Contents

Chapter 3

Syntactic Parsing

Peter Ljunglöf

Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg, Sweden

Mats Wirén

Department of Linguistics, Stockholm University, Sweden

3.1 Introduction

This chapter presents basic techniques for grammar-driven natural language parsing, that is, analysing a string of words (typically a sentence) to determine its structural description according to a formal grammar. In most circumstances, this is not a goal in itself but rather an intermediary step for the purpose of further processing, such as the assignment of a meaning to the sentence. To this end, the desired output of grammar-driven parsing is typically a hierarchical, syntactic structure suitable for semantic interpretation (the topic of Chapter 4). The string of words constituting the input will usually have been processed in separate phases of tokenisation (Chapter 1) and lexical analysis (Chapter 2), which is hence not part of parsing proper.

To get a grasp of the fundamental problems discussed here, it is instructive to consider the ways in which parsers for natural languages differ from parsers for computer languages (for related discussion, see Steedman 1983 and Karttunen and Zwicky 1985). One such difference concerns the power of the grammar formalisms used – the *generative capacity*. Computer languages are usually designed so as to permit encoding by unambiguous grammars

and parsing in linear time of the length of the input. To this end, carefully restricted subclasses of context-free grammar are used, with the syntactic specification of ALGOL 60 (Backus et al. 1963) as a historical exemplar. In contrast, natural languages are typically taken to require more powerful devices, as first argued by Chomsky (1956).[1] One of the strongest cases for expressive power has been the occurrence of long-distance dependencies, as in English *wh*-questions:

Who did you sell the car to __? (3.1)

Who do you think that you sold the car to __? (3.2)

Who do you think that he suspects that you sold the car to __? (3.3)

In (3.1)–(3.3) it is held that the noun phrase "who" is displaced from its canonical position (indicated by "__") as indirect object of "sell". Since there is no clear limit as to how much material may be embedded between the two ends, as suggested by (3.2) and (3.3), linguists generally take the position that these dependencies might hold at unbounded distance. Although phenomena like this have at times provided motivation to move far beyond context-free power, several formalisms have also been developed with the intent purpose of making minimal increases to expressive power (see Section 3.2.4). Generally speaking, the reason for trying to limit expressiveness is to retain efficient parsability, that is, parsing in polynomial time of the length of the input. Additionally, for the purpose of determining the expressive power needed for linguistic formalisms, *strong generative capacity* (the structural descriptions assigned by the grammar) is usually considered more relevant than *weak generative capacity* (the sets of strings generated); compare Chomsky (1965, pp. 60–61).

A second difference concerns the extreme structural *ambiguity* of natural language. At any point in a pass through a sentence, there will typically be several grammar rules which might apply. A classic example is the following:

Put the block in the box on the table (3.4)

Assuming that "put" subcategorises for two objects, there are two possible analyses of (3.4):

Put the block [in the box on the table] (3.5)

Put [the block in the box] on the table (3.6)

If we add another prepositional phrase ("in the kitchen"), we get five analyses; if we add yet another, we get fourteen, and so on. Other examples of the same phenomenon are conjuncts and nominal compounding. As discussed in

[1] For a background on formal grammars and formal-language theory, see Hopcroft et al. (2006).

detail by Church and Patil (1982), "every-way ambiguous" constructions of this kind have a number of analyses which grows exponentially with the number of added components. Even though only one of them may be appropriate in a given context, the purpose of a general grammar might be to capture what is possible in *any* context. As a result of this, even the process of just returning all the possible analyses would lead to a combinatorial explosion. Thus, much of the work on parsing – hence, much of the following exposition – deals somehow or other with ways in which the potentially enormous search spaces can be efficiently handled, and how the most appropriate analysis can be selected (*disambiguation*). The latter problem also leads naturally to extensions of grammar-driven parsing with statistical inference, as dealt with in Chapter 11.

A third difference stems from the fact that natural language data are inherently *noisy*, both because of errors (under some conception of "error") and because of the ever persisting incompleteness of lexicon and grammar relative to the vastness of possible natural language constructions. In contrast, a computer language has a complete syntax specification, which means that by definition all correct input strings are parsable. Furthermore, the associated parsers usually have the *valid prefix property*, which means that they will detect an error at the earliest possible time, as soon as a prefix of the input has been seen for which there is no valid continuation(Aho et al. 2006). In natural language parsing, it is notoriously difficult to distinguish whether a failure to produce a parsing result is due to an error in the input or to lack of coverage of the grammar, also because a natural language by its nature has no precise delimitation. Thus, input not licensed by the grammar may well be perfectly adequate according to native speakers of the language. Moreover, input containing errors may still carry useful bits of information that might be desirable to try to recover. *Robustness* refers to the ability of always producing *some* result (also) in response to such input (Menzel 1995).

The rest of this chapter is organised as follows. Section 3.2 gives a background on grammar formalisms and basic concepts in natural language parsing, and introduces a small context-free grammar that is used in examples throughout. Section 3.3 presents a basic tabular algorithm for parsing with context-free grammar, the Cocke-Kasami-Younger algorithm. Section 3.4 then describes the main approaches to tabular parsing in an abstract way, in the form of "parsing as deduction", again using context-free grammar. Section 3.5 discusses some implementational issues in relation to this abstract framework. Section 3.6 then goes on to describing LR parsing, and its nondeterministic generalisation GLR parsing. Section 3.7 introduces a simple form of constraint-based grammar and describes tabular parsing using this kind of grammar formalism. Section 3.8 discusses in some further depth the three main challenges in natural language parsing that have been touched upon in this introductory section – robustness, disambiguation and efficiency. Finally, Section 3.9 provides some brief historical notes on parsing relative to where we stand today.

3.2 Background

This section introduces grammar formalisms, primarily context-free grammars, and basic parsing concepts, which will be used in the rest of this chapter.

3.2.1 Context-Free Grammars

Ever since its introduction by Chomsky (1956), context-free grammar (CFG) has been the most influential grammar formalism for describing language syntax. This is not because CFG has been generally adopted as such, but rather because most grammar formalisms are derived from or can somehow be related to CFG. For this reason, CFG is often used as a base formalism when parsing algorithms are described.

The standard way of defining a context-free grammar is as a tuple $G = \langle \Sigma, N, S, R \rangle$, where Σ and N are disjoint finite sets of *terminal* and *nonterminal* symbols, respectively, and $S \in N$ is the *start symbol*. The nonterminals are also called *categories*, and the set $V = N \cup \Sigma$ contains the *symbols* of the grammar. R is a finite set of *production rules* of the form $A \to \alpha$, where $A \in N$ is a nonterminal and $\alpha \in V^*$ is a sequence of symbols.

We use capital letters A, B, C, ... for nonterminals, lower-case letters s, t, w, ... for terminal symbols, and upper-case X, Y, Z, ... for general symbols (elements in V). Greek letters α, β, γ, ... will be used for sequences of symbols, and we write ϵ for the empty sequence.

The rewriting relation $\Rightarrow$ is defined by $\alpha B \gamma \Rightarrow \alpha\beta\gamma$ if and only if $B \to \beta$. A *phrase* is a sequence of terminals $\beta \in \Sigma^*$ such that $A \Rightarrow \cdots \Rightarrow \beta$ for some $A \in N$. Accordingly, the term *phrase-structure grammar* is sometimes used for grammars with at least context-free power. The sequence of rule expansions is called a *derivation* of β from A. A (*grammatical*) *sentence* is a phrase which can be derived from the start symbol S. The *string language* $L(G)$ accepted by G is the set of sentences of G.

