Chapter 23

PARSING

John Carroll

Abstract

This chapter introduces key concepts and techniques for natural language parsing; that is, finding the grammatical structure of sentences. The chapter introduces the fundamental algorithms for parsing with context-free (CF) phrase structure grammars, how these deal with ambiguous grammars, and how CF grammars and associated disambiguation models can be derived from syntactically annotated text. It reviews dependency analysis, and outlines the main approaches to dependency parsing based both on manually written grammars and on learning from dependency treebanks. It also describes techniques used for parsing with grammars that use feature structures to encode linguistic information.

Keywords

Parsing; syntax; phrase structure grammar; dependency grammar; disambiguation; treebanks

23.1 Introduction

Parsing is an important technology used in many language processing tasks. Parsing has always been an active area of research in computational linguistics, and many different approaches have been explored over the years. More recently, many aspects of parser development and evaluation methodology have become standardised, and shared tasks and common datasets for evaluation have helped to drive progress forward. However, there is still a diverse range of techniques being investigated. The diversity is along a number of dimensions, the main ones being:

- Representation of the set of possible sentences of the language and their parses—is this through a formal grammar, and if so how is it encoded and where is it derived from?
- Type of parser output—are parses **phrase structure trees**, **dependency structures**, **feature structures**, or some other kind of linguistic description?
- Parsing algorithm—is processing **deterministic** or **non-deterministic**, and what operations does the parser perform?
- Ambiguity resolution—at what stage in processing is **disambiguation** attempted, what type of disambiguation method is used, and how is search over possible parses managed?

The latter dimension is particularly important, since arguably the most significant problem faced in parsing is ambiguity. Church and Patil (1982) observe that there may be hundreds or thousands of parses for perfectly natural sentences. (Indeed, as the research field has developed and computers have become more powerful, some current approaches to parsing represent and process numbers of potential parses many orders of magnitude larger than this). Consider the sentence (23.1).

(23.1) *They saw some change in the market.*

Although to a human there is a single, obvious meaning for this sentence, there are a number of 'hidden' ambiguities which stem from multiple ways in which words can be used (lexical ambiguity), and in which words and phrases can be combined (syntactic ambiguity). When considered in isolation many of these possibilities may be quite plausible, but in the context of the rest of the sentence they contribute to very unlikely meanings. Sources of lexical and syntactic ambiguity in this sentence include the following.

- The word *saw* has two possible readings, as a past tense verb or a singular common noun.
- *Some* can be a determiner, preceding a noun and denoting an amount of it; it can act as a pronoun, meaning some people or things; or it can be an adverb meaning the same as *somewhat* (as in *The market always changes some*).
- *Change* can be a noun or a verb; for example it would be a verb when following *They saw some* in sentences such as *They saw some change their strategies*.
- The prepositional phrase *in the market* may relate either to *change* or to the action of

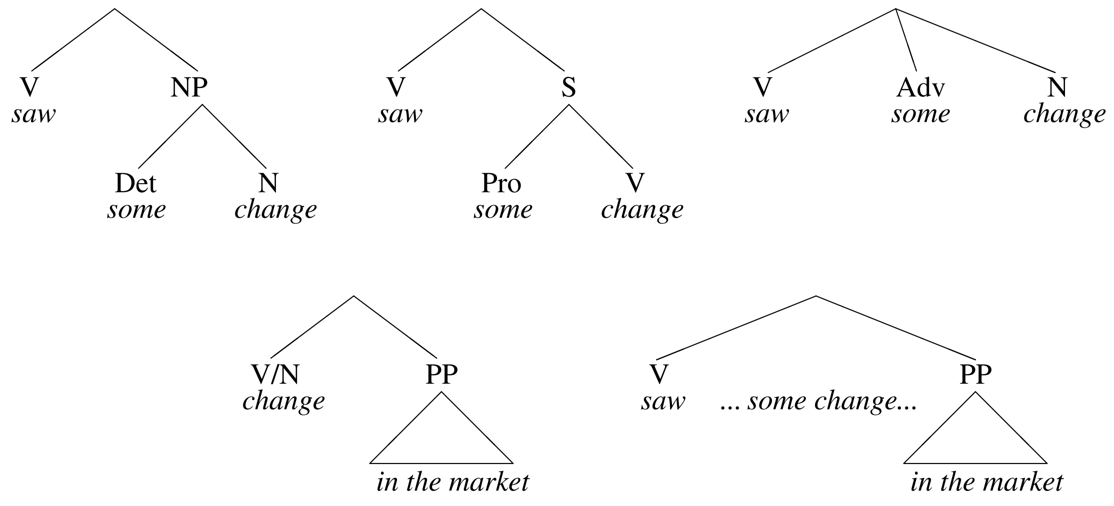


Figure 23.1: Ambiguity in natural language: fragments of multiple phrase structure trees for the sentence *They saw some change in the market.*

seeing, corresponding respectively to the paraphrases *They saw some change that was in the market* and *In the market, they saw some change*.

Of these ambiguities, those that can contribute to full parses are illustrated in Figure 23.1, producing a total of 6 distinct parses for the whole sentence.

Early parsing systems often used manually-developed heuristics to resolve ambiguities (for example 'prefer to attach prepositional phrases low'—i.e. to the most recent noun rather than a preceding verb) sometimes encoding these in metrics used to score and rank complete parses (Heidorn 1982). Another approach involved making disambiguation decisions based on inference over semantic relationships between words, with data either coming from a hand-coded 'semantic lexicon' or from automatic processing of dictionary and encyclopaedia entries (Binot and Jensen 1987). Such approaches have several shortcomings. In particular, heuristics can only cover a very small proportion of lexical and syntactic ambiguities, inference breaks down if any piece of relevant information is missing, and building semantic lexicons is very labour-intensive so is only practical for a system targeted at a limited domain.

However, from the late 1980s, drawing on work in the field of corpus linguistics and inspired by significant advances in speech recognition resulting from statistical and machine learning techniques, parsing research turned to approaches based on information learned from large amounts of text that had been manually annotated with syntactic structure. This type of annotated text is called a **treebank**. The first widely used treebank was the Penn Treebank (Marcus et al. 1993); the major part of this consists of around one million words of text from the Wall Street Journal, each sentence associated with a phrase structure tree representing its syntactic structure. Treebanks for languages other than English have followed, and there are now large treebanks for most of the world's major languages.

The first work using treebanks demonstrated three main ways in which the information in a treebank may be used by a parser; each of these ways is still under active investigation.

- A hand-crafted grammar already exists, and the information in the treebank is used for disambiguating the analyses produced by the grammar (Briscoe and Carroll 1993; Toutanova et al. 2002).

