(S[q]/N_i))/N_i$ *How (1), What (4), Which (1), the (1), what (4)*

Pied-piping relative pronouns:

- $((NP_i\backslash NP_i)/(S[dcl]\backslash NP))\backslash((NP\backslash NP)/NP_i)$ *what (1), whom (1)*
- $((NP_i\backslash NP_i)/(S[to]\backslash NP))\backslash((NP\backslash NP)/NP_i)$ *which (4)*

Free relative pronouns: *(do not pass a dependency)*

- $NP/(S[b]/NP)$ *what (1)*
- $NP/(S[b]\backslash NP)$ *to (1)*
- $NP/(S[dcl]/NP)$ *What (36), Whatever (1), what (296), whatever (5), who (1)*
- $NP/(S[dcl]\backslash NP)$ *What (32), what (144), whatever (1)*

Tough-movement adjectives:

- $(S[adj]\backslash NP_i)/((S[to]\backslash NP)/NP_i)$ *able (1), difficult (14), easier (8), easy (6), even (1), expensive (2), fairer (1), hard (11), harder (4), impossible (8), reasonable (1), tough (2), willing (1), wrong (1)*

Tough-movement verbs:

- $((S[b]\backslash NP_i)/((S[to]\backslash NP)/NP_i))/NP$ *cost (2)*
- $((S[dcl]\backslash NP_i)/((S[to]\backslash NP)/NP_i))/NP$ *cost (1), take (1)*
- $((S[ng]\backslash NP_i)/((S[to]\backslash NP)/NP_i))/NP$ *taking (1)*
- $((S[pt]\backslash NP_i)/((S[to]\backslash NP)/NP_i))/NP$ *got (1)*

Subject extraction verbs:

- $((\mathsf{S[b]}\backslash\mathsf{NP})/\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *realize (1), say (1)*
- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *agrees (1), alleged (1), argued (1), believe (9), believes (4), claimed (2), confirm (1), contends (2), estimate (1), estimated (1), expects (1), fear (2), feared (1), feel (1), hope (1), hopes (4), knew (1), read (1), said (24), say (11), says (11), suggested (1), think (7), thought (3)*
- $((\mathsf{S[pt]}\backslash\mathsf{NP})/\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *feared (1), hoped (2), said (2)*

C.2 Raising and control dependencies

Object control verbs:

- $((\mathsf{S[b]}\backslash\mathsf{NP})/(\mathsf{S[adj]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Call (1), Keep (1), Make (1), Sock (1), Spend (1), accept (1), add (1), be (1), benefit (1), bid (1), buy (1), call (1), cut (4), find (5), force (1), get (13), give (2), have (2), hit (1), hold (2), keep (30), knock (1), lead (1), leave (9), let (1), make (93), nudge (1), place (1), plow (1), prove (1), push (3), put (7), reduce (1), rein (1), run (1), see (1), sell (1), set (2), shake (1), start (1), steal (1), steer (1), sweeten (1), take (3), think (1), turn (6), walk (1), win (1), woo (1), wring (1)*
- $((\mathsf{S[b]}\backslash\mathsf{NP})/(\mathsf{S[b]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Help (1), Let (16), build (1), have (10), hear (1), help (42), let (45), make (32), see (21), tell (1), watch (2)*
- $((\mathsf{S[b]}\backslash\mathsf{NP})/(\mathsf{S[ng]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *appreciate (1), be (1), get (2), have (10), imagine (1), involve (1), justify (1), keep (13), leave (3), ruin (1), save (1), see (12), send (3), set (1), show (2), spend (7), take (2), tolerate (1), want (3), waste (2)*
- $((\mathsf{S[b]}\backslash\mathsf{NP})/(\mathsf{S[pss]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *find (1), get (2), have (9), hear (1), keep (3), leave (1), see (2), want (1), watch (1)*
- $((\mathsf{S[b]}\backslash\mathsf{NP})/(\mathsf{S[pt]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *get (5), have (7), keep (1), make (1), see (1)*
- $((\mathsf{S[b]}\backslash\mathsf{NP})/(\mathsf{S[to]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Allow (1), Use (1), abandon (1), advise (1), allow (71), allowed (1), arouse (1), ask (8), attract (1), authorize (1), be (4), become (1), bring (2), call (1), cause (11), challenge (1), chisel (1), compel (1), convince (2), cost (3), declare (1), dispatch (1), do (3), enable (19), encourage (8), entitle (3), expect (23), find (1), force (27), free (2), get (18), go (1), have (1), help (3), induce (3), inspire (1), invest (1), invite (1), lead (3), like (4), make (2), modify (1), motivate (1), need (5), offer (1), order (1), pay (1), permit (10), persuade (19), prod (2), prompt (3), push (3), rally (1), release (1), require (25), send (1), sensitize (1), set (1), show (1), spend (2), spur (1), take (18), teach (4), tell (5), urge (7), use (48), want (6), wish (1)*
- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/(\mathsf{S[adj]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Make (1), bid (1), brought (2), call (2), called (27), calls (4), consider (6), considered (3), considers (1), created (1), cut (1), declared (2), deemed (1), deems (1), dragged (1), drives (2), drove (1), earned (1), filed (1), find (8), finds (4), flattened (1), forced (1), found (10), get (2), gets (1), has (2), hold (1), keep (1), keeps (1), kept (4), kicked (1), labeled (2), leave (2), leaves (5), left (2), made (18), make (16), makes (31), moved (1), nursed (1), outnumbered (1), proclaimed (1), pushed (3), pushes (1), put (8), puts (2), rates (1), remanded (1), resembles (1), sent (3), set (1), sets (2), showed (1), snatched (1), take (2), termed (3), thought (1), took (4), tugged (1), turns (1), want (2), was (2), wish (1), wishes (1), won (1)*
- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/(\mathsf{S[b]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Make (1), adds (1), felt (2), had (1), hear (1), heard (2), help (2), helped (16), helps (6), let (19), lets (6), made (11), make (6), makes (11), said (2), saw (14), watched (4)*

- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/(\mathsf{S[ng]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *'s (1), began (1), busies (1), covers (1), devotes (1), displayed (1), envision (1), expects (1), fear (1), featured (1), find (3), finds (1), found (6), had (6), has (5), have (8), hear (1), heard (2), keeps (1), kept (1), leave (2), leaves (1), left (1), necessitated (1), posted (1), recalls (1), reported (1), saw (4), see (10), sees (1), sent (12), show (1), showed (3), shows (3), spend (3), spends (3), spent (14), spotted (1), took (1), wore (1)*
- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/(\mathsf{S[pss]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *finds (2), found (3), had (2), has (1), have (2), hear (1), is (1), observed (1), ordered (2), see (2), spent (2), wanted (2), wants (2), was (1)*
- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/(\mathsf{S[pt]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *got (1), had (2), have (1), paid (1), see (1), slashed (1), want (1)*
- $((\mathsf{S[dcl]}\backslash\mathsf{NP})/(\mathsf{S[to]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *'s (1), EXPECT (1), FORCE (1), adapted (1), added (2), advise (2), advised (2), advises (1), allow (13), allowed (21), allows (24), amended (1), appointed (1), ask (2), asked (32), asks (3), authorized (3), believes (1), brought (1), call (1), called (2), cause (2), caused (16), causes (6), cautioned (2), charge (2), chose (1), commissioned (1), consider (4), considers (1), construed (1), convinced (1), costs (2), declared (1), defined (1), defy (1), did (2), directed (3), dispatched (1), drove (1), elected (1), empowers (1), enabled (5), enables (5), encourage (3), encouraged (4), encourages (1), entitle (1), entitles (1), expect (68), expected (17), expects (95), extends (1), flew (1), flies (1), forbade (1), force (4), forced (10), forces (5), found (3), free (1), freed (1), frees (1), gained (2), galvanized (1), gave (1), get (1), got (1), has (1), helped (1), hired (2), induced (1), inspired (1), instructed (2), instructs (1), introduced (1), invested (1), invited (3), invites (3), is (1), judge (2), lead (4), leads (3), led (10), left (2), meant (2), moves (1), named (2), need (5), needed (2), needs (4), offset (1), ordered (17), pays (1), permit (1), permits (4), permitted (2), persuaded (6), prepare (1), pressure (1), priced (1), prompted (9), prompts (1), rallied (1), require (5), required (9), requires (10), retained (1), selected (1), set (1), shed (2), showed (1), signed (1), spent (1), spurred (1), structured (1), take (3), takes (2), taunted (1), teaches (1), tell (1), tells (3), tempt (1), tempts (1), told (7), took (6), train (1), trains (1), trusted (1), understood (1), urge (1), urged (22), urges (6), use (10), used (24), uses (9), want (19), wanted (3), wants (14), warned (1)*
- $((\mathsf{S[ng]}\backslash\mathsf{NP})/(\mathsf{S[adj]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Calling (1), Holding (1), Keeping (2), Making (1), Putting (1), bidding (1), bleeding (1), bringing (1), burying (1), calling (7), declaring (1), driving (2), finding (3), getting (4), holding (4), inviting (1), keeping (15), knocking (1), leaving (4), making (28), moving (3), plying (1), pulling (1), punching (1), pushing (1), putting (3), rendering (2), setting (1), slicing (1), taking (2), tugging (1), turning (3), wishing (1)*
- $((\mathsf{S[ng]}\backslash\mathsf{NP})/(\mathsf{S[b]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Letting (1), Watching (1), having (8), helping (8), letting (11), making (9), seeing (3), watching (7)*
- $((\mathsf{S[ng]}\backslash\mathsf{NP})/(\mathsf{S[ng]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *adopting (1), having (5), keeping (2), leaving (3), seeing (1), sending (1), setting (1), showing (2), signing (1), spending (3), surviving (1), watching (2)*
- $((\mathsf{S[ng]}\backslash\mathsf{NP})/(\mathsf{S[pss]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Having (1), driving (1), getting (1), having (5), keeping (1), leaving (2), seeing (1)*
- $((\mathsf{S[ng]}\backslash\mathsf{NP})/(\mathsf{S[pt]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *getting (2), having (2), leaving (1)*
- $((\mathsf{S[ng]}\backslash\mathsf{NP})/(\mathsf{S[to]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *Allowing (1), Amending (1), Commissioning (1), Getting (1), Persuading (1), Urging (1), abetting (1), admonishing (1), advising (3), aiding (1), allowing (28), asking (11), believing (1), broadening (1), causing (6), cautioning (1), counseling (1), directing (1), driving (1), enabling (7), encouraging (3), entitling (1), expanding (1), expecting (5), forbidding (3), forcing (12), getting (5), helping (1), hiring (1), imploring (1), instructing (1), inviting (1), leading (1), leaving (1), lobbying (1), naming (1), needing (1), ordering (4), permitting (4), persuading (2), picking (1), prepping (1), pressuring (3), prodding (1), prompting*