Some algorithms only work for particular *normal forms* of context-free grammars:

- In Section 3.3 we will use grammars in *Chomsky normal form* (CNF). A grammar is in CNF when each rule is either (*i*) a unary terminal rule of the form $A \to w$, or (*ii*) a binary nonterminal rule of the form $A \to B\,C$. It is always possible to transform a grammar into CNF such that it accepts the same language.[2] However, the transformation can change the structure of the grammar quite radically; e.g., if the original

[2]Formally, only grammars that do not accept the empty string can be transformed into CNF, but from a practical point of view we can disregard this, as we are not interested in empty string languages.

S	→	NP VP	Det	→	*a* | *an* | *the*
NP	→	Det NBar	Adj	→	*old*
NBar	→	Adj Noun	Noun	→	*man* | *men* | *ship* | *ships*
NBar	→	Noun	Verb	→	*man* | *mans*
NBar	→	Adj			
VP	→	Verb			
VP	→	Verb NP			

FIGURE 3.1: Example grammar

grammar has n rules, the transformed version may in the worst case have $O(n^2)$ rules (Hopcroft et al. 2006).

- We can relax this normal form by allowing (*iii*) unary nonterminal rules of the form $A \rightarrow B$. The transformation to this form is much simpler, and the transformed grammar is structurally closer; e.g., the transformed grammar will have only $O(n)$ rules. This relaxed variant of CNF is also used in Section 3.3.

- In Section 3.4 we relax the normal form even further, such that each rule is either (*i*) a unary terminal rule of the form $A \rightarrow w$, or (*ii*) a non-empty nonterminal rule of the form $A \rightarrow B_1 \cdots B_d$ ($d > 0$).

- In Section 3.6, the only restriction is that the rules are non-empty.

We will not describe how transformations are carried out here, but refer to any standard textbook on formal languages, such as Hopcroft et al. (2006).

3.2.2 Example Grammar

Throughout this chapter we will make use of a single (toy) grammar in our running examples. The grammar is shown in figure 3.1, and is on Chomsky normal form relaxed according to the first relaxation condition above. Thus it only contains unary and binary nonterminal rules, and unary terminal rules. The right-hand sides of the terminal rules correspond to *lexical items*, whereas the left-hand sides are *preterminal* (or *part-of-speech*) symbols. In practice, lexical analysis is often carried out in a phase distinct from parsing (as described in Chapter 2); the preterminals then take the role of terminals during parsing. The example grammar is lexically ambiguous, since the word "man" can be a noun as well as a verb. Hence, the *garden path* sentence "the old man a ship", as well as the more intuitive "the old men man a ship", can be recognised using this grammar.

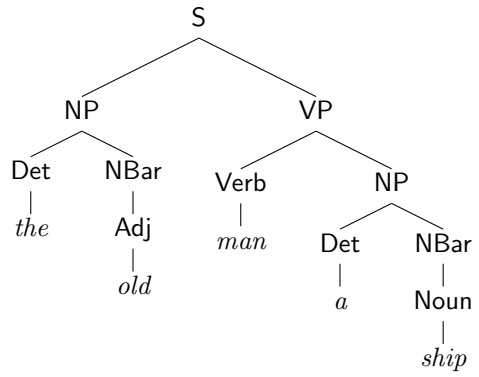

FIGURE 3.2: Syntax tree of the sentence "the old man a ship"

3.2.3 Syntax Trees

The standard way to represent the syntactic structure of a grammatical sentence is as a *syntax tree*, or a parse tree, which is a representation of all the steps in the derivation of the sentence from the root node. This means that each internal node in the tree represents an application of a grammar rule. The syntax tree of the example sentence "the old man a ship" is shown in figure 3.2. Note that the tree is drawn upside-down, with the root of the tree at the top and the leaves at the bottom.

Another representation, which is commonly used in running text, is as a *bracketed* sentence, where the brackets are labelled with nonterminals:

[S [NP [Det *the*] [NBar [Adj *old*]]] [VP [Verb *man*] [NP [Det *a*] [NBar [Noun *ship*]]]]]

3.2.4 Other Grammar Formalisms

In practice, pure CFG is not widely used for developing natural language grammars (though grammar-based language modelling in speech recognition is one such case; see Chapter 15). One reason for this is that CFG is not expressive enough – it cannot describe all peculiarities of natural language, e.g., Swiss-German or Dutch scrambling (Shieber 1985a), or Scandinavian long-distance dependencies (Kaplan and Zaenen 1995). But the main practical reason is that it is difficult to use; e.g., agreement, inflection and other common phenomena are complicated to describe using CFG.

The example grammar in fig 3.1 is overgenerating – it recognises both the noun phrases "a men" and "an man", as well as the sentence "the men mans a ship". However, to make the grammar syntactically correct, we must duplicate the categories Noun, Det and NP into singular and plural versions. All grammar rules involving these categories must be duplicated too. And if the

language is, e.g., German, then Det and Noun have to be inflected on number (SING/PLUR), gender (FEM/NEUTR/MASC) and case (NOM/ACC/DAT/GEN).

Ever since the late 1970s, a number of extensions to CFGs have emerged, with different properties. Some of these formalisms, for example, Regulus and Generalised Phrase-Structure Grammar (GPSG), are context-free-equivalent, meaning that grammars can be compiled to an equivalent CFG which then can be used for parsing. Other formalisms, such as Head-driven Phrase-Structure Grammar (HPSG) and Lexical-Functional Grammar (LFG), have more expressive power, but their similarities with CFG can still be exploited when designing tailor-made parsing algorithms.

There are also several grammar formalisms (e.g., categorial grammar, TAG, dependency grammar) that have not been designed as extensions of CFG, but have other pedigrees. However, most of them have been shown later to be equivalent to CFG or some CFG extension. This equivalence can then be exploited when designing parsing algorithms for these formalisms.

Mildly context-sensitive grammars According to Chomsky's hierarchy of grammar formalisms (Chomsky 1959), the next major step after context-free grammar is *context-sensitive grammar*. Unfortunately, this step is substantial; arguably, context-sensitive grammars can express an unnecessarily large class of languages, with the drawback that parsing is no longer polynomial in the length of the input. Joshi (1985) suggested the notion of *mild context-sensitivity* to capture the precise formal power needed for defining natural languages. Roughly, a grammar formalism is regarded as mildly context-sensitive (MCS) if it can express some linguistically motivated non-context-free constructs (multiple agreement, crossed agreement and duplication), and can be parsed in polynomial time with respect to the input.

Among the most restricted MCS formalisms are Tree-Adjoining Grammar (TAG; Joshi et al. 1975; Joshi and Schabes 1997) and Combinatory Categorial Grammar (CCG; Steedman 1985, 1986), which are equivalent to each other (Vijay-Shanker and Weir 1994). Extending these formalisms we obtain a hierarchy of MCS grammar formalisms, with an upper bound in the form of Linear Context-Free Rewriting Systems (LCFRS; Vijay-Shanker et al. 1987), Multiple Context-Free Grammar (MCFG; Seki et al. 1991) and Range Concatenation Grammar (RCG; Boullier 2004), among others.

Constraint-based formalisms A key characteristic of constraint-based grammars is the use of feature terms (sets of attribute–value pairs) for the description of linguistic units, rather than atomic categories as in CFG. Feature terms are partial (underspecified) in the sense that new information may be added as long as it is compatible with old information. Regulus (Rayner et al. 2006) and Generalised Phrase-Structure Grammar (GPSG; Gazdar et al. 1985) are examples of constraint-based formalisms which are context-free equivalent, whereas Head-driven Phrase-Structure Grammar (HPSG; Pol-

lard and Sag 1994) and Lexical-Functional Grammar (LFG; Bresnan 2001) are strict extensions of CFG. Not only CFG can be augmented with feature terms – constraint-based variants of, e.g., TAG and dependency grammars also exist. Constraint-based grammars are further discussed in Section 3.7.