- A grammar is extracted from the syntactic structures in the treebank, together with associated statistical information which is used to disambiguate the analyses produced by the grammar (Charniak 1996; Xia 1999).

- There is no explicit grammar, and the search for the best parse is constrained only by information about numbers of occurrences of various types of syntactic configurations in the treebank (Sampson 1986; Magerman 1995).

As well as being used for developing parsers, treebanks are also used for evaluating their accuracy, by providing a **gold standard** set of parses for some set of test inputs. However, the most obvious evaluation metric of exact match of parses against the gold standard is usually not appropriate because (i) apparent differences between parses might not ultimately correspond to any real differences in meaning, and also (ii) the parser might have been designed to analyse certain constructions differently than the standard. Instead, parser accuracy is usually measured as the percentage of phrases or grammatical relationships between words correctly identified (Carroll et al. 1998). However, differences between the syntactic representations output by different types of parser mean that comparative evaluations have to be interpreted carefully (Clark and Curran 2007).

Although a number of treebanks are now available, they are very expensive in terms of human effort to produce and can therefore cover only a limited range of genres, topic domains and languages. Even within a single language there are significant differences in language use between genres (e.g. newspaper text and mobile phone text messages) and domains (e.g. finance news and biomedical abstracts), which causes parsers developed for one genre or domain to perform poorly in another. These issues have motivated a number of strands of research including: unsupervised learning of syntax from unannotated text (Ponvert et al. 2011; Scicluna and Higuera 2014); projecting syntactic annotations from a treebank in one language to another (Tiedemann 2014); and adapting parsers trained on text in a (source) domain to deal effectively with text in another (target) domain (Yu et al. 2015).

23.2 Context-free grammar parsing

Context-free phrase structure grammars form the basis of many natural language parsing systems. Chapter 4 introduces these grammars, explains how they group sequences of words into phrases, and how the phrase structure can be represented as a tree. The task of a phrase structure parser is to find the tree (or trees) corresponding to a given input string (sentence). Context-free (CF) parsing algorithms are also fundamental in that parsing techniques for other grammar frameworks are often based on them. The following sections describe the two CF parsing algorithms of most relevance to natural language processing, shift-reduce parsing and the CYK tabular parsing algorithm.

23.2.1 Shift-reduce parsing

The shift-reduce algorithm is conceptually one of the simplest parsing techniques. The algorithm comprises two main steps, *shift* and *reduce*, which are applied to a buffer and a stack of partial analyses. Initially the buffer holds the complete input sentence and the stack is empty. Words are shifted from the buffer onto the stack; when the top items of the stack match the right side of a rule in the CF grammar (Chapter 4), the reduce step replaces these with the category on the left side of the rule. This process is depicted in Figure 23.2 (assuming analogous application of the reduce operation for rules with other numbers of daughters). In the case of grammars that are unambiguous (no string has more than one analysis), as long as the algorithm always carries out a reduce when there is an applicable rule it will successfully analyse any input string in the language defined by the grammar.

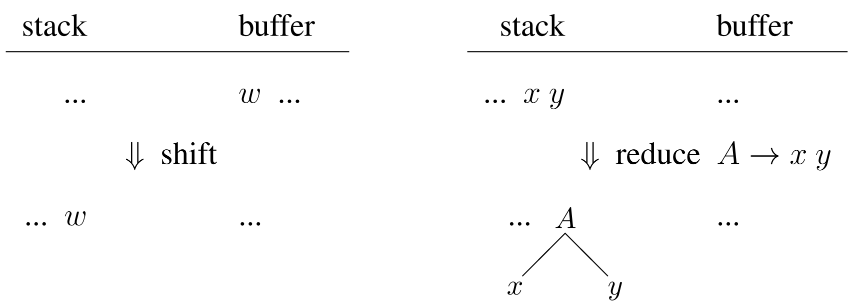


Figure 23.2: Steps in the shift-reduce algorithm; note that the top-most item of the stack is on the right hand end

The shift-reduce algorithm—and variants of it—are applied widely in compilers for programming languages for parsing the source code of computer programs, since grammars for these are designed to be unambiguous or to contain only ambiguities that can be disambiguated using limited contextual information. However, natural language is highly ambiguous, and attempting to use the algorithm as described above with natural language grammars would usually result in the algorithm choosing to shift or reduce wrongly at some point and failing to find a complete analysis when one existed. In addition, the algorithm would have to **backtrack** to find further analyses in case the one found first was not the most plausible interpretation of the input. The algorithm therefore has to be adapted in order to make it applicable for natural language parsing.

One way of dealing with ambiguity in shift-reduce parsing is to look ahead at unprocessed words in the buffer to decide what the next step should be; Marcus (1980) used this approach in a study which attempted to model human sentence understanding, investigating the question of whether it could be deterministic. In the generalized LR parsing technique (Tomita 1985), the stack (which would normally hold a linear sequence of words and phrasal constituents) becomes a graph which is able to represent all possible ways of analysing the words processed so far, allowing all possible parses to be computed efficiently. Finally, more recent work on data-driven

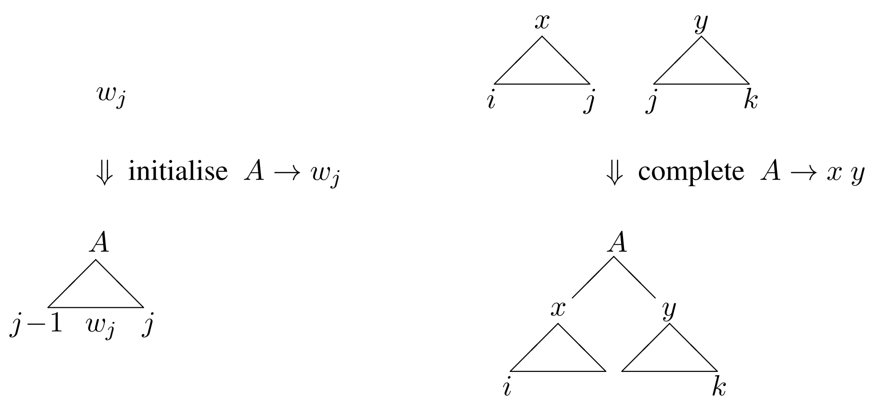

w_j
$\Downarrow$ initialise $A \to w_j$
A
$j-1$ w_j j
x
y
i j j k
$\Downarrow$ complete $A \to x\ y$
A
x y
i k

Figure 23.3: Steps in the CYK algorithm

dependency parsing (section 23.3 below) uses the shift-reduce algorithm together with a machine learning classifier to deterministically select the next step the parser should perform.