(7), pushing (2), requiring (12), restructuring (1), spending (3), taking (3), telling (4), training (1), urging (17), using (34), warning (2)

- $((\mathsf{S[pt]}\backslash\mathsf{NP})/(\mathsf{S[adj]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *believed (1), brought (1), called (2), found (1), got (1), gotten (2), kept (4), left (3), made (18), played (1), proved (1)*
- $((\mathsf{S[pt]}\backslash\mathsf{NP})/(\mathsf{S[b]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *heard (1), helped (3), let (1), made (3), seen (9), watched (2)*
- $((\mathsf{S[pt]}\backslash\mathsf{NP})/(\mathsf{S[ng]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *been (1), called (1), done (1), filed (2), had (4), left (2), opposed (1), seen (1), sent (2), spent (3)*
- $((\mathsf{S[pt]}\backslash\mathsf{NP})/(\mathsf{S[pss]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *got (1), had (2), heard (2), left (1)*
- $((\mathsf{S[pt]}\backslash\mathsf{NP})/(\mathsf{S[to]}\backslash\mathsf{NP}_i))/\mathsf{NP}_i$ *advised (1), allowed (10), asked (13), called (1), cast (1), caused (2), done (1), enabled (3), encouraged (3), expected (5), forced (4), found (1), hired (2), instructed (1), invited (3), led (3), left (1), lobbied (1), mobilized (1), modified (1), moved (1), ordered (2), permitted (1), persuaded (2), positioned (1), preferred (1), pressed (1), prodded (1), prompted (2), required (2), sent (2), shown (2), signed (1), steered (1), told (2), tried (1), urged (4), used (15), waited (1), warned (1), wished (1)*

Auxiliaries, modals and raising verbs:

- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *Be (1), Get (1), Go (1), Make (1), Put (1), Rest (1), Step (1), accelerate (1), appear (13), are (1), back (2), band (1), be (850), become (36), begin (1), bode (1), bounce (1), bow (1), break (4), buzz (1), change (1), close (2), come (16), comprise (1), continue (1), cut (2), decline (1), do (11), drift (1), dry (1), earn (1), end (1), expire (1), fall (5), feel (18), fetch (1), find (1), finish (2), fly (1), get (43), glaze (1), go (28), grow (3), have (1), hold (1), hum (1), keep (3), level (1), lie (1), look (17), make (10), mean (1), move (3), open (2), pass (1), plead (2), pop (1), prove (11), pull (3), read (2), remain (41), report (1), rise (2), roll (1), run (2), say (3), seem (8), settle (1), shift (1), sink (1), sit (3), sound (3), split (1), stay (23), step (3), take (2), think (4), trend (1), turn (3), walk (1)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *Do (2), GET (1), Go (1), Please (1), be (6), can (1), come (1), could (1), dare (1), do (6), get (3), go (1), help (97), may (2), respond (1), say (4), start (1), to (2)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *DON'T (1), be (1), help (1), say (1), spot (1)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *Consider (1), Try (1), advocate (1), authorize (1), avoid (16), be (204), begin (45), bother (1), cease (1), come (1), consider (12), contemplate (2), continue (36), delay (2), deny (1), discuss (1), end (11), enjoy (2), favor (1), get (1), go (4), hate (1), help (4), hesitate (1), include (3), involve (4), justify (3), keep (10), like (2), mean (2), mind (3), permit (1), prefer (2), recall (1), recommend (2), reconsider (2), relish (1), replace (1), require (1), resist (1), resume (2), risk (4), rule (2), see (1), stand (1), start (21), stop (21), try (4), wind (2)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *Be (1), Get (1), are (2), be (1764), become (4), get (19), go (1), have (2), stay (1)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *'ve (2), Get (1), Have (1), feel (1), get (8), has (1), have (429)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *act (1), afford (12), agree (8), appear (14), attempt (7), be (22), begin (8), bid (1), bother (1), choose (4), combine (1), continue (97), dare (1), decide (3), do (1), elect (1), endeavor (1), expect (10), fail (5), get (3), go (2), gon (1), grow (2), happen (1), hate (1), have (170), help (4), hesitate (4), hope (3), intend (6), know (1), learn (4), like (32), listen (1), love (4), manage (3), meet (1), move (5), need (19), offer (4), plan (8), pop (1), prefer (7), prepare (1), proceed (1), promise (1), prove (4), refuse (5), return (1), rush (1), say (1), seek (20), seem (25), serve (3), set (1), sought (1), start (7), stay (1), stick (1), struggle (1), tend (6), threaten (1), try (67), turn (6), undertake (1), vote (4), wait (5), want (70), wish (1), work (6)*

- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *’S (1), ’m (55), ’re (77), ’s (190), Are (3), Fires (1), IS (1), Is (2), Sounds (1), Steps (1), Was (1), act (1), acts (1), am (19), appear (10), appeared (8), appears (9), are (888), be (1), became (43), become (10), becomes (14), bounce (1), bounced (1), broke (3), came (9), closed (52), come (6), comes (3), continues (2), cool (1), cooled (1), costs (1), counts (1), crawls (1), curled (1), dates (1), did (6), do (3), does (1), drifted (1), edged (5), ended (39), ends (1), expires (1), falls (4), fare (1), feel (11), feels (2), fell (15), felt (11), finished (17), finishes (1), flew (1), fly (1), get (5), gets (9), go (6), goes (14), got (15), grew (4), has (1), hit (1), holds (2), increased (1), is (1403), joined (1), kept (1), know (1), lashed (1), live (1), look (12), looked (5), looks (9), looms (1), lost (1), makes (2), melts (1), move (1), moved (2), multipled (1), narrowed (1), obscures (1), opened (3), picks (1), pleaded (11), proved (6), proves (3), pulled (3), ran (1), ranked (1), ranks (1), reached (1), remain (31), remained (30), remains (63), replied (1), report (1), returned (1), rose (3), runs (1), s (1), said (3), scuttled (1), see (1), seem (16), seemed (11), seems (30), sees (1), sell (1), set (1), settled (3), settles (1), shine (1), slipped (1), smells (1), snapped (1), sounded (1), sounds (5), split (2), stand (1), stands (2), stay (2), stayed (2), steered (1), stepped (2), steps (1), stood (1), stopped (1), stops (1), stretches (1), surged (1), survived (1), test (1), thrashed (1), took (3), traded (5), turn (2), turned (5), turns (3), was (715), went (27), were (504), wore (1), yelled (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *’d (69), ’ll (115), ’s (3), Ca (1), Can (2), Could (1), Did (1), Do (10), HAS (1), May (1), Should (1), WILL (2), Would (1), announced (1), announces (1), are (1), ca (184), can (863), chanted (1), could (1032), dare (2), did (415), do (520), does (381), got (1), had (2), has (2), help (9), helped (52), helps (13), is (18), let (1), may (772), might (328), mighta (1), must (238), need (8), said (5), say (3), says (6), shall (23), should (394), was (3), were (1), will (2944), wo (230), would (2089)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *’d (2), advises (1), ca (1), can (1), could (1), deem (1), did (1), do (1), has (1), have (9), helped (3), hovered (1), may (2), said (4), say (1), says (2), start (1), that (1), think (1), will (2), would (3)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[em]}\backslash\mathsf{NP}_i)$ *is (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *’m (43), ’re (190), ’s (103), ARE (1), Means (1), admits (1), admitted (2), ai (1), am (8), anticipate (1), anticipated (1), anticipates (4), are (1004), avoid (2), avoided (2), avoids (1), been (1), began (71), begin (6), begins (5), came (2), chance (1), confirm (1), consider (1), considered (7), continue (2), continued (3), delayed (1), denied (2), denies (2), deny (2), described (1), disclaims (1), discussed (1), end (1), ended (2), endorsed (1), ends (1), enjoy (2), enjoys (1), favor (4), favors (3), fears (1), gave (1), gets (1), go (2), hates (1), include (4), included (1), includes (5), inhibit (1), involve (2), involved (3), involves (2), is (1298), keep (4), keeps (3), kept (12), lies (1), like (1), loves (1), mean (1), means (7), meant (1), missed (1), originated (1), pioneered (1), ponder (1), prohibits (1), proposed (4), proposes (3), quit (1), recalls (2), recommend (1), recommended (1), recounts (1), regrets (1), remembered (1), reported (3), requires (1), resembles (1), resisted (1), resumed (2), risk (2), risked (2), risks (3), spent (2), stand (1), start (7), started (38), starts (2), stop (2), stopped (14), suggested (3), suggests (1), tried (1), was (313), went (2), were (225), wind (2), wound (2)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *’m (6), ’re (6), ’s (15), Is (1), WAS (1), am (4), and (1), appeared (2), are (854), be (3), became (3), been (5), climbed (1), felt (2), get (8), gets (7), got (6), had (8), has (1), have (3), helped (1), is (1166), remains (1), rose (1), seem (3), seemed (1), seems (1), sounds (1), was (1495), were (748), will (1), wree (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *’d (17), ’m (5), ’re (9), ’s (75), ’ve (174), HAS (2), IS (1), ai (1), became (4), could (4), did (4), does (2), feel (1), feels (2), felt (4), get (1), gets (3), got (5), had*

(1017), has (2512), have (1421), helped (1), might (2), remained (1), remains (1), seems (2), should (1)

- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *'s (1), Expects (1), FAILED (1), Helps (1), PLANS (1), Seems (1), act (1), agree (3), agreed (165), aim (1), aimed (1), aims (10), allowed (1), appear (17), appeared (25), appears (38), are (15), arranged (1), asked (14), aspire (1), attempt (6), attempted (5), attempts (5), awoke (1), began (18), begin (3), begins (6), came (4), care (1), ceased (1), charged (1), choose (3), chooses (2), chose (5), claim (2), claims (4), combine (1), consented (1), conspire (1), conspired (5), continue (38), continued (65), continues (63), contracted (2), contributed (1), dared (1), decide (6), decided (37), decides (8), decline (5), declined (122), declines (12), deserve (1), deserves (1), designed (1), determined (2), dived (1), dove (1), eased (1), edged (1), elected (2), exercise (2), expect (18), expected (13), expects (104), fail (4), failed (66), fails (6), fell (19), fights (1), figures (3), filed (1), flocked (1), fought (1), get (1), gets (2), go (1), goes (4), got (6), had (58), happen (2), happened (2), happens (5), has (48), hate (3), have (83), help (1), helped (5), helps (2), hesitate (2), hope (17), hoped (5), hopes (40), intend (9), intended (12), intends (44), is (136), jumped (2), learn (3), like (9), likes (4), lobbied (1), looks (1), love (1), loves (1), manage (3), managed (17), manages (2), mean (1), meant (3), met (1), moved (7), moves (1), need (40), needed (8), needs (15), offered (12), opt (1), opted (1), ought (32), plan (36), planned (24), plans (166), pledged (4), plummeted (2), plunged (1), prefer (5), prefers (2), prepared (4), prepares (4), proceeds (2), professed (1), professes (1), promise (1), promised (7), promises (9), proposed (5), proposes (2), proved (1), purport (1), purports (1), pushed (1), rebound (1), recovered (1), refuse (8), refused (21), refuses (6), remain (3), remains (3), resolved (1), rose (13), rush (1), rushed (6), say (2), scramble (3), scrambled (9), scurries (1), seek (5), seeks (11), seem (29), seemed (19), seems (49), served (2), serves (1), set (6), sets (1), signed (1), soared (2), sought (8), spreads (1), stand (2), stands (4), start (2), started (8), starts (2), stood (4), strive (1), strove (1), struggled (4), struggles (5), sufficed (1), surged (2), tend (36), tended (3), tends (11), threaten (1), threatened (10), threatens (5), tried (36), tries (16), try (27), tumbled (1), turn (2), turned (5), turns (4), unites (1), used (27), volunteered (2), vote (1), voted (25), vowed (11), vows (1), wait (1), waited (1), waits (1), wake (1), want (107), wanted (52), wants (72), was (55), went (5), were (21), wish (5), wishes (2), worked (6), wrote (1)*
- $(\mathsf{S[em]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *will (1), would (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *Being (1), Keeping (1), Looking (4), Punching (1), becoming (15), being (38), blinking (1), breaking (2), buying (1), clanging (1), closing (1), coming (4), cruising (1), doing (7), drifting (1), edging (1), eroding (1), falling (5), feeling (4), fighting (1), finishing (1), flaring (1), flying (1), following (1), getting (33), going (8), growing (11), hanging (1), holding (1), inching (2), keeping (1), leading (1), looking (5), lying (1), making (4), meaning (1), moving (1), opening (1), pleading (1), prancing (1), proving (1), rearing (1), remaining (3), riding (1), running (4), settling (1), sitting (3), slowing (1), sounding (1), staggering (1), standing (1), staying (1), steaming (1), steering (1), stepping (1), talking (1), tapering (1), tottering (1), trading (2), trailing (2), trending (1), turning (2), winding (1), yielding (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *being (2), chanting (1), helping (7), saying (2), trying (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *meaning (1), saying (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *beginning (1), considering (24), contemplating (1), exploring (1), finishing (1), regarding (1), ruling (2), rushing (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *Being (1), being (334), getting (13)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *Having (12), getting (2), having (38)*

- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *Having (1), Hoping (2), SEEKING (1), Seeking (2), Trying (3), about (16), acting (2), agreeing (10), aiming (2), appearing (2), applying (1), ascending (1), asking (1), attempting (24), beginning (34), bothering (1), choosing (1), claiming (3), closing (1), conspiring (6), continuing (15), deciding (2), declining (2), easing (1), expecting (1), failing (19), gearing (1), going (202), gon (4), having (12), helping (6), hoping (14), jumping (1), learning (2), leaving (1), lobbying (2), looking (7), moving (6), negotiating (3), offering (9), opting (2), planning (15), pledging (1), preferring (3), preparing (9), pressing (1), proceeding (1), promising (5), proposing (4), proving (2), racing (2), refusing (7), remaining (1), rushing (7), scrambling (7), seeking (33), setting (1), sitting (1), starting (16), striving (2), struggling (9), taking (1), threatening (10), toiling (1), trying (175), turning (1), voting (2), vowing (3), waiting (14), wanting (5), willing (1), wishing (1), working (8)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *Carried (1), Rated (16), are (2), believed (3), blown (2), booked (1), bought (1), broken (1), brought (1), called (2), carved (1), caught (1), come (1), considered (26), converted (1), crowded (1), cut (1), declared (5), deemed (5), feared (1), filed (1), found (9), gunned (1), held (6), invited (1), is (1), jammed (1), judged (1), kept (1), labeled (1), left (3), located (1), made (20), opened (2), paid (2), passed (2), peeled (1), pegged (1), phased (1), plowed (1), pressured (1), presumed (1), priced (2), proved (1), proven (1), published (1), pulled (2), pushed (1), put (5), rated (34), rendered (2), reported (1), revised (1), scaled (1), sealed (1), set (1), shut (1), spread (1), stitched (1), stretched (1), stripped (1), taken (3), tangled (1), termed (2), thought (5), thrown (2), tilted (1), torn (2), turned (5), was (1), watered (1), wired (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *does (1), should (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *caught (3), found (1), heard (1), left (1), quoted (1), seated (1), seen (4), spent (2), tied (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *are (3), believed (1), discovered (1), found (1), is (1), left (2), ordered (1), reported (1), was (3)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *had (3), have (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *Asked (2), URGED (1), adjusted (1), agreed (1), alleged (3), allowed (32), amazed (1), appalled (1), appointed (2), approached (1), asked (22), assigned (2), authorized (3), believed (18), born (1), bound (3), built (1), called (1), cast (1), chosen (2), claimed (1), cleared (2), compelled (3), conditioned (1), considered (5), constructed (1), counted (1), created (3), designated (3), designed (48), destined (1), determined (2), discovered (1), disinclined (1), disposed (1), elected (1), employed (1), encouraged (5), engineered (1), entitled (2), equipped (1), estimated (4), expanded (1), expected (288), flattened (1), forced (52), formed (1), found (3), freed (1), held (1), hired (5), inclined (4), inflated (1), instructed (1), intended (25), invited (5), known (6), led (2), left (4), made (2), meant (16), modified (1), motivated (1), named (5), needed (21), obligated (3), obliged (2), ordered (6), paid (2), perceived (3), permitted (6), picked (1), planned (1), pleased (1), poised (3), positioned (3), prepared (13), pressured (2), priced (55), programmed (2), projected (4), provided (1), put (1), raised (1), reached (4), refocused (1), reported (2), reputed (1), required (29), restated (1), restructured (1), retained (1), rumored (7), said (25), scheduled (59), seen (2), selected (3), sent (1), set (11), shown (5), signed (1), slated (10), solicited (1), stopped (1), structured (1), stunned (1), supposed (33), surprised (1), tapped (1), targeted (3), taught (1), tempted (1), thought (14), timed (1), told (4), trained (1), understood (1), urged (2), used (83), vexed (1)*
- $(\mathsf{S[pt]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *averaged (1), become (36), been (242), broken (3), changed (1), closed (1), come (4), done (1), fallen (4), gone (10), gotten (5), grown (5), held (2), kept (1), moved (2), pleaded (2), proved (6), remained (3), retreated (1), run (2), sat (2), seemed (1),*

sounded (1), stalled (1), stayed (2), stopped (1), surged (1), taken (2), turned (6), voted (1), was (1), were (1), worked (1)

- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *been (1), did (1), helped (13), told (1), will (1), would (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[ng]\backslash NP}_i)$ *admitted (1), avoided (2), been (331), begun (11), considered (2), discussed (1), finished (1), involved (1), proposed (4), regretted (1), reported (1), started (11), stopped (9), suggested (1), targeted (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[pss]\backslash NP}_i)$ *Been (1), are (1), become (4), been (640), done (1), gotten (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[pt]\backslash NP}_i)$ *has (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[to]\backslash NP}_i)$ *REQUIRED (1), acted (1), afforded (1), agreed (63), asked (1), attempted (2), been (7), begun (13), believed (2), chose (2), chosen (4), climbed (1), come (6), committed (1), compelled (2), continued (5), decided (8), declined (2), destined (1), elected (3), emerged (1), expected (6), failed (16), forced (1), forgotten (1), gone (2), got (13), grown (2), guaranteed (1), had (9), helped (4), hoped (4), hurried (1), impelled (1), intended (1), intervened (1), learned (1), legitimized (1), liked (1), loved (1), managed (6), materialized (1), moved (4), needed (1), obligated (1), obliged (1), offered (6), planned (4), pledged (1), poised (1), prepared (1), promised (4), proposed (1), proved (1), pushed (1), refused (10), resolved (1), rung (1), rushed (3), seemed (4), served (2), set (2), sought (5), started (6), striven (1), struggled (1), sued (1), supposed (3), sworn (1), taken (1), tended (2), threatened (2), tried (15), used (1), voted (2), waited (3), wanted (2), worked (1)*
- $(\mathsf{S[to]\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *TO (3), To (124), na (5), to (12824)*
- $(\mathsf{S[to]\backslash NP}_i)/(\mathsf{S[ng]\backslash NP}_i)$ *to (3)*
- $(\mathsf{S\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *could (1)*

Verbs that take a VP or adjectival complement:

- $(\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[adj]\backslash NP}_i)$ *Be (1), Get (1), Go (1), Make (1), Put (1), Rest (1), Step (1), accelerate (1), appear (12), are (1), back (2), band (1), be (850), become (36), begin (1), bode (1), bounce (1), bow (1), break (4), buzz (1), change (1), close (2), come (15), comprise (1), continue (1), cut (2), decline (1), do (11), drift (1), dry (1), earn (1), end (1), expire (1), fall (5), feel (18), fetch (1), find (1), finish (2), fly (1), get (43), glaze (1), go (28), grow (3), have (1), hold (1), hum (1), keep (3), level (1), lie (1), look (17), make (10), mean (1), move (3), open (2), pass (1), plead (2), pop (1), prove (11), pull (3), read (2), remain (41), report (1), rise (2), roll (1), run (2), say (3), seem (8), settle (1), shift (1), sink (1), sit (3), sound (3), split (1), stay (23), step (3), take (2), think (4), trend (1), turn (3), walk (1)*
- $(\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *Do (2), GET (1), Go (1), Please (1), be (6), can (1), come (1), could (1), dare (1), do (6), get (3), go (1), help (97), may (2), respond (1), say (4), start (1), to (2)*
- $(\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[dcl]\backslash NP}_i)$ *DON'T (1), be (1), help (1), say (1), spot (1)*
- $(\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[ng]\backslash NP}_i)$ *Consider (1), Try (1), advocate (1), authorize (1), avoid (15), be (204), begin (44), bother (1), cease (1), come (1), consider (12), contemplate (2), continue (36), delay (2), deny (1), discuss (1), end (11), enjoy (2), favor (1), get (1), go (4), hate (1), help (4), hesitate (1), include (3), involve (4), justify (3), keep (10), like (2), mean (2), mind (3), permit (1), prefer (2), recall (1), recommend (2), reconsider (2), relish (1), replace (1), require (1), resist (1), resume (2), risk (4), rule (2), see (1), stand (1), start (21), stop (21), try (4), wind (2)*
- $(\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[pss]\backslash NP}_i)$ *Be (1), Get (1), are (2), be (1761), become (4), get (19), go (1), have (2), stay (1)*

- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *'ve (2), Get (1), Have (1), feel (1), get (8), has (1), have (429)*
- $(\mathsf{S[b]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *act (1), afford (12), agree (8), appear (15), attempt (7), be (22), begin (8), bid (1), bother (1), choose (4), combine (1), continue (97), dare (1), decide (3), do (1), elect (1), endeavor (1), expect (10), fail (5), get (3), go (2), gon (1), grow (2), happen (1), hate (1), have (170), help (4), hesitate (4), hope (3), intend (6), know (1), learn (4), like (32), listen (1), love (4), manage (3), meet (1), move (5), need (19), offer (4), plan (8), pop (1), prefer (7), prepare (1), proceed (1), promise (1), prove (4), refuse (5), return (1), rush (1), say (1), seek (20), seem (25), serve (3), set (1), sought (1), start (7), stay (1), stick (1), struggle (1), tend (6), threaten (1), try (67), turn (6), undertake (1), vote (4), wait (5), want (70), wish (1), work (6)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *'S (1), 'm (55), 're (77), 's (191), Are (3), Fires (1), IS (1), Is (2), Sounds (1), Steps (1), Was (1), act (1), acts (1), am (19), appear (10), appeared (8), appears (9), are (888), be (1), became (43), become (10), becomes (14), bounce (1), bounced (1), broke (3), came (9), closed (52), come (6), comes (3), continues (2), cool (1), cooled (1), costs (1), counts (1), crawls (1), curled (1), dates (1), did (6), do (3), does (1), drifted (1), edged (5), ended (39), ends (1), expires (1), falls (4), fare (1), feel (11), feels (2), fell (15), felt (11), finished (17), finishes (1), flew (1), fly (1), get (5), gets (9), go (6), goes (14), got (15), grew (4), has (1), hit (1), holds (2), increased (1), is (1405), joined (1), kept (1), know (1), lashed (1), live (1), look (12), looked (5), looks (9), looms (1), lost (1), makes (2), melts (1), move (1), moved (2), multipled (1), narrowed (1), obscures (1), opened (3), picks (1), pleaded (11), proved (6), proves (3), pulled (3), ran (1), ranked (1), ranks (1), rated (1), reached (1), remain (31), remained (30), remains (63), replied (1), report (1), returned (1), rose (3), runs (1), s (1), said (3), scuttled (1), see (1), seem (16), seemed (11), seems (30), sees (1), sell (1), set (1), settled (3), settles (1), shine (1), slipped (1), smells (1), snapped (1), sounded (1), sounds (5), split (2), stand (1), stands (2), stay (2), stayed (2), steered (1), stepped (2), steps (1), stood (1), stopped (1), stops (1), stretches (1), surged (1), survived (1), test (1), thrashed (1), took (3), traded (5), turn (2), turned (5), turns (3), was (715), went (27), were (503), wore (1), yelled (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *'d (69), 'll (118), 's (3), Ca (1), Can (2), Could (1), Did (1), Do (10), HAS (1), May (1), WILL (2), Would (1), announced (1), announces (1), are (1), ca (185), can (867), chanted (1), could (1031), dare (2), did (414), do (519), does (382), got (1), had (2), has (2), help (9), helped (52), helps (13), is (18), let (1), may (783), might (329), mighta (1), must (240), need (8), said (5), say (3), says (6), shall (24), should (394), was (3), were (1), will (2968), wo (237), would (2096)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *'d (2), advises (1), ca (1), can (1), could (1), deem (1), did (1), do (1), has (1), have (9), helped (3), hovered (1), may (2), said (4), say (1), says (2), start (1), that (1), think (1), will (2), would (3)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[em]}\backslash\mathsf{NP}_i)$ *is (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *'m (43), 're (189), 's (106), ARE (1), Means (1), admits (1), admitted (2), ai (1), am (8), anticipate (1), anticipated (1), anticipates (4), are (1007), avoid (2), avoided (2), avoids (1), been (1), began (71), begin (6), begins (5), came (2), chance (1), confirm (1), consider (1), considered (7), continue (2), continued (3), delayed (1), denied (2), denies (2), deny (2), described (1), disclaims (1), discussed (1), end (1), ended (2), endorsed (1), ends (1), enjoy (2), enjoys (1), favor (4), favors (3), fears (1), gave (1), gets (1), go (2), hates (1), include (4), included (1), includes (5), inhibit (1), involve (2), involved (3), involves (2), is (1298), keep (4), keeps (3), kept (12), lies (1), like (1), loves (1), mean (1), means (7), meant (1), missed (1), originated (1), pioneered (1), ponder (1), prohibits (1), proposed (4), proposes (3), quit (1), recalls (2), recommend (1), recommended (1), recounts (1), regrets (1), remembered (1), reported (3), requires (1), resembles (1), resisted (1), resumed (2), risk (2), risked (2), risks*

(3), spent (2), stand (1), start (7), started (38), starts (2), stop (2), stopped (14), suggested (3), suggests (1), tried (1), was (312), went (2), were (225), wind (2), wound (2)

- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *'m (6), 're (6), 's (15), Is (1), WAS (1), am (4), appeared (2), are (853), be (3), became (3), been (5), climbed (1), felt (2), get (8), gets (7), got (6), had (8), has (1), have (3), helped (1), is (1165), remains (1), rose (1), seem (3), seemed (1), seems (1), sounds (1), was (1493), were (749), will (1), wree (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *'d (17), 'm (5), 're (9), 's (77), 've (174), HAS (2), IS (1), ai (1), became (4), could (4), did (4), does (2), feel (1), feels (2), felt (4), get (1), gets (3), got (5), had (1020), has (2533), have (1440), helped (1), might (2), remained (1), remains (1), seems (2), should (1)*
- $(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *'s (1), Expects (1), FAILED (1), Helps (1), PLANS (1), Seems (1), act (1), agree (3), agreed (165), aim (1), aimed (1), aims (10), allowed (1), appear (17), appeared (25), appears (40), are (15), arranged (1), asked (14), aspire (1), attempt (6), attempted (5), attempts (5), awoke (1), began (18), begin (3), begins (6), came (4), care (1), ceased (1), charged (1), choose (3), chooses (2), chose (5), claim (2), claims (4), combine (1), consented (1), conspire (1), conspired (5), continue (38), continued (65), continues (64), contracted (2), contributed (1), dared (1), decide (6), decided (37), decides (8), decline (5), declined (122), declines (12), deserve (1), deserves (1), determined (2), dived (1), dove (1), eased (1), edged (1), elected (2), exercise (2), expect (16), expected (13), expects (104), fail (4), failed (66), fails (6), fell (19), fights (1), figures (3), filed (1), flocked (1), fought (1), get (1), gets (2), go (1), goes (4), got (6), had (58), happen (2), happened (2), happens (5), has (49), hate (3), have (83), help (1), helped (5), helps (2), hesitate (2), hope (17), hoped (5), hopes (40), intend (9), intended (12), intends (44), is (136), jumped (2), learn (3), like (9), likes (4), lobbied (1), looks (1), love (1), loves (1), manage (3), managed (17), manages (2), mean (1), meant (3), met (1), moved (7), moves (1), need (40), needed (8), needs (15), offered (12), opt (1), opted (1), ought (33), plan (36), planned (24), plans (166), pledged (4), plummeted (2), plunged (1), prefer (5), prefers (2), prepared (4), prepares (4), proceeds (2), professed (1), professes (1), promise (1), promised (7), promises (9), proposed (5), proposes (2), proved (1), purport (1), purports (1), pushed (1), rebound (1), recovered (1), refuse (8), refused (21), refuses (6), remain (3), remains (3), resolved (1), rose (13), rush (1), rushed (6), say (2), scramble (3), scrambled (9), scurries (1), seek (5), seeks (11), seem (29), seemed (19), seems (52), served (2), serves (1), set (6), sets (1), signed (1), soared (2), sought (8), spreads (1), stand (2), stands (4), start (2), started (8), starts (2), stood (4), strive (1), strove (1), struggled (4), struggles (5), sufficed (1), surged (2), tend (36), tended (3), tends (11), threaten (1), threatened (10), threatens (5), tried (36), tries (16), try (27), tumbled (1), turn (2), turned (5), turns (4), unites (1), used (27), volunteered (2), vote (1), voted (25), vowed (11), vows (1), wait (1), waited (1), waits (1), wake (1), want (107), wanted (52), wants (72), was (55), went (5), were (21), wish (5), wishes (2), worked (6), wrote (1)*
- $(\mathsf{S[em]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *will (1), would (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *Being (1), Keeping (1), Looking (4), Punching (1), becoming (15), being (38), blinking (1), breaking (2), buying (1), clanging (1), closing (1), coming (4), cruising (1), doing (7), drifting (1), edging (1), eroding (1), falling (5), feeling (4), fighting (1), finishing (1), flaring (1), flying (1), following (1), getting (33), going (8), growing (11), hanging (1), holding (1), inching (2), keeping (1), leading (1), looking (5), lying (1), making (4), meaning (1), moving (1), opening (1), pleading (1), prancing (1), presumed (1), proving (1), rearing (1), remaining (3), riding (1), running (4), settling (1), sitting (3), slowing (1), sounding (1), staggering (1), standing (1), staying (1), steaming (1), steering (1), stepping (1), talking (1), tapering (1), tottering (1), trading (2), trailing (2), trending (1), turning (2), winding (1), yielding (1)*

- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *being (2), chanting (1), did (1), helping (7), saying (2), trying (1), will (2)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[dcl]}\backslash\mathsf{NP}_i)$ *meaning (1), saying (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *beginning (1), considering (24), contemplating (1), exploring (1), finishing (1), regarding (1), ruling (2), rushing (1)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *Being (1), being (332), getting (13)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *Having (12), getting (2), having (38)*
- $(\mathsf{S[ng]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *Having (1), Hoping (2), SEEKING (1), Seeking (2), Trying (3), about (16), acting (2), agreeing (10), aiming (2), appearing (2), applying (1), ascending (1), asking (1), attempting (24), beginning (34), bothering (1), choosing (1), claiming (3), closing (1), conspiring (6), continuing (15), deciding (2), declining (2), easing (1), expecting (1), failing (19), gearing (1), going (202), gon (4), having (12), helping (6), hoping (13), jumping (1), learning (2), leaving (1), lobbying (2), looking (7), moving (6), negotiating (3), offering (9), opting (2), planning (15), pledging (1), preferring (3), preparing (9), pressing (1), proceeding (1), promising (5), proposing (4), proving (2), racing (2), refusing (7), remaining (1), rushing (7), scrambling (7), seeking (33), set (1), setting (1), sitting (1), starting (16), striving (2), struggling (9), taking (1), threatening (10), toiling (1), trying (175), turning (1), voting (2), vowing (3), waiting (14), wanting (5), willing (1), wishing (1), working (8)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[adj]}\backslash\mathsf{NP}_i)$ *Carried (1), Rated (16), are (2), believed (3), blown (2), booked (1), bought (1), broken (1), brought (1), called (2), carved (1), caught (1), come (1), considered (26), converted (1), crowded (1), cut (1), declared (5), deemed (5), feared (1), filed (1), found (9), gunned (1), held (6), invited (1), is (1), jammed (1), judged (1), kept (1), labeled (1), left (3), located (1), made (20), opened (2), paid (2), passed (2), peeled (1), pegged (1), phased (1), plowed (1), pressured (1), priced (2), proved (1), proven (1), published (1), pulled (2), pushed (1), put (5), rated (34), rendered (2), reported (1), revised (1), scaled (1), sealed (1), set (1), shut (1), spread (1), stitched (1), stretched (1), stripped (1), taken (3), tangled (1), termed (2), thought (5), thrown (2), tilted (1), torn (2), turned (5), was (1), watered (1), wired (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[b]}\backslash\mathsf{NP}_i)$ *does (1), should (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[ng]}\backslash\mathsf{NP}_i)$ *caught (3), found (1), heard (1), left (1), quoted (1), seated (1), seen (4), spent (2), tied (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[pss]}\backslash\mathsf{NP}_i)$ *are (3), being (1), believed (1), discovered (1), found (1), is (1), left (2), ordered (1), reported (1), was (3)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[pt]}\backslash\mathsf{NP}_i)$ *had (3), have (1)*
- $(\mathsf{S[pss]}\backslash\mathsf{NP}_i)/(\mathsf{S[to]}\backslash\mathsf{NP}_i)$ *Asked (2), URGED (1), adjusted (1), agreed (1), alleged (3), allowed (32), amazed (1), appalled (1), appointed (2), approached (1), asked (22), assigned (2), authorized (3), believed (18), born (1), bound (3), built (1), called (1), cast (1), chosen (2), claimed (1), cleared (2), compelled (3), conditioned (1), considered (5), constructed (1), counted (1), created (3), designated (3), designed (48), destined (1), determined (2), discovered (1), disinclined (1), disposed (1), elected (1), employed (1), encouraged (5), engineered (1), entitled (2), equipped (1), estimated (4), expanded (1), expected (288), flattened (1), forced (52), formed (1), found (3), freed (1), held (1), hired (5), inclined (4), inflated (1), instructed (1), intended (25), invited (5), known (6), led (2), left (4), made (2), meant (16), modified (1), motivated (1), named (5), needed (21), obligated (3), obliged (2), ordered (6), paid (2), perceived (3), permitted (6), picked (1), planned (1), pleased (1), poised (3), positioned (3), prepared (13), pressured (2), priced (55), programmed (2), projected (4), provided (1), put (1), raised (1), reached (4), refocused (1), reported (2), reputed (1), required (29), restated (1), restructured (1), retained (1),*

rumored (7), said (25), scheduled (59), seen (2), selected (3), sent (1), set (11), shown (5), signed (1), slated (10), solicited (1), stopped (1), structured (1), stunned (1), supposed (33), surprised (1), tapped (1), targeted (3), taught (1), tempted (1), thought (14), timed (1), told (4), trained (1), understood (1), urged (2), used (83), vexed (1)

- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[adj]\backslash NP}_i)$ *averaged (1), become (36), been (242), broken (3), changed (1), closed (1), come (4), done (1), fallen (4), gone (10), gotten (5), grown (5), held (2), kept (1), moved (2), pleaded (2), proved (6), remained (3), retreated (1), run (2), sat (2), seemed (1), sounded (1), stalled (1), stayed (2), stopped (1), surged (1), taken (2), turned (6), voted (1), was (1), were (1), worked (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *been (1), did (1), helped (13), told (1), will (1), would (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[ng]\backslash NP}_i)$ *admitted (1), avoided (2), been (331), begun (11), considered (2), discussed (1), finished (1), involved (1), proposed (4), regretted (1), reported (1), started (11), stopped (9), suggested (1), targeted (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[pss]\backslash NP}_i)$ *Been (1), are (1), become (4), been (640), done (1), gotten (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[pt]\backslash NP}_i)$ *has (1)*
- $(\mathsf{S[pt]\backslash NP}_i)/(\mathsf{S[to]\backslash NP}_i)$ *REQUIRED (1), acted (1), afforded (1), agreed (63), asked (1), attempted (2), been (7), begun (13), believed (2), chose (2), chosen (4), climbed (1), come (6), committed (1), compelled (2), continued (5), decided (8), declined (2), destined (1), elected (3), emerged (1), expected (6), failed (15), forced (1), forgotten (1), gone (2), got (13), grown (2), guaranteed (1), had (9), helped (4), hoped (4), hurried (1), impelled (1), intended (1), intervened (1), learned (1), legitimized (1), liked (1), loved (1), managed (6), materialized (1), moved (4), needed (1), obligated (1), obliged (1), offered (6), planned (4), pledged (1), poised (1), prepared (1), promised (4), proposed (1), proved (1), pushed (1), refused (10), resolved (1), rung (1), rushed (3), seemed (4), served (2), set (1), sought (5), started (6), striven (1), struggled (1), sued (1), supposed (3), sworn (1), taken (1), tended (2), threatened (2), tried (15), used (1), voted (2), waited (3), wanted (2)*
- $(\mathsf{S[to]\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *TO (3), To (124), na (5), to (12806)*
- $(\mathsf{S[to]\backslash NP}_i)/(\mathsf{S[ng]\backslash NP}_i)$ *to (3)*
- $(\mathsf{S\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i)$ *could (1)*
- $(((\mathsf{S[dcl]\backslash NP}_i)/(\mathsf{S[adj]\backslash NP}_i))/\mathsf{PP})/\mathsf{PP}$ *opened (1)*
- $(((\mathsf{S[dcl]\backslash NP}_i)/(\mathsf{S[to]\backslash NP}_i))/\mathsf{PP})/(\mathsf{S[adj]\backslash NP}_i)$ *was (1), were (1)*
- $((\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[adj]\backslash NP}_i))/\mathsf{PP}$ *look (1), settle (1)*
- $((\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[b]\backslash NP}_i))/\mathsf{PP}$ *live (1)*
- $((\mathsf{S[b]\backslash NP}_i)/(\mathsf{S[to]\backslash NP}_i))/\mathsf{PP}$ *be (1), rely (5), wait (1), work (4)*
- $((\mathsf{S[b]\backslash NP}_i)/\mathsf{PP})/(\mathsf{S[adj]\backslash NP}_i)$ *bode (1), close (1), come (2), cut (2), do (3), emerge (1), feel (1), focus (1), get (2), go (5), jump (1), make (1), plead (5), press (1), start (1), take (1), think (1)*
- $((\mathsf{S[b]\backslash NP}_i)/\mathsf{S[dcl]})/(\mathsf{S[adj]\backslash NP}_i)$ *make (5)*
- $((\mathsf{S[b]\backslash NP}_i)/\mathsf{S[em]})/(\mathsf{S[adj]\backslash NP}_i)$ *be (2), make (2), rule (1)*
- $((\mathsf{S[b]\backslash NP}_i)/\mathsf{S[em]})/(\mathsf{S[to]\backslash NP}_i)$ *prove (1)*
- $((\mathsf{S[b]\backslash NP}_i)/\mathsf{S[for]})/(\mathsf{S[adj]\backslash NP}_i)$ *be (14), wait (1)*
- $((\mathsf{S[b]\backslash NP}_i)/\mathsf{S[qem]})/(\mathsf{S[adj]\backslash NP}_i)$ *be (1), care (1)*