Immediate dominance/linear precedence When describing languages with a relatively free word order, such as Latin, Finnish or Russian, it can be fruitful to separate *immediate dominance* (ID; the parent-child relation) from *linear precedence* (LP; the linear order between the children) within phrases. The first formalism to make use of the ID/LP distinction was Generalised Phrase-Structure Grammar (GPSG; Gazdar et al. 1985), and it has also been used in HPSG and other recent grammar formalisms. The main problem with ID/LP formalisms is that parsing can become very expensive. Some work has therefore been done to devise ID/LP formalisations which are easier to parse (Nederhof and Satta 2004a; Daniels and Meurers 2004).

Head grammars In some linguistic theories there is a notion of the syntactic *head* of a phrase; e.g., the head of a verb phrase could be argued to be the main verb, whereas the head of a noun phrase could be the main noun. The simplest head grammar formalism is to mark one right-hand side symbol in each context-free rule; more advanced examples include HPSG. The head information can, e.g., be used for driving the parser by trying to find the head first and then its arguments (Kay 1989).

Lexicalised grammars The nonterminals in a CFG do not depend on the lexical words at the surface level. This is a standard problem for *PP attachment* – which noun phrase or verb phrase constituent a specific prepositional phrase should be attached to. E.g., considering a sentence beginning with "I bought a book...", it is clear that a following PP "...with my credit card" should be attached to the verb "bought", whereas the PP "...with an interesting title" should attach to the noun "book". To be able to express such lexical syntactic preferences, context-free grammars and other formalisms can be lexicalised in different ways (Eisner and Satta 1999; Eisner 2000; Joshi and Schabes 1997).

Dependency grammars In contrast to constituent-based formalisms, *dependency grammar* lacks phrasal nodes; instead the structure consists of lexical elements linked by binary dependency relations (Tesnière 1959; Nivre 2006). A dependency structure is a directed acyclic graph between the words in the surface sentence, where the edges are labelled with syntactic functions (such as SUBJ, OBJ, MOD, etc.). Apart from this basic idea, the dependency grammar tradition constitutes a diverse family of different formalisms which can impose different constraints on the dependency relation (such as allowing

or disallowing crossing edges), and incorporate different extensions (such as feature terms).

Type-theoretical grammars Some formalisms are based on dependent type theory utilising the Curry–Howard isomorphism between propositions and types. These formalisms include ALE (Carpenter 1992), Grammatical Framework (Ranta 1994, 2004; Ljunglöf 2004) and Abstract Categorial Grammar (de Groote 2001).

3.2.5 Basic Concepts in Parsing

A *recogniser* is a procedure which determines whether or not an input sentence is grammatical according to the grammar (including the lexicon). A *parser* is a recogniser which produces associated structural analyses according to the grammar (in our case, parse trees or feature terms). A *robust parser* attempts to produce useful output, such as a partial analysis, even if the input is not covered by the grammar. More generally, it also has the ability to produce a single analysis even if the input is highly ambiguous (see Section 3.8.1).

We can think of a grammar as inducing a search space consisting of a set of states representing stages of successive grammar-rule rewritings and a set of transitions between these states. When analysing a sentence, the parser (recogniser) must rewrite the grammar rules in some sequence. A sequence which connects the state S, the string consisting of just the start category of the grammar, and a state consisting of exactly the string of input words, is called a *derivation*. Each state in the sequence then consists of a string over V^* and is called a *sentential form*. If such a sequence exists, the sentence is said to be grammatical according to the grammar.

Parsers can be classified along several dimensions according to the ways in which they carry out derivations. One such dimension concerns rule invocation: In a *top-down* derivation, each sentential form is produced from its predecessor by replacing one nonterminal symbol A by a string of terminal or nonterminal symbols $X_1 \cdots X_d$, where $A \to X_1 \cdots X_d$ is a grammar rule. Conversely, in a *bottom-up* derivation, each sentential form is produced by replacing $X_1 \cdots X_d$ with A given the same grammar rule, thus successively applying rules in the reverse direction.

Another dimension concerns the way in which the parser deals with ambiguity, in particular, whether the process is *deterministic* or *nondeterministic*. In the former case, only a single, irrevocable choice may be made when the parser is faced with local ambiguity. This choice is typically based on some form of lookahead or systematic preference.

A third dimension concerns whether parsing proceeds from left to right (strictly speaking front to back) through the input or in some other order, for example, inside-out from the right-hand-side heads.

3.3 The Cocke-Kasami-Younger Algorithm

The Cocke-Kasami-Younger (CKY, sometimes written CYK) algorithm, first described in the 1960s (Kasami 1965; Younger 1967), is one of the simplest context-free parsing algorithms. A reason for its simplicity is that it only works for grammars in Chomsky normal form (CNF).

The CKY algorithm builds an upper triangular matrix $\mathcal{T}$, where each cell $\mathcal{T}_{i,j} (0 \leq i < j \leq n)$ is a set of nonterminals. The meaning of the statement $A \in \mathcal{T}_{i,j}$ is that A spans the input words $w_{i+1} \cdots w_j$, or written more formally, $A \Rightarrow^* w_{i+1} \cdots w_j$.

CKY is a purely bottom-up algorithm consisting of two parts. First we build the lexical cells $\mathcal{T}_{i-1,i}$ for the input word w_i by applying the lexical grammar rules, then the nonlexical cells $\mathcal{T}_{i,k}$ $(i < k-1)$ are filled by applying the binary grammar rules:

$$\mathcal{T}_{i-1,i} = \{\, A \mid A \rightarrow w_i \,\}$$
$$\mathcal{T}_{i,k} = \{\, A \mid A \rightarrow B\,C,\, i < j < k,\, B \in \mathcal{T}_{i,j},\, C \in \mathcal{T}_{j,k} \,\}$$

The sentence is recognised by the algorithm if $S \in \mathcal{T}_{0,n}$, where S is the start symbol of the grammar.

To make the algorithm less abstract, we note that all cells $\mathcal{T}_{i,j}$ and $\mathcal{T}_{j,k}$ $(i < j < k)$ must already be known when building the cell $\mathcal{T}_{i,k}$. This means that we have to be careful when designing the i and k loops, so that smaller spans are calculated before larger spans.

One solution is to start by looping over the end node k, and then loop over the start node i in the *reverse* direction. The pseudo-code is as follows:

procedure CKY($\mathcal{T}, w_1 \cdots w_n$)
  $\mathcal{T}_{i,j} := \emptyset$ **for all** $0 \leq i, j \leq n$
  for $i := 1$ **to** n **do**
    for all lexical rules $A \rightarrow w$ **do**
      if $w = w_i$ **then** add A to $\mathcal{T}_{i-1,i}$
  for $k := 2$ **to** n **do**
    for $i := k-2$ **downto** 0 **do**
      for $j := i+1$ **to** $k-1$ **do**
        for all binary rules $A \rightarrow B\,C$ **do**
          if $B \in \mathcal{T}_{i,j}$ and $C \in \mathcal{T}_{j,k}$ **then** add A to $\mathcal{T}_{i,k}$

But there are also several alternative possibilities for how to encode the loops in the CKY algorithm; e.g., instead of letting the outer k loop range over end positions, we could equally well let it range over span lengths. We have to keep in mind, however, that smaller spans must be calculated before larger spans.

As already mentioned, the CKY algorithm can only handle grammars in Chomsky normal form. Furthermore, converting a grammar to CNF is a bit

complicated, and can make the resulting grammar much larger, as mentioned in Section 3.2.1. Instead we will show how to modify the CKY algorithm directly to handle unary grammar rules and longer right-hand sides.