23.2.2 Tabular parsing

For ambiguous grammars, tabular parsing algorithms overcome some of the drawbacks of the basic shift-reduce parsing algorithm. The most basic tabular parsing algorithm is the Cocke-Younger-Kasami (CYK) algorithm (Cocke and Schwartz 1970; Younger 1967; Kasami 1965). Strictly, it requires that the grammar be expressed in Chomsky Normal Form (Chomsky 1959): the right side of each rule must either be a single word or exactly two non-terminal categories (left hand categories of other rules). Figure 23.3 illustrates the operation of the algorithm.

First, in the the *initialise* step, each word w_j is recorded as a constituent of length 1 covering input positions $j-1$ to j. Then, successively processing larger segments of the input, *complete* steps form a new higher-level constituent for every pair of contiguous constituents of categories x and y and rule $A \to x\ y$. This process continues until no further complete steps can be performed. Taking the sentence (23.1) as an example, one of the complete steps might involve a rule PP $\rightarrow$ Prep NP being applied to the category Prep corresponding to the word *in* between input positions 4 and 5, and an NP (*the market*) between 5 and 7, producing a PP (prepositional phrase) between positions 4 and 7. With minor changes the CYK algorithm can be adapted to parse with any *2-normal-form* grammar, in which no rule has more than two daughters (Lange and Leiß 2009). The algorithm can also be extended to work with any arbitrary CF grammar without restriction on the rule right sides; this variant is known as bottom-up passive chart parsing (Kay 1986).

The ambiguity inherent in natural language means that a given segment of the input string may end up being analysed as a constituent of a given category in several different ways. With any parsing algorithm, each of these different ways must be recorded, of course, but subsequent parsing steps must treat the set of analyses as a single entity, otherwise the computation becomes theoretically intractable. Tomita (1985) coined the terms:

- **local ambiguity packing** for the way in which analyses of the same type covering the same segment of the input are conceptually 'packed' into a single entity; and

- **subtree sharing** where if a particular sub-analysis forms part of two or more higher level analyses then there is only a single representation of the sub-analysis, and this is shared between them.

The final representation produced by the parser is called a **parse forest** (see e.g. Billot and Lang 1989), and is produced quite naturally if each step records back-pointers to the phrases and words contributing to it. Figure 23.4 shows a fragment of the parse forest that might be constructed for (23.1), which would unpack into two distinct parse trees (one in which *in the market*

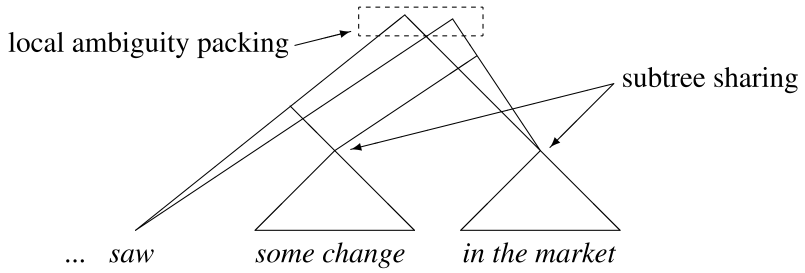


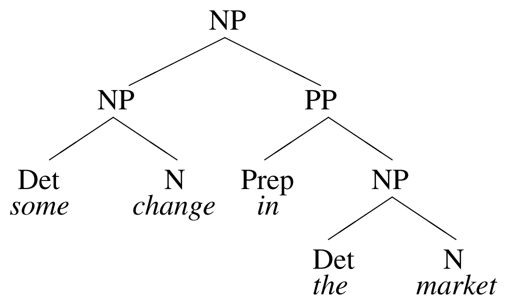
Figure 23.4: A fragment of a parse forest

modifies the noun phrase *some change*, and another in which it modifies the verb phrase *saw some change*).

Many further tabular parsing algorithms exist. Some, like CYK, only record complete constituents, whereas others (for example the active chart parsing algorithm) also store partial constituents which record that a particular category has been found and that further ones must also be found in specified locations relative to it. Some algorithms build all sub-analyses possible for the input, whereas others—for example Earley's (1970) algorithm—use top-down information derived from the grammar to avoid producing some partial analyses that could not contribute to a complete parse. The common factor between these algorithms is that they cope efficiently with ambiguity by not deriving the same constituent by the same set of steps more than once; they do this by storing derived constituents in a **well-formed substring table** (Sheil 1976), or **chart**, and retrieving entries from the table as needed, rather than recomputing them.

23.2.3 Data-driven phrase structure parsing

As mentioned in Section 23.1, grammars and information for making disambiguation decisions can be extracted from treebanks. The Penn Treebank contains phrase structure trees with atomic



$\Rightarrow$

$$\begin{array}{ll}
\text{PP} \rightarrow \text{Prep NP} & 1 \\
\text{NP} \rightarrow \text{NP PP} & \frac{1}{3} \\
\text{NP} \rightarrow \text{Det N} & \frac{2}{3}
\end{array}$$

Figure 23.5: Deriving a probabilistic CF grammar from a phrase-structure tree

categories (see Chapter 4), which means that a CF treebank grammar can be created from it by constructing a CF rule for each distinct local (one-level) tree in the treebank. The probability that a particular rule should be applied can be estimated directly from the treebank by accumulating a frequency count for each rule and then normalising frequency counts so that the probabilities of each set of rules with the same left hand category sum to one (Figure 23.5). This results in a **Probabilistic CF Grammar** (PCFG). When parsing with a PCFG, the probability of a parse tree is the product of the probabilities of the rules in the tree. A version of the CYK algorithm that computes the probability of constituents on each *complete* step can be used to find the parse with the highest probability efficiently.

Although Charniak (1996) shows that a PCFG derived from the Penn Treebank can give moderately accurate results, with around 80% correct identification of phrase boundaries, PCFG has a number of shortcomings. In particular, it cannot account for the substantial influence on preferred readings exerted by syntactic context and word choice. For example, in English, right-branching syntactic structures are more prevalent than left-branching structures, but PCFG cannot capture this tendency probabilistically. Nor can it model the fact that in the most plausible reading of (23.2a) below, the prepositional phrase *in the market* modifies *some change*, whereas in (23.2b) a similar prepositional phrase (with one word different) modifies the main verb, *saw*.

(23.2) a. *They saw some change in the market.*

b. *They saw some change in the afternoon.*

A further anomaly arises from the fact that most treebanks give rise to a very large number of distinct rules; since the probability associated with each rule is estimated independently of all others, minor variants of the same rule may be assigned very different probabilities due to data sparseness.

These problems with PCFG have motivated a lot of research in recent years, leading to statistical models which are still derived from treebanks but which better capture the interactions between syntactic constructions and word choice. One successful approach (Collins 1997) models the derivation of a phrase structure tree as a sequence of steps in which child nodes are added to a partial analysis of the input sentence, their probabilities being conditioned on properties of parent and sibling nodes that have already been added, including the input words dominated by these nodes. Such **history-based models** require a very large number of probabilities, and thus rely heavily on obtaining good estimates for the probabilities of events that have not been observed in the training data. Applying the model is often computationally expensive, so a pragmatic approach is to use a relatively simple approximation of the model in a first pass in order to prune the search space sufficiently that the full model can be used.