- $((S[b]\backslash NP_i)/(S[ng]\backslash NP_i))/(S[adj]\backslash NP_i)$ *be (1), end (2), keep (1)*
- $((S[b]\backslash NP_i)/(S[to]\backslash NP_i))/(S[adj]\backslash NP_i)$ *be (16), become (1), cost (1), do (1), seem (1)*
- $((S[dcl]\backslash NP_i)/(S[adj]\backslash NP_i))/PP$ *closed (17), fattened (1), grew (1), settled (1)*
- $((S[dcl]\backslash NP_i)/(S[ng]\backslash NP_i))/PP$ *'re (1)*
- $((S[dcl]\backslash NP_i)/(S[ng]\backslash NP_i))/S[em]$ *said (1)*
- $((S[dcl]\backslash NP_i)/(S[to]\backslash NP_i))/PP$ *agreed (2), called (6), calls (1), contracted (1), converged (1), ended (2), joined (1), merged (1), pleaded (1), relies (2), rely (1), splits (1), went (1), were (1)*
- $((S[dcl]\backslash NP_i)/PP)/(S[adj]\backslash NP_i)$ *are (1), came (1), closed (16), comes (3), ended (4), finished (3), gets (2), got (3), is (2), look (1), looked (1), peck (1), pleaded (4), sat (1), settled (3), start (1), stopped (1), think (1), wake (1), was (17), went (1), were (1), woke (1)*
- $((S[dcl]\backslash NP_i)/PP)/(S[ng]\backslash NP_i)$ *began (1), regards (1)*
- $((S[dcl]\backslash NP_i)/PP)/(S[to]\backslash NP_i)$ *is (1)*
- $((S[dcl]\backslash NP_i)/S[dcl])/(S[adj]\backslash NP_i)$ *become (1), makes (2), yelled (1)*
- $((S[dcl]\backslash NP_i)/S[em])/(S[adj]\backslash NP_i)$ *is (1), made (2), make (1), turned (1), voted (1)*
- $((S[dcl]\backslash NP_i)/S[for])/(S[adj]\backslash NP_i)$ *is (1)*
- $((S[dcl]\backslash NP_i)/S[qem])/(S[adj]\backslash NP_i)$ *'s (1), is (1)*
- $((S[dcl]\backslash NP_i)/S)/(S[adj]\backslash NP_i)$ *is (1)*
- $((S[dcl]\backslash NP_i)/S)/(S[to]\backslash NP_i)$ *opted (1), wants (1)*
- $((S[dcl]\backslash NP_i)/(S[adj]\backslash NP_i))/(S[adj]\backslash NP_i)$ *came (1), is (1), traded (1), were (1)*
- $((S[dcl]\backslash NP_i)/(S[b]\backslash NP_i))/(S[adj]\backslash NP_i)$ *could (1), may (1)*
- $((S[dcl]\backslash NP_i)/(S[ng]\backslash NP_i))/(S[adj]\backslash NP_i)$ *are (1), feel (1)*
- $((S[dcl]\backslash NP_i)/(S[to]\backslash NP_i))/(S[adj]\backslash NP_i)$ *'re (1), ended (1), has (3), have (1), is (1), served (1), set (1), turn (1), was (2), were (1)*
- $((S[dcl]\backslash NP_i)/((S[to]\backslash NP)/NP))/(S[adj]\backslash NP_i)$ *take (1)*
- $((S[ng]\backslash NP_i)/(S[adj]\backslash NP_i))/PP$ *closing (1)*
- $((S[ng]\backslash NP_i)/(S[b]\backslash NP_i))/PP$ *whispering (2)*
- $((S[ng]\backslash NP_i)/(S[to]\backslash NP_i))/PP$ *appealing (1), counting (2), depending (1), negotiating (2), relying (1), waiting (1), working (1)*
- $((S[ng]\backslash NP_i)/PP)/(S[adj]\backslash NP_i)$ *Looking (1), buying (1), chipping (1), closing (2), ending (1), fighting (1), getting (1), going (3), holding (1), pressing (1), steering (1), thrashing (1)*
- $((S[ng]\backslash NP_i)/PP)/(S[to]\backslash NP_i)$ *going (1)*
- $((S[ng]\backslash NP_i)/S[dcl])/(S[adj]\backslash NP_i)$ *making (1)*
- $((S[ng]\backslash NP_i)/S[em])/(S[adj]\backslash NP_i)$ *making (2)*
- $((S[ng]\backslash NP_i)/(S[adj]\backslash NP_i))/(S[adj]\backslash NP_i)$ *leaving (1)*
- $((S[ng]\backslash NP_i)/(S[to]\backslash NP_i))/(S[adj]\backslash NP_i)$ *getting (1)*
- $((S[ng]\backslash NP_i)/(S[to]\backslash NP_i))/(S[to]\backslash NP_i)$ *going (1)*
- $((S[pss]\backslash NP_i)/(S[adj]\backslash NP_i))/PP$ *sold (1)*
- $((S[pss]\backslash NP_i)/(S[to]\backslash NP_i))/PP$ *designed (1), offered (2), organized (1), priced (64), quoted (6), scheduled (1), used (1)*

- $((S[pss]\backslash NP_i)/PP)/(S[adj]\backslash NP_i)$ *brought (1), pieced (1), piled (1), put (1), rolled (1), set (1), spread (1), taken (1), thrown (1), whittled (1)*
- $((S[pss]\backslash NP_i)/S[qem])/(S[to]\backslash NP_i)$ *forced (1)*
- $((S[pss]\backslash NP_i)/(S[to]\backslash NP_i))/(PP/NP)$ *called (3)*
- $((S[pt]\backslash NP_i)/(S[to]\backslash NP_i))/PP$ *depended (1), signed (1)*
- $((S[pt]\backslash NP_i)/PP)/(S[adj]\backslash NP_i)$ *appeared (1), been (1), brushed (1), pleaded (1)*
- $((S[pt]\backslash NP_i)/S[em])/(S[adj]\backslash NP_i)$ *become (2), been (1), made (1)*
- $((S[pt]\backslash NP_i)/S[qem])/(S[adj]\backslash NP_i)$ *left (1)*
- $((S[pt]\backslash NP_i)/(S[ng]\backslash NP_i))/(S[adj]\backslash NP_i)$ *been (2)*
- $((S[pt]\backslash NP_i)/(S[to]\backslash NP_i))/(S[adj]\backslash NP_i)$ *banded (1), been (4), seemed (1)*
- $((S[to]\backslash NP_i)/(S[b]\backslash NP_i))/(S[b]\backslash NP_i)$ *to (1)*
- $((S[to]\backslash NP_i)/(S[to]\backslash NP_i))/(S[b]\backslash NP_i)$ *to (1)*
- $((S[to]\backslash NP_i)\backslash(S[adj]\backslash NP_i))/(S[b]\backslash NP_i)$ *to (1)*
- $(S[dcl]\backslash NP_i)\backslash(S[adj]\backslash NP_i)$ *adjusted (1)*
- $(S[ng]\backslash NP_i)\backslash(S[adj]\backslash NP_i)$ *beginning (1), declining (1)*
- $(S[pss]\backslash NP_i)\backslash(S[adj]\backslash NP_i)$ *designated (1)*

VP-modifiers with VP or adjectival argument:

- $((S\backslash NP_i)/(S\backslash NP_i))/(S[adj]\backslash NP_i)$ *– (1), -LRB- (1), In (1), although (2), by (1), for (2), in (5), on (1), since (1), slightly (1), though (4), until (1), whether (1), while (7)*
- $((S\backslash NP_i)/(S\backslash NP_i))/(S[b]\backslash NP_i)$ *– (1), than (1)*
- $((S\backslash NP_i)/(S\backslash NP_i))/(S[dcl]\backslash NP_i)$ *as (2)*
- $((S\backslash NP_i)/(S\backslash NP_i))/(S[ng]\backslash NP_i)$ *– (1), Without (1), after (4), by (4), in (7), on (1), though (1), upon (1), while (8)*
- $((S\backslash NP_i)/(S\backslash NP_i))/(S[pss]\backslash NP_i)$ *– (1), -LRB- (1), as (9), if (8), once (1), though (3)*
- $((S\backslash NP_i)/(S\backslash NP_i))/(S[pt]\backslash NP_i)$ *had (1)*
- $((S\backslash NP_i)/(S\backslash NP_i))/(S[to]\backslash NP_i)$ *– (1)*
- $((S\backslash NP_i)/(S\backslash NP_i))\backslash(S[adj]\backslash NP_i)$ *than (29)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[adj]\backslash NP_i)$ *– (24), -LCB- (2), -LRB- (3), about (1), albeit (1), although (2), around (1), as (10), at (9), between (1), by (8), for (16), from (48), if (15), in (14), like (3), of (2), possibly (1), than (7), their (1), though (1), through (1), to (3), until (5), where (1), wherever (1)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[b]\backslash NP_i)$ *– (5), -LRB- (5), than (3), to (6)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[dcl]\backslash NP_i)$ *– (2), : (1), as (7), on (1), so (1), that (34), what (1), which (59), who (18)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[ng]\backslash NP_i)$ *- (1), – (18), -LRB- (1), By (1), In (1), When (2), Without (3), about (7), after (108), around (1), as (2), at (2), before (62), beyond (1), by (402), despite (4), for (50), from (17), if (1), in (135), including (3), into (3), like (1), of (6), on (2), over (2), since (3), than (5), through (1), thus (1), to (10), toward (4), upon (3), when (11), while (71), with (2), without (70)*

- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[pss]\backslash NP_i)$ *– (1), as (34), if (6), than (2), unless (6), until (1), when (11), while (1)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[pt]\backslash NP_i)$ *- (1), have (1)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))/(S[to]\backslash NP_i)$ *– (5), -LRB- (1), as (3), enough (8), except (4), if (1), likely (1), so (1), sufficient (1), sufficiently (2), than (5), threatens (1)*
- $((S\backslash NP_i)\backslash(S\backslash NP_i))\backslash(S[adj]\backslash NP_i)$ *than (4)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))/((S\backslash NP)\backslash(S\backslash NP_i)))/(S[adj]\backslash NP_i)$ *as (1), from (1)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))/((S\backslash NP)\backslash(S\backslash NP_i)))/(S[dcl]\backslash NP_i)$ *as (1)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))/((S\backslash NP)\backslash(S\backslash NP_i)))/(S[ng]\backslash NP_i)$ *by (1), from (1)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))/((S\backslash NP)\backslash(S\backslash NP_i)))\backslash(S[adj]\backslash NP_i)$ *than (5)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))\backslash((S\backslash NP)\backslash(S\backslash NP_i)))/(S[adj]\backslash NP_i)$ *– (1), as (1), for (1), from (2), than (9)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))\backslash((S\backslash NP)\backslash(S\backslash NP_i)))/(S[b]\backslash NP_i)$ *than (1)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))\backslash((S\backslash NP)\backslash(S\backslash NP_i)))/(S[dcl]\backslash NP_i)$ *as (2), than (2), that (1), which (1), whichever (1)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))\backslash((S\backslash NP)\backslash(S\backslash NP_i)))/(S[ng]\backslash NP_i)$ *before (1), than (1)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))\backslash((S\backslash NP)\backslash(S\backslash NP_i)))/(S[pss]\backslash NP_i)$ *as (3), than (7)*
- $(((S\backslash NP)\backslash(S\backslash NP_i))\backslash((S\backslash NP)\backslash(S\backslash NP_i)))/(S[to]\backslash NP_i)$ *as (1)*

Yes/no questions:

- $(S[q]/(S[adj]\backslash NP_i))/NP_i$ *Are (3), Is (4), Was (1), Were (1), is (6), was (2)*
- $(S[q]/(S[b]\backslash NP_i))/NP_i$ *Can (5), Could (4), Did (4), Do (10), Does (8), Must (1), SHOULD (1), Should (7), Will (5), Would (4), ca (1), can (12), could (5), did (11), do (18), does (20), might (1), must (1), should (11), will (9), would (14)*
- $(S[q]/(S[dcl]\backslash NP_i))/NP_i$ *Will (1), had (2), has (1), should (1), was (1)*
- $(S[q]/(S[ng]\backslash NP_i))/NP_i$ *'s (3), Are (1), Is (5), Was (1), am (1), are (3), is (5)*
- $(S[q]/(S[pss]\backslash NP_i))/NP_i$ *ARE (1), Is (1)*
- $(S[q]/(S[pt]\backslash NP_i))/NP_i$ *Has (1), Have (2), has (2), have (1)*
- $(S[q]/(S[to]\backslash NP_i))/NP_i$ *'s (1)*

Appendix D

File formats

There are three sets of files which mirror the directory and file structure of the Penn Treebank: the human-readable files in HTML format, the machine-readable corpus files (`*.auto`), and the predicate-argument structure files (`*.parg`).

For the "yy"th file in section "xx", "wsj_xxyy.mrg", there is a corresponding human-readable HTML file "wsj_xxyy.html" in the directory tree under `data/HTML/`, a derivation file "wsj_xxyy.auto" under `data/AUTO/` and a predicate-argument structure file "wsj_xxyy.parg" under `data/PARG/`.

The distribution also includes two lexicon files (extracted from sections 02-21 and 00), and a file that contains the entire corpus (sections 00-24) in a format that can be read by Douglas Rohde's TGrep2 search tool.

D.1 The human-readable corpus files

The human-readable corpus is a set of HTML files, corresponding to each file in the original Penn Treebank distribution. These files are provided to allow easy inspection of the derivations and the corresponding predicate-argument structure. For each sentence that could be translated successfully to a CCG derivation, the syntactic derivation tree itself is shown as a pretty-printed bracketed string, followed by the list of bilexical dependencies in the predicate-argument structure (not shown here):

```
Sentence 2

{S[dcl] {S[dcl] {NP {N {N/N  Mr.}
                  {N  Vinken}}}
          {S[dcl]\NP {(S[dcl]\NP)/NP  is}
                  {NP {NP {N  chairman}}
                      {NP\NP {(NP\NP)/NP  of}
                          {NP {NP {N {N/N  Elsevier}
                                     {N  N.V.}}}
                              {NP[conj] {,  ,}
                                   {NP {NP[nb]/N  the}
                                       {N {N/N  Dutch}
                                          {N {N/N  publishing}
                                             {N  group}}}}}}}}}}
   {.  .}
```

The format of the word-word dependencies in the predicate-argument gives first the head word of the functor, followed by its lexical category and the head word of any filled arguments of this functor. Which argument slot is filled by which word is indicated by appropriate colors.

Sentences for which no CCG derivation could be produced are indicated as such.

D.2 The machine-readable derivation files

The machine-readable derivation files contain the syntactic derivations in a format that is designed to be read in automatically. They do not indicate the word-word dependencies in the predicate-argument structure. Each sentence appears on one line, preceded by one line which identifies the sentence:

```
ID=wsj0001.1 PARSER=GOLD NUMPARSE=1
(<T S[dcl] 0 2> (<T S[dcl] 1 2> (<T NP 0 2> ...) ))
```

The sentence ID consists of the original Penn Treebank file name, followed by the number of the sentence in this file. Each node is indicated by parentheses "(" and ")" and a description, which follows the left parenthesis. The node description itself is delimited by angled brackets `‘‘<X...>’’`, where `X` is `L` for leaf nodes and `T` for other nodes. In leaf nodes, the description contains six fields:

```
<L CCGcat mod_POS-tag orig_POS-tag word PredArgCat>
```

The original POS tag is the tag assigned to this word in the Penn Treebank. The modified POS tag might differ from this tag if it was changed during the translation to CCG. `PredArgCat` is another representation of the lexical category (`CCGcat`) which encodes the underlying predicate-argument structure (described in more detailed below). The node description of a leaf node is also enclosed in parentheses:

```
 (<L N NN NN chairman N>)
```

`PredArgCat`, the last field of the node description of leaf nodes, is designed to encode the word-word dependencies in the underlying predicate-argument structure. This is a simplified version of the predicate-argument structure representation presented in section 2.5. Recall that complex categories are recursive structures that consist of a result and an argument category. Each category has a head index, such that the identity of the lexical heads of two arguments can be indicated by giving them the same head index. This mechanism (explained in more detail in section 2.5) is used to indicate non-local dependencies that are mediated through lexical items such as relative pronouns or control verbs.

The head of a lexical category is simply the word itself, and in complex categories, the head of the result is the same as the head of the entire category (with the exception of determiners NP/N, where the lexical head of the NP is the same as the head of the N. Since we only annotate lexical categories with their head indices, the head index for the entire category is omitted. Similarly, the head index of any part of a complex category is omitted if it is identical to the head index of the entire category.

- For atomic categories (N, etc.), the head index is not indicated, since it is simply the word itself: `<L N NNP NNP Vinken N>`

- In ordinary complex categories that do not mediate any non-local dependencies (S[dcl]\NP)/NP etc.), each argument has a distinct head index:
  `<LL (S[dcl]\NP)/NP VBD VBD took ((S[dcl]\NP_6)/NP_7))>`
  If an argument is complex, the head index of each of its parts is given:
  `<L PP/(S[ng]\NP) IN IN as PP/(S[ng]_183\NP_181)_183>`

- In adjuncts of the form X|X, the head indices of the result X are identical to that of the argument X (recall that the head index of the entire category, which is different, is omitted): `<L N/N NNP NNP Pierre N_73/N_73>`
  This is also the case if X itself is a complex category, eg.:

  `<L (S\NP)\(S\NP) NN NN yesterday (S_9\NP_4)_9\(S_9\NP_4)_9>`
  Similarly, in adjuncts that take themselves arguments, eg. (NP\NP)/NP or (S\NP)\(S\NP))/NP, the head indices are adjusted accordingly:
  `<L (NP\NP)/NP IN IN of (NP_5\NP_5)/NP_6>`
  `<L ((S\NP)\(S\NP))/NP IN IN by ((S_6\NP_1)_6\(S_6\NP_1)_6)/NP_7>`

- The mediation of non-local dependencies is indicated by co-indexation. As explained in section 2.5.7, we distinguish between locally mediated (bounded) and long-range (undounded) dependencies. Locally mediated dependencies are indicated by `:B` (bounded):
  `<L (S[dcl]\NP)/(S[pt]\NP) VBZ VBZ has (S[dcl]\NP_3)/(S[pt]_4\NP_3:B)_4>` True long-range depdencies are indicated by `:U` (unbounded):
  `<L (NP\NP)/(S[dcl]/NP) IN IN that (NP_8\NP_8)/(S[dcl]_9/NP_8:U)_9>`

For non-leaf nodes, the description contains the following three fields: the category of the node `CCGcat`, the index of its head daughter `head` (0 = left or only daughter, 1 = right daughter), and the number of its daughters, `dtrs`: `<T CCGcat head dtrs>`[1] The following example describes a node with category S[dcl]\NP and two children (0 and 1), the left of which (child 0) is the head:

```
<T S[dcl]\NP 0 2>
```

Sentences that are not translated are not indicated in this file format.

D.3 The predicate-argument structure files

The predicate-argument structure files give for each sentence a list of the word-word dependencies in the predicate-argument structure, including locally mediated and long-range dependencies, which are indicated as such. For each file in the original Treebank, there is one corresponding predicate-argument structure file. Each sentence is enclosed by `<s>` (followed by the index of the last token in the sentence) and `<\s>`. Each dependency appears on one line.

```
<s> 12
1      0      N/N            1      Vinken Mr.
1      2      (S[dcl]\NP)/NP 1      Vinken is
3      2      (S[dcl]\NP)/NP 2      chairman is
3      4      (NP\NP)/NP     1      chairman of
6      4      (NP\NP)/NP     2      N.V. of
6      5      N/N            1      N.V. Elsevier
11     4      (NP\NP)/NP     2      group of
11     8      NP[nb]/N       1      group the
11     9      N/N            1      group Dutch
11     10     N/N            1      group publishing
<\s>
```

A dependency between the *i*th and *j*th word (`word_i` and `word_j`) where the *j*th word has the lexical (functor) category *cat_j*, and the *i*th word is head of the constituent which fills the *k*th argument slot of *cat_j* is described as follows:

```
i   j   cat_j    arg_k   word_i  word_j
```

[1] With the exception of argument clusters, the head corresponds generally to the lexical head.

Words in each sentence are numbered from 0 to *n*.