3.3.1 Handling Unary Rules

The CKY algorithm can only handle grammars with rules of the form $A \rightarrow w_i$ and $A \rightarrow B\,C$. Unfortunately most practical grammars also contain lots of unary rules of the form $A \rightarrow B$. There are two possible ways to solve this problem. Either we transform the grammar into CNF, or we modify the CKY algorithm.

If $B \in \mathcal{T}_{i,k}$ and there is a unary rule $A \rightarrow B$, then we should also add A to $\mathcal{T}_{i,k}$. Furthermore, the unary rules can be applied after the binary rules, since binary rules only apply to smaller phrases. Unfortunately, we cannot simply loop over each unary rule $A \rightarrow B$ to test if $B \in \mathcal{T}_{i,k}$. The reason for this is that we cannot possibly know in which order the unary rules will be applied, which means that we cannot know in which order we have to select the unary rules $A \rightarrow B$. Instead we need to add the *reflexive, transitive closure* UNARY-CLOSURE$(B) = \{\, A \mid A \Rightarrow^* B \,\}$ for each $B \in \mathcal{T}_{i,k}$. Since there are only a finite number of nonterminals, UNARY-CLOSURE() can be precompiled from the grammar into an efficient lookup table.

Now, the only thing we have to do is to map UNARY-CLOSURE() onto $\mathcal{T}_{i,k}$ within the k and i loops, and after the j loop (as well as onto $\mathcal{T}_{i-1,i}$ after the lexical rules have been applied). The final pseudo-code for the extended CKY algorithm is:

procedure UNARY-CKY($\mathcal{T}, w_1 \cdots w_n$)
  $\mathcal{T}_{i,j} := \emptyset$ **for all** $0 \leq i, j \leq n$
  for $i := 1$ **to** n **do**
    for all lexical rules $A \rightarrow w$ **do**
      if $w = w_i$ **then** add A to $\mathcal{T}_{i-1,i}$
    for all $B \in \mathcal{T}_{i-1,i}$ **do**
      add UNARY-CLOSURE(B) to $\mathcal{T}_{i-1,i}$
  for $k := 2$ **to** n **do**
    for $i := k - 2$ **downto** 0 **do**
      for $j := i + 1$ **to** $k - 1$ **do**
        for all binary rules $A \rightarrow B\,C$ **do**
          if $B \in \mathcal{T}_{i,j}$ and $C \in \mathcal{T}_{j,k}$ **then** add A to $\mathcal{T}_{i,k}$
      for all $B \in \mathcal{T}_{i,k}$ **do**
        add UNARY-CLOSURE(B) to $\mathcal{T}_{i,k}$

3.3.2 Example Session

The final CKY matrix after parsing the example sentence "the old man a ship" is shown in figure 3.3. In the initial lexical pass, the cells in the first

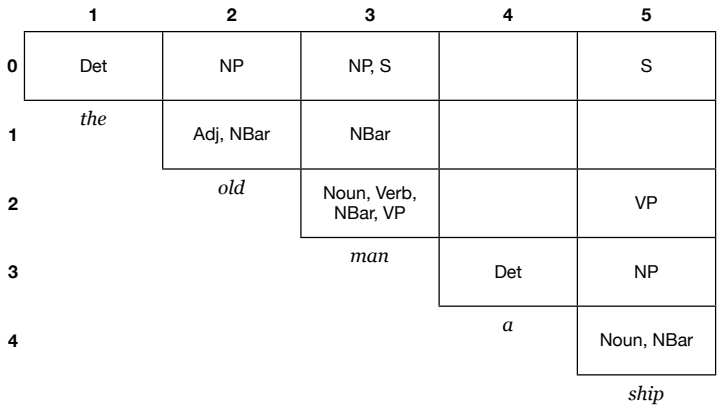

FIGURE 3.3: CKY matrix after parsing the sentence "the old man a ship".

diagonal are filled. E.g., the cell $\mathcal{T}_{2,3}$ is initialised to $\{\mathsf{Noun}, \mathsf{Verb}\}$, after which UNARY-CLOSURE() adds NBar and VP to it.

Then other cells are filled from left to right, bottom up. E.g., when filling the cell $\mathcal{T}_{0,3}$, we have already filled $\mathcal{T}_{0,2}$ and $\mathcal{T}_{1,3}$. Now, since $\mathsf{Det} \in \mathcal{T}_{0,1}$ and $\mathsf{NBar} \in \mathcal{T}_{1,3}$, and there is a rule $\mathsf{NP} \to \mathsf{Det}\,\mathsf{NBar}$, NP is added to $\mathcal{T}_{0,3}$. And since $\mathsf{NP} \in \mathcal{T}_{0,2}$, $\mathsf{VP} \in \mathcal{T}_{2,3}$ and $\mathsf{S} \to \mathsf{NP}\,\mathsf{VP}$, the algorithm adds S to $\mathcal{T}_{0,3}$.

3.3.3 Handling Long Right-Hand Sides

To handle longer right-hand sides (RHS), there are several possibilities. A straightforward solution is to add a new inner loop for each RHS length. This means that, e.g., ternary rules will be handled by the following loop inside the k, i and j nested loops:

for $k, i, j := \ldots$ **do**
  for all binary rules $\ldots$ **do** $\ldots$
  for $j' := j + 1$ to $k - 1$ **do**
    for all ternary rules $A \to B\,C\,D$ **do**
      if $B \in \mathcal{T}_{i,j}$ and $C \in \mathcal{T}_{j,j'}$ and $D \in \mathcal{T}_{j',k}$ **then**
        add A to $\mathcal{T}_{i,k}$

To handle even longer rules we need to add new inner loops inside the j' loop. And for each nested loop, the parsing time increases. In fact, the worst case complexity is $O(n^{d+1})$, where d is the length of the longest right-hand side. This is discussed further in Section 3.8.3.

A more general solution is to replace each long rule $A \to B_1 \cdots B_d$ $(d > 2)$ by the $d - 1$ binary rules $A \to B_1\,X_2$, $X_2 \to B_2\,X_3$, $\ldots$, $X_{d-1} \to B_{d-1}\,B_d$, where each $X_i = \langle B_i \cdots B_n \rangle$ is a new nonterminal. After this transformation

the grammar only contains unary and binary rules, which can be handled by the extended CKY algorithm.

Another variant of the binary transform is to do the RHS transformations implicitly during parsing. This gives rise to the well known chart parsing algorithms which we introduce in the next section.

3.4 Parsing as Deduction

In this section we shall use a general framework for describing parsing algorithms in a high-level manner. The framework is called *deductive parsing*, and was introduced by Pereira and Warren (1983a); a related framework introduced later was the *parsing schemata* of Sikkel (1998). Parsing in this sense can be seen as "a deductive process in which rules of inference are used to derive statements about the grammatical status of strings from other such statements" (Shieber et al. 1995).

3.4.1 Deduction Systems

The statements in a deduction system are called *items*, and are represented by formulae in some formal language. The inference rules and axioms are written in natural deduction style and can have side conditions mentioning, e.g., grammar rules. The inference rules and axioms are rule schemata; in other words, they contain metavariables to be instantiated by appropriate terms when the rule is invoked. The set of items built in the deductive process is sometimes called a *chart*.

The general form of an inference rule is

$$\frac{e_1 \quad \cdots \quad e_n}{e} \; \phi$$

where e, e_1, ..., e_n are items and ϕ is a side condition. If there are no antecedents (i.e., $n = 0$), the rule is called an *axiom*. The meaning of an inference rule is that whenever we have derived the items e_1, ..., e_n, and the condition ϕ holds, we can also derive the item e. The inference rules are applied until no more items can be added. It does not make any difference in which order the rules are applied – the final chart is the reflexive, transitive closure of the inference rules. However, one important constraint is that the system is terminating, which is the case if the number of possible items is finite.