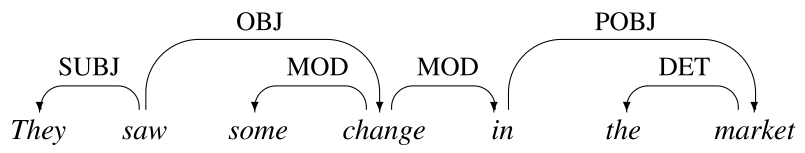
Further investigation of the types of dependence found to be useful in history-based models has led to the insight that much of this information can be encoded in a standard PCFG derived from a transformed version of the original treebank. One particularly effective type of transformation involves adding information to tree node labels about the context in which they appear, for example splitting up node categories into a finer-grained set based on the category of the parent node in the tree improves the parsing accuracy of the derived PCFG substantially (Johnson 1998). Another useful transformation breaks up rules with more than two right hand categories into equivalent unary and binary rules; this emulates the effect of adding child nodes individually in the history-based model, and again results in large improvements in accuracy (Klein and Manning 2003). Petrov and Klein (2007) show how these beneficial transformations can be learned from a treebank and how the parser search space can be managed, leading to one of the best performing current models for phrase structure parsing.

For reasons of computational tractability, the types of parsing models outlined above have to make independence assumptions which mean that only close-range syntactic and lexical interactions can be modelled probabilistically. However, if such a parser can return a ranked set of the top few parse candidates, a second model incorporating global features of syntactic representations can be used to re-rank them. This can lead to a significant increase in accuracy (Charniak and Johnson 2005).

23.3 Dependency parsing

In **dependency grammar** (Mel'čuk 1987), a syntactic analysis takes the form of a set of directed links between words, each link being labelled with the grammatical function (for example, *subject* or *object*) that relates a dependent word to its governor (Chapter 4). A dependency grammar analysis of the sentence (23.1) might be (23.3).

(23.3)



A dependency analysis does not group words into phrases and phrases hierarchically into trees, but instead encodes relationships between pairs of words. Dependency grammar has been argued to be more appropriate than phrase structure grammar for languages with relatively free word order; in many such languages the order in which the arguments of a predicate are expressed may vary, not being determined by their grammatical roles but rather by pragmatic factors such as focus and discourse prominence. (In English, word order is constrained so that the subject almost always precedes its associated main verb and the direct object follows it; this is not the case in German, for example.) Dependencies that are labelled with their type (as in the example above) are sometimes termed 'grammatical relations', and encode important aspects of predicate-argument structure without needing to commit to a particular theory of how phrases are structured hierarchically.

23.3.1 Grammar-based dependency parsing

There are a number of successful approaches to parsing with dependency grammar. In Link Grammar (Grinberg et al. 1995), a grammarian builds a lexicon in which each word is associated with a set of possible links, each one marked with an indication of whether the other word in the dependency relation should appear to the left or right in the sentence; for example in (23.4), the word *change* optionally has a modifier to its left (encoded as {*MOD*–}) and is in either an

OBJ relation with a word to its left or a SUBJ relation with a word to its right (encoded as *OBJ– or SUBJ+*).

(23.4) *saw*: SUBJ– & OBJ+

some: MOD+

change: {MOD–} & (OBJ– or SUBJ+)

The parsing process consists of pairing up words via their link specifications (e.g. pairing up a *SUBJ+* with a *SUBJ–* to its right), subject to the constraint that no link should cross another. Governors are not distinguished from dependents, as in standard dependency grammar, so a language processing application using link grammar would have to do extra work to infer this information.

Functional Dependency Grammar parsing (Tapanainen and Järvinen 1997) works by first labelling each word with all its possible function types (according to a lexicon), and then applying a collection of hand-written rules that introduce links between specific function types in a given context, and perhaps also remove other function type readings. One of the rules, for instance, might add a subject dependency between a noun and an immediately following finite verb, and remove any other possible functions for that noun. Finally, a further set of rules are applied to remove unlikely linkages, although some ambiguity may still be left at the end in cases where the grammar has insufficient information to be able to resolve the ambiguity.

An alternative approach, taken by Constraint Dependency Grammar (Maruyama 1990), is to view parsing as a constraint satisfaction process. Initially, each word is hypothesised to depend on every other word, and then a set of constraints are applied which specify restrictions on the possible dependencies between a word and the possible governors of that word. For example, one constraint might be that a preposition requires a preceding verb or noun governor. In Weighted Constraint Dependency Grammar (Foth et al. 2005), constraints have weights associated with them to add flexibility to deal with ungrammatical inputs or grammatical constructions that are more conveniently specified as preferences rather than ‘hard’ rules.

23.3.2 Data-driven dependency parsing

As well as treebanks of phrase structure trees, there are now several treebanks of dependency analyses available. Much recent research into parsing has focussed on data-driven approaches to dependency parsing, using the information in these treebanks to direct the parsing and disambiguation process.

One approach, transition-based dependency parsing (pioneered by Yamada and Matsumoto 2003; Nivre and Scholz 2004), is based on the shift-reduce algorithm described in Section 23.2.1, but adapted to build dependency analyses rather than phrase structure. This can consist of replacing the reduce step with operations *left-arc* and *right-arc*; these each take the top two words on the stack, create a dependency link of a specified type between them in a leftwards or rightwards direction respectively, and leave just the governor on the stack. Dependency links are accumulated as the parse proceeds, and all are returned at the end. Figure 23.6 illustrates some of the steps that might be performed when parsing the sentence (23.1). Parsing is deterministic, each step being selected by a machine learning classifier trained on a dependency treebank. The training procedure consists of determining the steps the parser would take to produce the analyses in

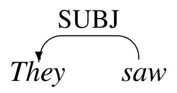
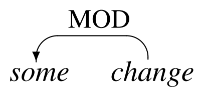
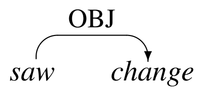
stack	buffer	dependencies
	They saw some change in the market	
	⇓ shift	
They	*saw some change in the market*	
	⇓ shift	
They saw	*some change in the market*	
	⇓ left-arc SUBJ	
saw	*some change in the market*	
	⇓ shift	
saw some	*change in the market*	
	⇓ shift	
saw some change	*in the market*	
	⇓ left-arc MOD	
saw change	*in the market*	
	⇓ shift	
saw change in	*the market*	
	⋮	
saw change		
	⇓ right-arc OBJ	
saw		

Figure 23.6: The first few and the final processing steps in a transition-based dependency parse of the sentence *They saw some change in the market*

the treebank, extracting a set of features characterising the state of the parsing process at each step, and providing these as training data to the classifier (Chapter 13). A typical set of features would be the words nearest the top of the stack, their left and right dependents (if any), and the first word in the buffer. For example, in Figure 23.6, when the stack contained *saw change* and the buffer *in the market*, the values of these features would be as in (23.5) and the correct action would be to shift the next word (*in*) from the buffer rather than perform a right- or left-arc step.