In the sentence *“Mr. Vinken is chairman of Elsevier”*, *Vinken* is the second word in the sentence and head of the constituent which fills the first (and only) argument slot of the N/N *Mr.*. At the same time, the N *Vinken* is head of the constituent which fills the first argument slot of the (S[dcl]\NP)/NP *is*, which is the third word in the sentence. Therefore:

```
1      0      N/N             1      Vinken Mr.
1      2      (S[dcl]\NP)/NP  1      Vinken is
```

Missing sentences are indicated by a pair of <s> and <\s> which does not enclose any dependencies:

```
<s> 0
<\s>
```

D.4 The lexicon files

The distribution contains two lexicon files that are extracted from sections 02-21 and 00. Each line contains one entry:

```
assume     (S[b]\NP)/NP          18   0.0013811094912913374  0.5294117647058824
assume     (S[b]\NP)/S[dcl]      3    0.011627906976744186   0.08823529411764706
assume     (S[b]\NP)/S[em]       6    0.02197802197802198    0.17647058823529413
assume     (S[dcl]\NP)/NP        2    1.2457178449081283E-4  0.058823529411764705
assume     (S[dcl]\NP)/S[dcl]    4    7.146685724495265E-4   0.11764705882352941
assume     (S[dcl]\NP)/S[em]     1    4.380201489268506E-4   0.029411764705882353
```

The word and its lexical category are followed by the frequency of the entry, the probability (relative frequency) of the word given the category and the probability of the category given the word.

D.5 CCGbank and TGrep2

CCGbank is searchable with TGrep2, an expression matcher for trees developed by Douglas Rohde. TGrep2 is available from `http://tedlab.mit.edu/~dr/TGrep2/`. The directory `data/TGREP2` contains a file `ccgbank.00-24.t2c` which can be searched by TGrep2. If TGrep2 version 1.15 or higher is used on ccgbank00-24.t2, it will run in CCG mode, which differs from its standard mode in the following ways:

- In TGrep2’s CCG mode, brackets (”[” and ”]”), parentheses (”(” and ”)”) and slashes (”\” and ”/”) can be part of a node label, but have to be preceded by a backslash in regular expression searches.
- In TGrep2’s CCG mode, curly brackets (”{” and ”}”) are used instead of parentheses (”(” and ”)”) to specify dominance relations and to bracket the output trees.
- In TGrep2’s CCG mode, the plus and minus signs (”+” and ”-”) are used instead of brackets (”[” and ”]”) to group disjunctive terms. For example, ”NP [> PP | > S]” in standard TGrep2 becomes ”NP + > PP | > S − ” in CCG mode.

Bibliography

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. In S. McCall, editor, *Polish Logic 1920-1939*, pages 207–231. Oxford University Press. Translated from *Studia Philosophica*, 1, 1-27.

Aone, C. and Wittenburg, K. (1990). Zero morphemes in Unification-based Combinatory Categorial Grammar. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 188–193, Pittsburgh, PA.

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, University of Edinburgh.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, **29**, 47–58.

Bar-Hillel, Y., Gaifman, C., and Shamir, E. (1960). On categorial and phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information*, pages 99–115. Addison-Wesley, Reading, MA. 1964.

Bies, A., Ferguson, M., Katz, K., MacIntyre, R., Tredinnick, V., Kim, G., Marcinkiewicz, M. A., and Schasberger, B. (1995). *Bracketing Guidelines for Treebank II Style Penn Treebank Project*. University of Pennsylvania.

Bos, J. (2005). Towards wide-coverage semantic interpretation. In *Proceedings of Sixth International Workshop on Computational Semantics IWCS-6*, pages 42–53, Tilburg, The Netherlands.

Bos, J., Clark, S., Steedman, M., Curran, J. R., and Hockenmaier, J. (2004). Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING'04)*, Geneva, Switzerland.

Bresnan, J., editor (1982). *The mental representation of grammatical relations*. MIT Press, Cambridge, MA.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2002). Automatic annotation of the Penn Treebank with LFG F-structure information. In *LREC 2002 Workshop on Linguistic Knowledge Acquisition and Representation - Bootstrapping Annotated Language Data*, pages 8–15, Las Palmas, Spain.

Carpenter, B. (1991). The generative power of Categorial Grammars and Head-driven Phrase Structure Grammars with lexical rules. *Computational Linguistics*, **17**(3), 301–314.

Carpenter, B. (1992). Categorial grammars, lexical rules, and the English predicative. In R. Levine, editor, *Formal Grammar: Theory and Implementation*, chapter 3. Oxford University Press.

Charniak, E. (1999). A Maximum-Entropy-inspired parser. Technical Report CS-99-12, Department of Computer Science, Brown University.

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

Chen, J. and Vijay-Shanker, K. (2000). Automated extraction of TAGs from the Penn Treebank. In *Proceedings of the 6th International Workshop on Parsing Technologies*, Trento, Italy.

Chiang, D. (2000). Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 456–463, Hong Kong.

Chiang, D. (2004). *Evaluation of Grammar Formalisms for Applications to Natural Language Processing and Biological Sequence Analysis*. Ph.D. thesis, University of Pennsylvania.

Clark, S. and Curran, J. R. (2003). Log-linear models for wide-coverage CCG parsing. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP'03)*, Sapporo, Japan.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain.

Clark, S., Hockenmaier, J., and Steedman, M. (2002). Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 327–334, Philadelphia, PA.

Clark, S., Steedman, M., and Curran, J. R. (2004). Object-extraction and question-parsing using CCG. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP'04)*, pages 111–118, Barcelona, Spain.

Collins, M. (1997). Three generative lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, Computer and Information Science, University of Pennsylvania.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic*, volume I. North-Holland, Amsterdam.

Eisner, J. (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, Santa Cruz, CA.

Gazdar, G., Klein, E., Pullum, G. K., and Sag, I. A. (1985). *Generalised Phrase Structure Grammar*. Blackwell, Oxford.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the International Workshop on Parsing Technologies*, Cambridge, MA.

Goodman, J. (1998). *Parsing Inside-Out*. Ph.D. thesis, Harvard University, Cambridge, MA.

Hepple, M. and Morrill, G. (1989). Parsing and derivational equivalence. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 10–18, Manchester, UK.

Hockenmaier, J. (2001). Statistical parsing for CCG with simple generative models. In *Proceedings of Student Research Workshop, 39th Annual Meeting of the Association for Computational Linguistics and 10th Meeting of the European Chapter*, pages 7–12, Toulouse, France.

Hockenmaier, J. (2003a). *Data and models for statistical parsing with Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, University of Edinburgh.

Hockenmaier, J. (2003b). Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Annual Meeting of the ACL*, Sapporo, Japan.

Hockenmaier, J. and Steedman, M. (2002a). Acquiring compact lexicalized grammars from a cleaner Treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pages 1974–1981, Las Palmas, Spain.

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

Hockenmaier, J., Bierner, G., and Baldridge, J. (2000). Providing Robustness for a CCG System. In *Proceedings of ESSLLI'2000 Workshop on Linguistic Theory and Grammar Implementation*, pages 97–112, Birmingham, UK.

Hockenmaier, J., Bierner, G., and Baldridge, J. (2004). Extending the Coverage of a CCG System. *Research on Language and Computation*, **2**(2), 165–208.

Joshi, A. and Schabes, Y. (1992). Tree adjoining grammars and lexicalized grammars. In M. Nivat and M. Podelski, editors, *Definability and Recognizability of Sets of Trees*. Elsevier, Princeton.

Joshi, A., Levy, L., and Takahashi, M. (1975). Tree-adjunct grammars. *Journal of Computer Systems Science*, **10**, 136–163.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic*. Kluwer, Dordrecht.

Kinyon, A. and Prolo, C. (2002). Identifying verb arguments and their syntactic function in the Penn Treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pages 1982–1987, Las Palmas, Spain.

Lambek, J. (1958). The mathematics of sentence structure. *American Mathematical Monthly*, **65**, 154–170.

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

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

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, **19**, 313–330.

Nunberg, G. (1990). *The linguistics of punctuation*. Number 18 in CSLI Lecture Notes. CSLI Publications.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, **31**(1), 71–105.

Pollard, C. and Sag, I. (1994). *Head Driven Phrase Structure Grammar*. CSLI/Chicago University Press, Chicago, IL.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, Philadelphia, PA.

Ross, J. R. (1967). *Constraints on Variables in Syntax.* Ph.D. thesis, MIT. Published as "Infinite Syntax!", Ablex, Norton, NJ. 1986.

Steedman, M. (1987). Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, **5**, 403–439.

Steedman, M. (1996). *Surface Structure and Interpretation*. MIT Press, Cambridge, MA. Linguistic Inquiry Monograph, 30.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA.

Uszkoreit, H. (1986). Categorial Unification Grammars. In *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, pages 187–194, Bonn, Germany.

Vijay-Shanker, K. and Weir, D. (1990). Polynomial time parsing of Combinatory Categorial Grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Pittsburgh, PA.

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

Villavicencio, A. (2002). *The Acquisition of a Unification-Based Generalised Categorial Grammar*. Ph.D. thesis, Computer Laboratory, University of Cambridge.

Watkinson, S. and Manandhar, S. (2001). Translating Treebank Annotation for Evaluation. In *Workshop on Evaluation for Language and Dialogue Systems, ACL/EACL*, pages 21–28, Toulouse, France.

Wittenburg, K. and Wall, R. (1991). Parsing with categorial grammar in predictive normal form. In M. Tomita, editor, *Current Issues in Parsing Technology*, pages 65–83. Kluwer, Dordrecht. Revised selected papers from International Workshop on Parsing Technology (IWPT) 1989, Carnegie Mellon University.

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

Wood, M. M. (1993). *Categorial Grammar*. Linguistic Theory Guides. Routledge, London.

Xia, F. (1999). Extracting Tree Adjoining Grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*.

Xia, F., Palmer, M., and Joshi, A. (2000). A uniform method of grammar extraction and its applications. In *Proceedings of the 2000 Conference on Empirical Methods in Natural Language Processing*, pages 53–62, Hong Kong.

Zeevat, H., Klein, E., and Calder, J. (1987). An introduction to unification categorial grammar. In N. e. a. Haddock, editor, *Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar, and Parsing*, pages 195–222. University of Edinburgh.