3.4.2 The CKY Algorithm

As a first example, we describe the extended CKY algorithm from Section 3.3.1 as a deduction system. The items are of the form $[\,i, k : A\,]$, corresponding to a non-terminal symbol A spanning the input words $w_{i+1} \cdots w_k$. This is equivalent to the statement $A \in \mathcal{T}_{i,k}$ in the previous section. We need three inference rules, of which one is an axiom.

Combine

$$\frac{[\,i, j : B\,] \quad [\,j, k : C\,]}{[\,i, k : A\,]}\ A \to B\,C \tag{3.7}$$

If there is a B spanning the input positions $i - j$, and a C spanning $j - k$, and there is a binary rule $A \to B\,C$, we know that A will span the input positions $i - k$.

Unary closure

$$\frac{[\,i, k : B\,]}{[\,i, k : A\,]}\ A \to B \tag{3.8}$$

If we have a B spanning $i - k$, and there is a rule $A \to B$, then we also know thath there is an A spanning $i - k$.

Scan

$$\frac{}{[\,i-1, i : A\,]}\ A \to w_i \tag{3.9}$$

Finally we need an axiom adding an item for each matching lexical rule.

Note that we do not have to say anything about the order in which the inference rules should be applied, as was the case when we presented the CKY algorithm in the previous section.

3.4.3 Chart Parsing

The CKY algorithm uses a bottom-up parsing strategy, which means that it starts by recognising the lexical nonterminals, i.e., the nonterminals that occur as left-hand sides in unary terminal rules. Then the algorithm recognises the parents of the lexical nonterminals, and so on until it reaches the starting symbol.

The problem with CKY is that it only works on restricted grammars. General context-free grammars have to be converted, which is not a difficult problem, but can be awkward. The resulting parse results also have to be back-translated into the original form. For these and other reasons, one often implements more general parsing strategies instead.

In the following we give examples of some well-known parsing algorithms for context-free grammars. First we give a very simple algorithm, and then two refinements; Kilbury's bottom-up algorithm (Leiss 1990), and Earley's top-down algorithm (Earley 1970). The algorithms are slightly modified for presentational purposes, but their essence is still the same.

Parse items

Parse items are of the form $[\, i, j : A \to \alpha \bullet \beta \,]$ where $A \to \alpha\beta$ is a context-free rule, and $0 \leq i \leq j \leq n$ are positions in the input string. The meaning is that α has been recognised spanning $i - j$; i.e., $\alpha \Rightarrow^* w_{i+1} \cdots w_j$. If β is empty, the item is called *passive*. Apart from the logical meaning, the item also states that it is searching for β to span the positions j and k (for some k). The goal of the parsing process is to deduce an item representing that the starting category is found spanning the whole input string; such an item can be written $[\, 0, n : S \to \alpha \bullet \,]$ in our notation.

To simplify presentation, we will assume that all grammars are of the relaxed normal form presented in Section 3.2.1, where each rule is either lexical $A \to w$ or nonempty $A \to B_1 \cdots B_d$. To extend the algorithms to cope with general grammars constitutes no serious problem.

The simplest chart parsing algorithm

Our first context-free chart parsing algorithm consists of three inference rules. The first two, Combine and Scan, remain the same in all our chart parsing variants; while the third, Predict, is very simple and will be improved upon later. The algorithm is also presented by Sikkel and Nijholt (1997), who call it *bottom-up Earley* parsing.

Combine

$$\frac{[\, i, j : A \to \alpha \bullet B\beta \,] \quad [\, j, k : B \to \gamma \bullet \,]}{[\, i, k : A \to \alpha B \bullet \beta \,]} \tag{3.10}$$

The basis for all chart parsing algorithms is *the fundamental rule*; saying that if there is an active item looking for a category B spanning $i-j$, and there is a passive item for B spanning $j - k$, then the dot in the active item can be moved forward, and the new item will span the positions $i - k$.

Scan

$$\frac{}{[\, i-1, i : A \to w_i \bullet \,]}\ A \to w_i \tag{3.11}$$

This is similar to the scanning axiom of the CKY algorithm.

Predict

$$\frac{}{[\, i, i : A \to \bullet \beta \,]}\ A \to \beta \tag{3.12}$$

This axiom takes care of introducing active items; each rule in the grammar is added as an active item spanning $i - i$ for any possible input position $0 \leq i \leq n$.

The main problem with this algorithm is that prediction is "blind"; active items are introduced for every rule in the grammar, at all possible input positions. Only very few of these items will be used in later inferences, which means that prediction infers a lot of useless items. The solution is to make prediction an inference rule instead of an axiom, so that an item is only predicted if it is potentially useful for already existing items.

In the rest of this section we introduce two basic prediction strategies, *bottom-up* and *top-down*.

3.4.4 Bottom-Up Left-Corner Parsing

The basic idea with bottom-up parsing is that we predict a grammar rule only when its first symbol has already been found. Kilbury's variant of bottom-up parsing (Leiss 1990) moves the dot in the new item forward one step. Since the first symbol in the right-hand side is called the left corner, the algorithm is sometimes called *bottom-up left-corner parsing* (Sikkel 1998).

Bottom-up predict

$$\frac{[\,i, k : B \rightarrow \gamma \bullet\,]}{[\,i, k : A \rightarrow B \bullet \beta\,]}\ A \rightarrow B\,\beta \tag{3.13}$$

Bottom-up prediction is like Combine for the first symbol on the right-hand side in a rule. If we have found a B spanning $i - k$, and there is a rule $A \rightarrow B\,\beta$, we can draw the conclusion that $A \rightarrow B \bullet \beta$ will span $i - k$.

Note that this algorithm does not work for grammars with ϵ-rules; there is no way an empty rule can be predicted. There are two possible ways ϵ-rules can be handled: *(i)* either convert the grammar to an equivalent ϵ-free grammar; or *(ii)* add extra inference rules to handle ϵ-rules.

3.4.5 Top-Down Earley-style Parsing

Earley prediction (Earley 1970) works in a top-down fashion; meaning that we start by stating that we want to find an S starting in position 0, and then move downwards in the presumptive syntactic structure until we reach the lexical tokens.

Top-down predict

$$\frac{[\,i, k : B \rightarrow \gamma \bullet A\alpha\,]}{[\,k, k : A \rightarrow \bullet\,\beta\,]}\ A \rightarrow \beta \tag{3.14}$$

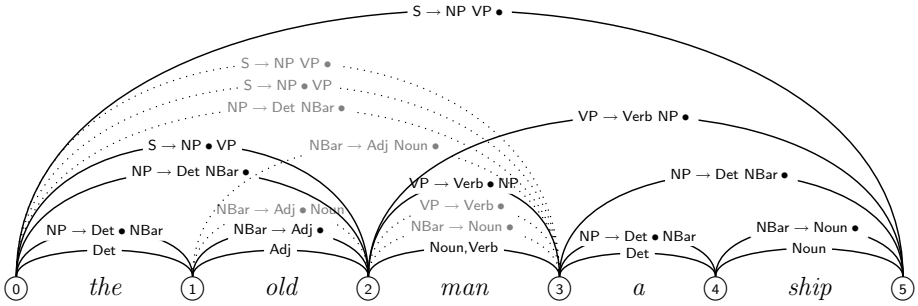

FIGURE 3.4: Final chart after bottom-up parsing of the sentence "the old man a ship". The dotted edges are inferred but useless.

If there is an item looking for an A beginning in position k, and there is a grammar rule for A, we can add that rule as an empty active item starting and ending in k.