(23.5) top of stack: *change*

2nd in stack: *saw*

first in buffer: *in*

left dependent of top of stack: *some*

right dependent of top of stack: –

left dependent of 2nd in stack: *They*

right dependent of 2nd in stack: –

With sufficient suitable training examples, the parser would learn that in similar circumstances it should shift a preposition onto the stack (so it could be linked to the object noun a little later), rather than linking the object to the main verb (which would force the preposition eventually also to be linked to the verb).

Another approach to data-driven dependency parsing, graph-based dependency parsing (McDonald et al. 2005), takes a more direct route to finding the best dependency analysis. One version of the approach starts by constructing a strongly connected weighted directed graph with the words in the input sentence as the nodes, each arc in the graph holding a score representing the likelihood of a dependency link between the pair of nodes it connects. These scores are derived from a dependency treebank, and can depend on features of the arc (e.g. the pair of words involved), as well as features of the rest of the input sentence. Next the 'maximum spanning tree' (the maximum scoring selection of arcs forming a single tree that spans all the vertices) is computed, from which the best-scoring analysis can be read off. Figure 23.7 shows an example of how this works. The way in which this version of graph-based dependency parsing is parameterised means that the selection of each dependency link is considered independently

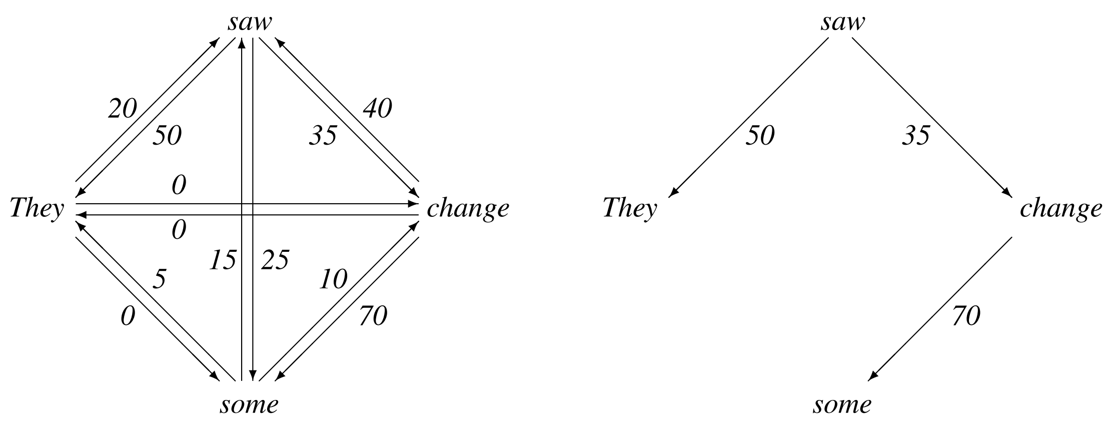
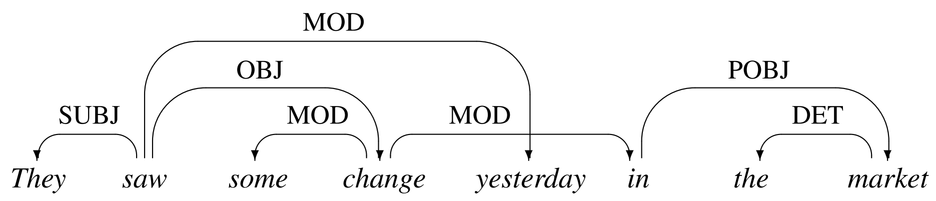


Figure 23.7: A graph representing the score of each possible dependency link for the sentence *They saw some change*, and the maximum spanning tree of the graph

of all the others. This is clearly an over-simplification, but even minor extensions in order to model interactions between adjacent arcs make the computation much more costly. Another property of the technique is that a maximum spanning tree can correspond to a 'non-projective' dependency structure; in such structures there are crossing dependency links. An example of a non-projective dependency structure is given in (23.6), in which two modifiers fail to follow a nested arrangement.

(23.6)



Non-projective dependencies are relatively rare in English text, but are somewhat more common in languages with freer word order or syntactic phenomena such as clause-final verb clusters (Zeman et al. 2012); however since global link structure cannot be modelled, this parsing algorithm may create non-projective dependencies even if there are none in the training data. In contrast, in transition-based dependency parsing, to produce non-projective dependencies the procedure outlined at the beginning of this section has to be modified; this is either done by adding further kinds of parsing operations, or by transforming the parser output to retrieve any non-projective dependencies (Björkelund and Nivre 2015).

As well as approaches to dependency parsing as outlined above whose primary computations are over dependency links, there are also parsing systems that take advantage of the fact that there is a straightforward correspondence between projective dependency analyses and a phrase structure trees, internally computing phrase structure trees but then converting these to dependency representations for output. Such systems include the RASP system (Briscoe et al. 2006) and the Stanford Parser (de Marneffe et al. 2006).

23.4 Feature-structure grammar parsing

Although data-driven approaches to parsing in which syntactic information is derived from treebanks have been successful, these approaches have some shortcomings. Firstly, most phrase structure and dependency treebanks encode only surface grammatical information explicitly, omitting or representing implicitly information such as 'deep role' in passive, raising and control constructions—for example that *they* is the agent of the verb *go* in the sentence (23.7).

(23.7) *They expect to go.*

This means that parsers trained on such treebanks may be unable to return some kinds of predicate-argument relations reliably. Secondly, although parsers that extract their knowledge of syntactic structure from a treebank may model the grammar of the text in the treebank well, they often do not work well on text with different characteristics; for example the Wall Street Journal text in the Penn Treebank contains very few questions, so a parser using grammatical information derived purely from the treebank would not be able to parse questions accurately.

These problems have been addressed by enriching treebanks, transforming them into a more expressive representation that makes all aspects of predicate-argument structure explicit. Examples of this approach are: CCGbank, a translation of the Penn Treebank into Combinatory Categorial Grammar derivations (Hockenmaier and Steedman 2007); Cahill et al.'s (2008) procedure for automatically annotating a phrase structure treebank with the functional structure of Lexical Functional Grammar (LFG; Bresnan 2000); and Candito et al.'s (2014) semi-automatic transformation of the Sequoia French dependency treebank to add a 'deep' level of representation.