Initial predict

$$\frac{}{[\,0,0 : S \rightarrow \bullet\, \beta\,]}\; S \rightarrow \beta \tag{3.15}$$

Top-down prediction needs an active item to be triggered, so we need some way of starting the inference process. This is done by adding an active item for each rule of the starting category S, starting and ending in 0.

3.4.6 Example Session

The final charts after bottom-up and top-down parsing of the example sentence "the old man a ship" are shown in figures 3.4 and 3.5. This is a standard way of visualising a chart, as a graph where the items are drawn as edges between the input positions. In the figures, the dotted and grayed-out edges correspond to useless items, i.e., items that are not used in any derivation of the final S item spanning the whole sentence.

The bottom-up chart contains the useless item $[\,2,3 : \mathsf{NBar} \rightarrow \mathsf{Noun} \bullet\,]$, which the top-down chart does not contain. One the other hand, the top-down chart contains a lot of useless cyclic predictions. This suggests that both bottom-up and top-down parsing have their advantages and disadvantages, and that combining the strategies could be the way to go. This leads us directly into the next section about dynamic filtering.

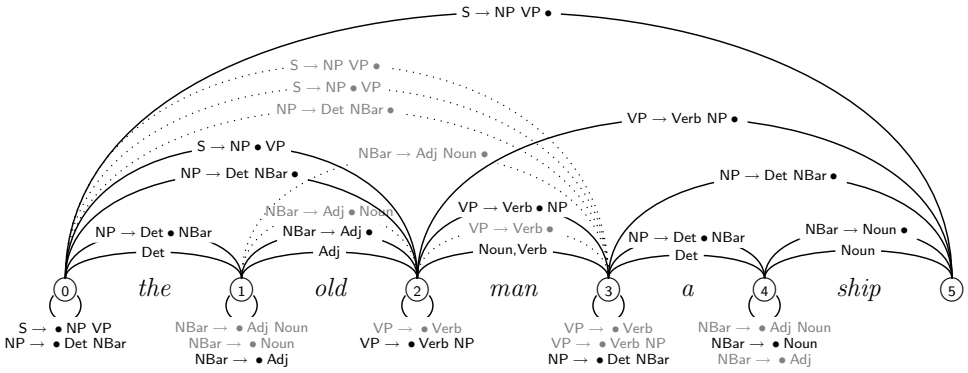

FIGURE 3.5: Final chart after top-down parsing of the sentence "the old man a ship". The dotted edges are inferred but useless.

3.4.7 Dynamic Filtering

Both the bottom-up and the top-down algorithm have disadvantages. Bottom-up prediction has no idea of what the final goal of parsing is, which means that it predicts items which will not be used in any derivation from the top node. Top-down prediction on the other hand never looks at the input words, which means that it predicts items which can never start with the next input word.

Note that these useless items do not make the algorithms incorrect in any way; they only decrease parsing efficiency. There are several ways the basic algorithms can be optimised; the standard optimizations are by adding top-down and/or bottom-up *dynamic filtering* to the prediction rules.

Bottom-up filtering adds a side condition stating that a prediction is only allowed if the resulting item can start with the next input token. For this we make use of a function FIRST() which returns the set of terminals than can start a given symbol sequence. The only thing we have to do is to add a side condition $w_k \in \text{FIRST}(\beta)$ to top-down prediction (3.14), and bottom-up prediction (3.13), respectively.[3] Possible defintions of the function FIRST() can be found in standard textbooks (Aho et al. 2006; Hopcroft et al. 2006).

Top-down filtering In a similar way, we can add constraints for top-down filtering of the bottom-up strategy. This means that we only have to add a constraint to bottom-up prediction (3.13) that there is an item looking for a C, where $C \Rightarrow^* A\delta$ for some δ. This *left-corner* relation can

[3]There is nothing that prevents us from adding a bottom-up filter to the combine rule (3.10) either. However, this filter is seldom used in practice.

be precompiled from the grammar, and the resulting parsing strategy is often called left-corner parsing (Sikkel and Nijholt 1997; Moore 2004).

Furthermore, both bottom-up and top-down filtering can be added as side-conditions to bottom-up prediction (3.13). Further optimizations in this direction, such as introducing special predict items and realizing the parser as an incremental algorithm, are discussed by Moore (2004).

3.5 Implementing Deductive Parsing

This section briefly discusses how to implement the deductive parsing framework, including how to store and retrieve parse results.

3.5.1 Agenda-driven Chart Parsing

A deduction engine should infer all consequences of the inference rules. As mentioned above, the set of all resulting items is called a *chart*, and can be calculated using a forward-chaining deduction procedure. Whenever an item is added to the chart, its consequences are calculated and added. However, since one item can give rise to several new items, we need to keep track of the items that are waiting to be processed. New items are thus added to a separate agenda which is used for bookkeeping.

The idea is as follows: First we add all possible consequences of the axioms to the agenda. Then we remove one item e from the agenda, add it to the chart and add all possible inferences that are trigged by e to the agenda. This second step is repeated until the agenda is empty.

Regarding efficiency, the bottleneck of the algorithm is searching the chart for items matching the inference rule. Because of this, the chart needs to be indexed for efficient antecedent lookup. Exactly what indexes are needed depend on the inference rules and will not be discussed here. For a thorough discussion about implementation issues, see Shieber et al. (1995).

3.5.2 Storing and Retrieving Parse Results

The set of syntactic analyses (or *parse trees*) for a given string is called a *parse forest*. The size of this set can be exponential in the length of the string, as mentioned in the introduction section. A classical example is a grammar for PP attachment containing the rules $\mathsf{NP} \rightarrow \mathsf{NP}\ \mathsf{PP}$ and $\mathsf{PP} \rightarrow \mathsf{Prep}\ \mathsf{NP}$. In some pathological cases (i.e., when the grammar is cyclic), there might even be an infinite number of trees. The polynomial parse time complexity stems from the fact that the parse forest can be compactly stored in polynomial space.

A parse forest can be represented as a context-free grammar recognizing the language consisting of only the input string (Bar-Hillel et al. 1964). The forest can then be further investigated to remove useless nodes, increase sharing and reduce space complexity (Billot and Lang 1989).

Retrieving a single parse tree from a (suitably reduced) forest is efficient, but the problem is to decide which tree is the best one. We do not want to examine exponentially many trees, but instead we want a clever procedure for directly finding the best tree. This is the problem of disambiguation, which is discussed in Section 3.8.2 and in Chapter 11.

3.6 LR Parsing

Instead of using the grammar directly, we can precompile it into a form which makes parsing more efficient. One of the most common strategies is LR parsing, which was introduced by Knuth (1965). It is mostly used for deterministic parsing of formal languages such as programming languages, but was extended to nondeterministic languages by Lang (1974) and Tomita (1985, 1987).

One of the main ideas of LR parsing is to handle a number of grammar rules simultaneously by merging common subparts of their right-hand sides, rather than attempting one rule at a time. An LR parser compiles the grammar into a finite automaton, augmented with *reductions* for capturing the nesting of nonterminals in a syntactic structure, making it a kind of pushdown automaton (PDA). The automaton is called an LR automaton, or an LR table.

3.6.1 The LR(0) Table

LR automata can be constructed in several different ways. The simplest construction is the LR(0) table, which uses no *lookahead* when it constructs its states. In practice, most LR algorithms use SLR(1) or LALR(1) tables, which utilise a lookahead of one input symbol. Details of how to construct these automata are, e.g., given by Aho et al. (2006). Our LR(0) construction is similar to the one by Nederhof and Satta (2004b).

States

The states in an LR table are sets of dotted rules $A \rightarrow \alpha \bullet \beta$. The meaning of being in a state is that any of the dotted rules in the state can be the correct one, but we haven't decided yet.