Another approach is to *manually* develop a grammar in a powerful and expressive framework such as LFG or Head-Driven Phrase Structure Grammar (HPSG; Pollard and Sag 1994). Such grammars produce precise, detailed semantic representations—but at the cost of requiring an expert grammarian to develop the grammar, and the need to integrate a component for disambiguating the analyses produced by the grammar. Developing such grammars is a labour-intensive process, but is aided by **grammar development environments** which provide sophisticated tools for inspecting, testing and debugging the grammar (see e.g. Copestake 2002).

CF parsing algorithms form the basis for parsing with these more expressive formalisms, but augmented with operations over feature structures, which are used to encode detailed linguistic information. During parsing, feature structures are combined with the **unification** operation. For example, the result of unifying the feature structure (23.8a) with (23.8b) is (23.8c).

(23.8) a.
$$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{CAT} & v \\ \text{AGR} & \begin{bmatrix} \text{PER} & 3 \\ \text{PLU} & - \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{ORTH} & saw \\ \text{SYN} & \begin{bmatrix} \text{CAT} & v \\ \text{VFORM} & past \end{bmatrix} \end{bmatrix}$$

c.
$$\begin{bmatrix} \text{ORTH} & saw \\ \text{SYN} & \begin{bmatrix} \text{CAT} & v \\ \text{AGR} & \begin{bmatrix} \text{PER} & 3 \\ \text{PLU} & - \end{bmatrix} \\ \text{VFORM} & past \end{bmatrix} \end{bmatrix}$$

Unification would fail if, in this example, the value of the CAT feature in one of the input feature structures was not *v*. In contrast to the atomic, unstructured symbols of CF grammar, feature structures allow a grammar writer to conveniently cross-classify categories and also to leave features underspecified when appropriate. Unification is used to communicate information introduced by lexical entries and grammar rules in order to validate proposed local and non-local syntactic dependencies, and also in some grammar theories it is the mechanism through which semantic representations are constructed.

A key property of unification is that the order in which a set of unifications is performed does not affect the final result; therefore any parsing strategy appropriate for CF grammars (such as one of those outlined in Section 23.2.2) is equally applicable to unification-based grammars.

If each category is represented purely by a feature structure then the test for compatibility of categories is unification (rather than category symbol equality), and for local ambiguity packing one category must stand in a **subsumption** relationship to the other (Oepen and Carroll 2000). Alternatively, the grammar may consist of a **context-free backbone** augmented with feature structures, in which case the parsing process would be driven by the backbone part of the grammar and the appropriate unifications either carried out on each *complete* step, or after the full context-free parse forest had been constructed (Maxwell and Kaplan 1993; Torisawa and Tsujii 1996).

Given that treebanks invariably contain a wide variety of local tree configurations with nodes whose syntactic category labels are only atomic, grammars extracted from treebanks tend to both **overgenerate** and **overaccept** (that is, they return complete parses for ungrammatical input, and return too many parses for grammatical input, respectively). Any input usually receives some sort of parse, so coverage is not a problem. Gaps in coverage are often a problem for manually-developed grammars, though, since they are typically more precise in terms of the fragment of the language they cover. Coverage—and also overgeneration and overacceptance—can be quantified with respect to a **test suite** (Oepen and Flickinger 1998). This is becoming increasingly important as a quality assurance measure for parsers that are deployed in language processing applications (Chapters 32–48).

Further reading and relevant resources

Jurafsky and Martin's (2008) textbook contains basic descriptions of various parsing techniques. Manning and Schütze (1999) give a general introduction to data-driven approaches to parsing. The second part of Roark and Sproat's (2007) book contains a more in-depth presentation of techniques for phrase structure parsing.

Sikkel (1997) gives detailed specifications of a large number of phrase structure parsing algorithms, including proofs of their correctness and how they interrelate. Gómez-Rodríguez et al. (2011) similarly present specifications of a number of dependency parsing algorithms. Kübler et al. (2009) survey approaches to dependency parsing, focussing on data-driven transition-based and graph-based techniques. A special issue of the journal *Natural Language Engineering*, 6(1), 2000 contains a number of articles on techniques for parsing with feature-structure grammars.

The Association for Computational Linguistics (ACL) special interest group on parsing, SIGPARSE, organises biennial conferences, under the title *International Conference on Parsing Technologies*. The first such event (then titled 'Workshop' and with the acronym IWPT) was held in 1989. Links to online papers and abstracts, and references to books containing published versions of some of the papers can be found at the SIGPARSE website

`http://www.cs.cmu.edu/~sigparse/`

Other, more focussed workshops have been organised on topics such as parser evaluation, parsing of morphologically rich languages, efficiency of parsing systems, incremental parsing, parsing with categorial grammars, parsing and semantic role labelling, tabulation in parsing, and syntactic analysis of non-canonical language.

Research into statistical techniques for parsing is frequently published in the conference series *Empirical Methods in Natural Language Processing* sponsored by ACL/SIGDAT, and workshops on *Computational Natural Language Learning* sponsored by ACL/SIGNLL; see

`http://www.sigdat.org/`

`http://ifarm.nl/signll/`

Parsing is also always well-represented at the major international computational linguistics conferences.

There are various sources for parser training and evaluation data. The Linguistic Data Consortium (LDC) distributes the Penn Treebank, large treebanks for Arabic, Chinese, and Czech, and data from the CoNLL-X shared task on multilingual dependency parsing, covering 10 further languages (Bulgarian, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish and Turkish). Also available is a translation of the Penn Treebank into a corpus of Combinatory Categorial Grammar derivations (CCGbank).

`http://www.ldc.upenn.edu/`

Other large treebanks under active development include the French Treebank and the LinGO Redwoods Treebank.

`http://www.llf.cnrs.fr/fr/Gens/Abeille/French-Treebank-fr.php`

`http://moin.delph-in.net/RedwoodsTop`

The Universal Dependencies (UD) initiative is developing a single coherent framework for annotation of similar syntactic constructions across languages; UD treebanks are available for around 50 languages.

`https://universaldependencies.github.io/docs/`

The easiest way to gain practical experience with natural language parsing is to obtain one of the number of publicly-available grammar development/parsing systems. With a little effort some of them can be retrained on new data or new grammars loaded, and others can be customised to some extent by adding new lexical entries for example.

- Berkeley Parser: `https://github.com/slavpetrov/berkeleyparser`
- Charniak-Johnson Parser: `https://github.com/BLLIP/bllip-parser`
- Link Grammar Parser: `http://www.abisource.com/projects/link-grammar/`
- LKB: `http://moin.delph-in.net/LkbTop`
- MaltParser: `http://maltparser.org/`
- MSTParser: `http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html`
- RASP: `http://users.sussex.ac.uk/~johnca/rasp/`
- Stanford Parser: `http://nlp.stanford.edu/software/lex-parser.shtml`

References

Billot, Sylvie, and Bernard Lang (1989). ‘The structure of shared forests in ambiguous parsing’. In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*, 143–151. Vancouver, Canada.

Binot, Jean-Louis, and Karen Jensen (1987). ‘A semantic expert using an online standard dictionary’. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 709–714. Milan, Italy.

Björkelund, Anders, and Joakim Nivre (2015). ‘Non-deterministic oracles for unrestricted non-projective transition-based dependency parsing’. In *Proceedings of the 14th International Conference on Parsing Technologies*, 76–86. Bilbao, Spain.

Bresnan, Joan (2000). *Lexical-Functional Syntax*. Oxford: Blackwell.

Briscoe, Ted, and John Carroll (1993). ‘Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars’. *Computational Linguistics*, 19(1), 25–59.

Briscoe, Ted, John Carroll, and Rebecca Watson (2006). ‘The second release of the RASP system’. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, 77–80. Sydney, Australia.

Cahill, Aoife, Michael Burke, Ruth O’Donovan, Stefan Riezler, Josef van Genabith, and Andy Way (2008). ‘Wide-coverage deep statistical parsing using automatic dependency structure annotation’. *Computational Linguistics*, 34(1), 81–124.

Candito, Marie, Guy Perrier, Bruno Guillaume, Corentin Ribeyre, Karën Fort, Djamé Seddah, and Éric de la Clergerie (2014). ‘Deep syntax annotation of the Sequoia French treebank’.

In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC)*, 2298–2305. Reykjavik, Iceland.

Carroll, John, Ted Briscoe, and Antonio Sanfilippo (1998). ‘Parser evaluation: a survey and a new proposal’. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, 447–454. Granada, Spain.

Charniak, Eugene (1996). ‘Tree-bank grammars’. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI’96)*, 1031–1036. Portland, OR, USA.

Charniak, Eugene, and Mark Johnson (2005). ‘Coarse-to-fine n-best parsing and MaxEnt discriminative reranking’. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, 173–180. Ann Arbor, MI, USA.

Chomsky, Noam (1959). ‘On certain formal properties of grammars’. *Information and Control*, 2(2), 137–167.

Church, Kenneth, and Ramesh Patil (1982). ‘Coping with syntactic ambiguity or how to put the block in the box on the table’. *American Journal of Computational Linguistics*, 8(3–4), 139–149.

Clark, Stephen, and James Curran (2007). ‘Formalism-independent parser evaluation with CCG and DepBank’. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 248–255. Prague, Czech Republic.

Cocke, John, and Jacob Schwartz (1970). *Programming Languages and their Compilers: Preliminary Notes*. Technical report, Courant Institute of Mathematical Sciences, New York University.

Collins, Michael (1997). ‘Three generative, lexicalised models for statistical parsing’. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 16–23. Madrid, Spain.

Copestake, Ann (2002). *Implementing Typed Feature Structure Grammars*. Stanford, CA: CSLI Press.

de Marneffe, Marie-Catherine, Bill MacCartney, and Christopher Manning (2006). ‘Generating typed dependency parses from phrase structure parses’. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, 449–454. Genoa, Italy.

Earley, Jay (1970). ‘An efficient context-free parsing algorithm’. *Communications of the Association for Computing Machinery*, 13(2), 94–102.

Foth, Kilian, Wolfgang Menzel, and Ingo Schröder (2005). ‘Robust parsing with weighted constraints’. *Natural Language Engineering*, 11(1), 1–25.

Gómez-Rodríguez, Carlos, John Carroll, and David Weir (2011). ‘Dependency parsing schemata and mildly non-projective dependency parsing’. *Computational Linguistics*, 37(3), 541–586.

Grinberg, Dennis, John Lafferty, and Daniel Sleator (1995). ‘A robust parsing algorithm for link grammars’. In *Proceedings of the 4th International Workshop on Parsing Technologies*, 111–125. Prague, Czech Republic.

Heidorn, George (1982). ‘Experience with an easily computed metric for ranking alternative parses’. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, 82–84. Toronto, Canada.

Hockenmaier, Julia, and Mark Steedman (2007). ‘CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank'. *Computational Linguistics*, 33(3), 355–396.

Johnson, Mark (1998). 'PCFG models of linguistic tree representations'. *Computational Linguistics*, 24(4), 613–632.

Jurafsky, Daniel, and James Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 2nd edition*. Englewood Cliffs, NJ: Prentice-Hall.

Kasami, Tadao (1965). *An Efficient Recognition and Syntax Analysis Algorithm for Context-free Languages*. Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, USA.

Kay, Martin (1986). 'Algorithm schemata and data structures in syntactic parsing'. In Barbara Grosz, Karen Spärck-Jones, and Bonnie Webber (eds.), *Readings in Natural Language Processing*. San Mateo, CA: Morgan Kaufmann, 35–70.

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

Kübler, Sandra, Ryan McDonald, and Joakim Nivre (2009). *Dependency Parsing*. San Rafael, CA: Morgan & Claypool Publishers.

Lange, Martin, and Hans Leiß (2009). 'To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm'. *Informatica Didactica*, 8, 2008–2010.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič (2005). 'Non-projective dependency parsing using spanning tree algorithms'. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 523–530. Vancouver, Canada.

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

Manning, Christopher, and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Marcus, Mitchell (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge, MA: MIT Press.

Marcus, Mitchell, Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). 'Building a large annotated corpus of English: the Penn Treebank'. *Computational Linguistics*, 19(2), 313–330.

Maruyama, Hiroshi (1990). 'Structural disambiguation with constraint propagation'. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 31–38. Pittsburgh, PA, USA.

Maxwell, John III, and Ronald Kaplan (1993). 'The interface between phrasal and functional constraints'. *Computational Linguistics*, 19(4), 571–590.

Mel'čuk, Igor (1987). *Dependency Syntax: Theory and Practice*. Albany, NY: SUNY Press.

Nivre, Joakim, and Mario Scholz (2004). 'Deterministic dependency parsing of English text'. In *Proceedings of the 20th International Conference on Computational Linguistics*, 64–70.

Geneva, Switzerland.

Oepen, Stephan, and John Carroll (2000). 'Ambiguity packing in constraint-based grammar — practical results'. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics*, 162–169. Seattle, WA, USA.

Oepen, Stephan, and Daniel Flickinger (1998). 'Towards systematic grammar profiling: Test suite technology ten years after'. *Computer Speech and Language*, 12(4), 411–436.