To build an LR(0) table we do the following. First we have to define the function PREDICT-CLOSURE(q), which is the smallest set such that:

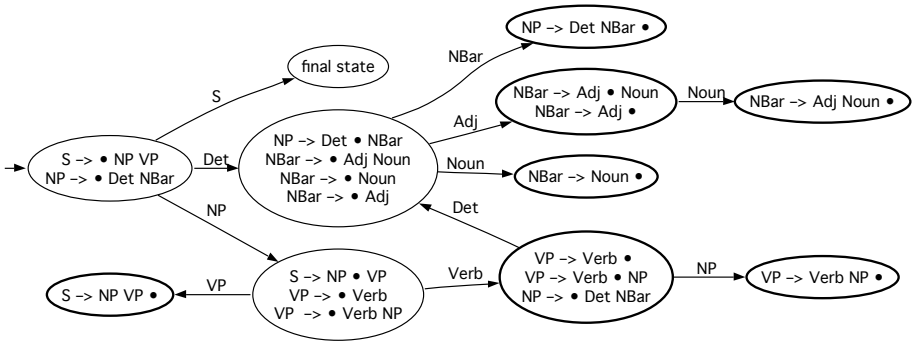

FIGURE 3.6: Example LR(0) table for the grammar in fig 3.1

- $q \subseteq$ PREDICT-CLOSURE(q), and
- if $(A \rightarrow \alpha \bullet B\beta) \in$ PREDICT-CLOSURE(q),
  then $(B \rightarrow \bullet\, \gamma) \in$ PREDICT-CLOSURE(q) for all $B \rightarrow \gamma$

Transitions

Transitions between states are defined by the function GOTO, taking a grammar symbol as argument. The function is defined as:

$$\text{GOTO}(q, X) = \text{PREDICT-CLOSURE}(\{\, A \rightarrow \alpha X \bullet \beta \mid A \rightarrow \alpha \bullet X\beta \in q \,\})$$

The idea is that all dotted rules $A \rightarrow \alpha \bullet X\beta$ will survive to the next state, with the dot moved forward one step. To this the closure of all top-down predictions are added. The initial state q_{init} of the LR table contains predictions of all S rules:

$$q_{\text{init}} = \text{PREDICT-CLOSURE}(\{\, S \rightarrow \bullet\, \gamma \mid S \rightarrow \gamma \,\})$$

We also need a special final state q_{final} which is reachable from the initial state by the dummy transition GOTO(q_{final}, S). Figure 3.6 contains the resulting LR(0) table of the example grammar in figure 3.1. The reducible states, marked with a thicker border in the figure, are the states that contain passive dotted rules, i.e., rules of the form $A \rightarrow \alpha \bullet$. For simplicity we have not included the lexical rules in the LR table.

3.6.2 Deterministic LR Parsing

An LR parser is a shift-reduce parser which uses the transitions of the LR table to push states onto a stack. When the parser is in a reducible state, containing a rule $B \rightarrow \beta \bullet$, it pops $|\beta|$ states off the stack and shifts to a new state by the symbol B.

In our setting we do not use LR states directly, but instead LR states indexed by input position, which we write as $\sigma = q@i$. An LR stack $\omega = \sigma_1 \dots \sigma_n$ is a sequence of indexed LR states. There are three basic operations:[4]

$$\begin{aligned} \textsc{top}(\omega) &= \sigma \quad (\text{where } \omega = \omega'\sigma) \\ \textsc{pop}(\omega) &= \omega' \quad (\text{where } \omega = \omega'\sigma) \\ \textsc{push}(\omega, \sigma) &= \omega\sigma \end{aligned}$$

The parser starts with a stack containing only the initial state in position 0. Then it shifts the next input symbol, and pushes the new state onto the stack. Note the difference with traversing a finite automaton; we do not forget the previous state, but instead push the new state on top. This way we know how to go backwards in the automaton, which we cannot do in a finite automaton.

After shifting, we try to reduce the stack as often as possible, and then we shift the next input token, reduce and continue until the input is exhausted. The parsing has succeeded if we end up with q_{final} as the top state in the stack.

function $\textsc{lr}(w_1 \cdots w_n)$
  $\omega := (q_{\text{init}}@0)$
  for $i := 1$ **to** n **do**
    $\omega := \textsc{reduce}(\textsc{shift}(\omega, w_i@i))$
  if $\textsc{top}(\omega) = q_{\text{final}}@n$ **then** success **else** failure

Shifting a symbol

To shift a symbol X onto the stack ω, we follow the edge labelled X from the current state $\textsc{top}(\omega)$, and push the new state onto the stack:

function $\textsc{shift}(\omega, X@i)$
  $\sigma_{\text{next}} := \textsc{goto}(\textsc{top}(\omega), X)\ @\ i$
  return $\textsc{push}(\omega, \sigma_{\text{next}})$

Reducing the stack

When the top state of the stack contains a rule $A \to B_1 \cdots B_d \bullet$, the nonterminal A has been recognised. The only way to reach that state is from a state containing the rule $A \to B_1 \cdots B_{d-1} \bullet B_d$, which in turn is reached from a state containing $A \to B_1 \cdots B_{d-2} \bullet B_{d-1} B_d$, and so on d steps back in the stack. This state, d steps back, contains the predicted rule $A \to \bullet\, B_1 \cdots B_d$. But there is only one way this rule could have been added to the state – as a prediction from a rule $C \to \alpha \bullet A\beta$. So, if we remove d states from the stack

[4] We will abuse the function $\textsc{top}(\omega)$ by sometimes letting it denote the indexed state $q@i$ and sometimes the LR state q. Furthermore we will write $\textsc{pop}^n$ for n applications of $\textsc{pop}$.

(getting the popped stack ω_{red}), we reach a state which has an A transition.[5] And since we started this paragraph by knowing that A was just recognised, we can shift the popped stack ω_{red}. This whole sequence (popping the stack and then shifting) is called a *reduction*.

However, it is not guaranteed that we can stop here. It is possible that we, after shifting A onto the popped stack, enter a new reducible state, and we can do the whole thing again. This is done until there are no more possible reductions:

function REDUCE(ω)
  $q_{\text{top}}@i := \text{TOP}(\omega)$
  if $(A \to B_1 \cdots B_d \bullet) \in q_{\text{top}}$ **then**
    $\omega_{\text{red}} := \text{POP}^d(\omega)$
    return REDUCE(SHIFT($\omega_{\text{red}}, A@i$))
  else
    return ω

Ungrammaticality and ambiguities

The LR automaton can only handle grammatically correct input. If the input is ungrammatical, we might end up in a state where we can neither reduce nor shift. In this case we have to stop parsing and report an error.

The automaton could also contain nondeterministic choices, even on unambiguous grammars. Thus we might enter a state where it is possible to both shift and reduce at the same time (or reduce in two different ways). In deterministic LR parsing this is called a shift/reduce (or reduce/reduce) conflict, which is considered to be a problem in the grammar. However, since natural language grammars are inherently ambiguous, we have to change the algorithm to handle these cases.

3.6.3 Generalised LR Parsing

To handle nondeterminism, the top-level LR algorithm does not have to be changed much, and only small changes have to be made on the shift and reduce functions. Conceptually, we can think of a "stack" for nondeterministic LR parsing as a set Ω of ordinary stacks ω, which we reduce and shift in parallel. When reducing the stack set, we perform all possible reductions on the elements and take the union of the results. This means that the number of stacks increases, but (hopefully) some of these stacks will be lost when shifting.