Pereira, Fernando, and Yves Schabes (1992). 'Inside-outside re-estimation for partially bracketed corpora'. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 128–135. Newark, DE, USA.

Petrov, Slav, and Dan Klein (2007). 'Improved inference for unlexicalized parsing'. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, 404–411. Rochester, NY, USA.

Pollard, Carl, and Ivan Sag (1994). *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press.

Ponvert, Elias, Jason Baldridge, and Katrin Erk (2011). 'Simple unsupervised grammar induction from raw text with cascaded finite state models'. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 1077–1086. Portland, Oregon, USA.

Roark, Brian, and Richard Sproat (2007). *Computational Approaches to Morphology and Syntax*. Oxford: Oxford University Press.

Sampson, Geoffrey (1986). 'A stochastic approach to parsing'. In *Proceedings of the 11th International Conference on Computational Linguistics*, 151–155. Bonn, Germany.

Scicluna, James, and Colin de la Higuera (2014). ‘PCFG induction for unsupervised parsing and language modelling’. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 1353–1362. Doha, Qatar.

Sheil, Beau (1976). ‘Observations on context-free parsing’. *Statistical Methods in Linguistics*. 71–109.

Sikkel, Klaas (1997). *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms*. Berlin: Springer-Verlag.

Tapanainen, Pasi, and Timo Järvinen (1997). ‘A non-projective dependency parser’. In *Proceedings of the 5th ACL Conference on Applied Natural Language Processing*, 64–71. Washington, DC, USA.

Tiedemann, Jörg (2014). ‘Rediscovering annotation projection for cross-lingual parser induction’. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING)*, 1854–1864. Dublin, Ireland.

Tomita, Masaru (1985). ‘An efficient context-free parsing algorithm for natural languages’. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 756–764. Los Angeles, CA, USA.

Torisawa, Kentaro, and Jun’ichi Tsujii (1996). ‘Computing phrasal signs in HPSG prior to parsing’. In *Proceedings of the 16th International Conference on Computational Linguistics*, 949–955. Copenhagen, Denmark.

Toutanova, Kristina, Christopher Manning, Stuart Shieber, Daniel Flickinger, and Stephan Oepen (2002). ‘Parse disambiguation for a rich HPSG grammar’. In *Proceedings of the 1st Workshop* *on Treebanks and Linguistic Theories*, 253–263. Sozopol, Bulgaria.

Xia, Fei (1999). ‘Extracting tree adjoining grammars from bracketed corpora’. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, 398–403. Beijing, China.

Yamada, Hiroyasu, and Yuji Matsumoto (2003). ‘Statistical dependency analysis with support vector machines’. In *Proceedings of the 8th International Workshop on Parsing Technologies*, 195–206. Nancy, France.

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

Yu, Juntao, Mohab Elkaref, and Bernd Bohnet (2015). ‘Domain adaptation for dependency parsing via self-training’. In *Proceedings of the 14th International Conference on Parsing Technologies*, 1–10. Bilbao, Spain.

Zeman, Daniel, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič (2012). ‘HamleDT: To parse or not to parse?’ In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*, 2735–2741. Istanbul, Turkey.

Glossary

backtrack: to explore a new search path by undoing decisions taken previously and choosing different outcomes.

chart: a table storing completely and/or partially recognised constituents during a parse.

context-free backbone: an approximate context-free representation of a more fine-grained phrase structure grammar, without feature structure (or other) augmentation.

context-free phrase structure grammar: a way of defining the syntactic structure of a language using rewrite rules in which each type of constituent is represented by a symbol (e.g. 'NP').

dependency grammar: a way of defining the syntactic structure of a language by specifying how words are related each other by directed dependency links.

dependency structure: a representation of the syntactic structure of a sentence in which each word is linked to the word that syntactically governs it.

deterministic: exploring a single search path and not backtracking.

disambiguation: selecting a plausible analysis for an ambiguous input.

feature structure: a recursively structured matrix of features and values encoding the grammatical properties of a constituent.

gold standard: a collection of test inputs, each manually annotated with the output desired from a natural language processing system.

grammar development environment: a computer system that supports a grammarian in writing, testing and maintaining a computational grammar.

history-based model: a probabilistic model for disambiguation using information from the history of parse decisions.

local ambiguity packing: representing a set of constituent phrases of the same syntactic type covering the same part of the input as a single entity in a parse forest.

non-deterministic: exploring more than one search path.

overacceptance: erroneously returning too many parses for a grammatical input.

overgeneration: erroneously returning one or more parses for an ungrammatical input.

parse forest: a compact representation of a set of complete parses, typically using local ambiguity packing and subtree sharing.

parsing: the process of analysing an input with the aim of producing one or more syntactic analyses (parses).

phrase structure tree: a representation of the syntactic structure of a sentence which records the constituent phrases and how they are structured hierarchically.

probabilistic CF grammar: a CF grammar in which each rule has an associated probability of being applied, usually derived from a treebank.

subsumption: a test determining whether a feature structure is equivalent to or more general than another.

subtree sharing: representing a sub-analysis only once in a parse forest even if it forms part of more than one higher level constituent phrase.

test suite: a set of test inputs used to monitor progress during development of a natural language processing system.

treebank: a syntactically annotated text corpus.

unification: a test determining whether two feature structures are compatible; if so, the result of merging their contents is returned.

well-formed substring table: see **chart**.

Bio

John Carroll is Professor of Computational Linguistics at the University of Sussex, UK. He is involved in research projects and collaborations in the areas of: natural language parsing; unsupervised and minimally supervised machine learning for natural language tasks; and applications of natural language processing, particularly to extracting information from clinical text. See `http://users.sussex.ac.uk/~johnca/` for more details.

Index

    forest

    tree

parser—see parsing

parsing

    algorithm

        active chart
        Cocke-Younger-Kasami (CYK)
        Earley
        maximum spanning tree
        passive chart
        shift-reduce
        tabular

    Berkeley Parser

    Charniak-Johnson Parser

    CoNLL-X shared task

    context-free (CF)

    dependency

    efficiency

    evaluation

    graph-based dependency

    LKB

    MaltParser

    MSTParser

    multilingual dependency

    RASP

    Stanford Parser

    statistical

    transition-based dependency

    unification-based

SIGDAT

SIGNLL

SIGPARSE

subsumption

subtree sharing

test suite

treebank

- CCGbank
- French Treebank
- LinGO Redwoods Treebank
- Penn Treebank
- Universal Dependencies

unification

well-formed substring table (WFST)

Abbreviations / Acronyms

CCG: Combinatory Categorial Grammar

CF: context-free

CYK: Cocke-Younger-Kasami

HPSG: Head-Driven Phrase Structure Grammar

LFG: Lexical Functional Grammar

PCFG: probabilistic context-free grammar

WFST: well-formed substring table