[5] Note that if A is the starting symbol S, and $\text{TOP}(\omega_{\text{red}})$ is the initial state $q_{\text{init}}@0$, then there will in fact *not* be any rule $C \to \alpha \bullet S\beta$ in that state. But in this case there is a dummy transition to the final state q_{final}, so we can still shift over S.

The top-level parsing function LR remains as before, with the slight modification that the initial stack set is the singleton set $\{ (q_{\text{init}}) \}$, and that the final stack set should contain some stack whose top state is $q_{\text{final}}@n$.

Using a set of stacks

The basic stack operations POP and PUSH are straightforward to generalise to sets:

$$\text{POP}(\Omega) = \{ \omega \mid \omega\sigma \in \Omega \}$$
$$\text{PUSH}(\Omega, \sigma) = \{ \omega\sigma \mid \omega \in \Omega \}$$

However, there is one problem when trying to obtain the TOP state – since there are several stacks, there can be several different top states. And the TOP operation cannot simply return an unstructured set of states, since we have to know which stacks correspond to which top state. Our solution is to introduce an operation TOP-PARTITION which returns a *partition* of the set of stacks, where all stacks in each part have the same unique top state. The simplest definition is to just make each stack a part of its own, $\text{TOP-PARTITION}(\Omega) = \{ \{\omega\} \mid \omega \in \Omega \}$, but there are several other possible definitions. Now we can define the TOP operation to simply return the top state of any stack in the set, since it will be unique.

$$\text{TOP}(\Omega) = \sigma \quad (\text{where } \omega\sigma \in \Omega)$$

Shifting

The difference compared to deterministic shift is that we loop over the stack partitions, and shift each partition in parallel, returning the union of the results:

function SHIFT($\Omega, X@i$)
$\Omega' := \emptyset$
for all $\omega_{\text{part}} \in$ TOP-PARTITION(Ω) **do**
  $\sigma_{\text{next}} :=$ GOTO(TOP(ω_{part}), X) @ i
  add PUSH($\omega_{\text{part}}, \sigma_{\text{next}}$) to Ω'
return Ω'

Reduction

Nondeterministic reduction also loops over the partition, and does reduction on each part separately, taking the union of the results. Also note that the original set of stacks is included in the final reduction result, since it is always possible that some stack has finished reducing and should shift next.

function REDUCE(Ω)
  for all $\omega_{\text{part}} \in$ TOP-PARTITION(Ω) **do**
    $q_{\text{top}}@i :=$ TOP(ω_{part})

for all $(A \to B_1 \cdots B_d \bullet) \in q_{\text{top}}$ **do**
  $\omega_{\text{red}} := \text{POP}^d(\omega_{\text{part}})$
  add REDUCE(SHIFT($\omega_{\text{red}}, A@i$)) to Ω
return Ω

Grammars with empty productions

The GLR algorithm as it is described in this chapter cannot correctly handle all grammars with ϵ-rules. This is a well-known problem for GLR parsers, and there are two main solutions. One possibility is of course to transform the grammar into ϵ-free form (Hopcroft et al. 2006). Another possibility is to modify the GLR algorithm, possibly together with a modified LR table (Nozohoor-Farshi 1991; Nederhof and Sarbo 1996; Aycock et al. 2001; Aycock and Horspool 2002; Scott and Johnstone 2006; Scott et al. 2007).

3.6.4 Optimised GLR parsing

Each stack which survives in the previous set-based algorithm corresponds to a possible parse tree. But since there can be exponentially many parse trees, this means that the algorithm is exponential in the length of the input.

The problem is the data structure – a set of stacks does not take into account that parallel stacks often have several parts in common. Tomita (1985, 1988) suggested to store a set of stacks as a directed acyclic graph (calling it a *graph-structured stack*), which together with suitable algorithms make GLR parsing polynomial in the length of the input.

The only things we have to do is to reimplement the five operations POP, PUSH, TOP-PARTITION, TOP and ($\cup$); the functions LR, SHIFT and REDUCE all stay the same. We represent a graph-structured stack as a pair $G : T$, where G is a directed graph over indexed states, and T is a subset of the nodes in G that constitute the current stack tops. Assuming that the graph is represented by a set of directed edges $\sigma' \mapsto \sigma$, the operations can be implemented as follows:

$$\begin{aligned}
\text{POP}(G : T) &= G : \{\, \sigma' \mid \sigma \in T,\, \sigma' \mapsto \sigma \in G \,\} \\
\text{PUSH}(G : T, \sigma) &= (G \cup \{\, \sigma' \mapsto \sigma \mid \sigma' \in T \,\}) : \{\sigma\} \\
\text{TOP-PARTITION}(G : T) &= \{\, G : \{\sigma\} \mid \sigma \in T \,\} \\
\text{TOP}(G : \{\sigma\}) &= \sigma \\
(G_1 : T_1) \cup (G_2 : T_2) &= (G_1 \cup G_2) : (T_1 \cup T_2)
\end{aligned}$$

The initial stack in the top-level LR function is still conceptually a singleton set, but will be encoded as a graph-structured stack $\emptyset : \{q_{\text{init}}@0\}$. Note that for the graph-structured version to be correct, we need the LR states to be indexed. Otherwise the graph will merge all nodes having the same LR state, regardless of where in the input it is recognised.

The graph-structured stack operations never remove edges from the graph, only add new edges. This means that it is possible to implement GLR parsing

using a global graph, where the only thing that is passed around is the set T of stack tops.

Tabular GLR parsing

The astute reader might have noticed the similarity between the graph-structured stack and the chart in Section 3.5: The graph (chart) is a global set of edges (items), to which edges (items) are added during parsing, but never removed. It should therefore be possible to reformulate GLR parsing as a tabular algorithm. For this we need two inference rules, corresponding to SHIFT and REDUCE, and one axiom corresponding to the initial stack. This tabular GLR algorithm is described by Nederhof and Satta (2004b).

3.7 Constraint-based Grammars

This section introduces a simple form of constraint-based grammar, or unification grammar, which for more than two decades has constituted a widely adopted class of formalisms in computational linguistics.

3.7.1 Overview

A key characteristic of constraint-based formalisms is the use of feature terms (sets of attribute–value pairs) for the description of linguistic units, rather than atomic categories as in context-free grammars. Feature terms can be nested: their values can be either atomic symbols or feature terms. Furthermore, they are partial (underspecified) in the sense that new information may be added as long as it is compatible with old information. The operation for merging and checking compatibility of feature constraints is usually formalised as *unification*. Some formalisms, such as PATR (Shieber et al. 1983; Shieber 1986) and Regulus (Rayner et al. 2006), are restricted to simple unification (of conjunctive terms), while others such as LFG (Kaplan and Bresnan 1982) and HPSG (Pollard and Sag 1994) allow disjunctive terms, sets, type hierarchies or other extensions. In sum, feature terms have proved to be an extremely versatile and powerful device for linguistic description. One example of this is unbounded dependency, as illustrated by examples (3.1)–(3.3) in Section 3.1, which can be handled entirely within the feature system by the technique of gap threading (Karttunen 1986).

Several constraint-based formalisms are phrase-structure-based in the sense that each rule is factored in a phrase-structure backbone and a set of constraints which specify conditions on the feature terms associated with the rule (for example, PATR, Regulus, CLE, HPSG, LFG and TAG, though the latter uses certain tree-building operations instead of rules). Analogously,

when parsers for constraint-based formalisms are built, the starting-point is often a phrase-structure parser which is augmented to handle feature terms. This is also the approach we shall follow here.

3.7.2 Unification

We make use of a constraint-based formalism with a context-free backbone and restricted to simple unification (of conjunctive terms), thus corresponding to PATR (Shieber et al. 1983; Shieber 1986). A grammar rule in this formalism can be seen as an ordered pair of a production $X_0 \rightarrow X_1 \cdots X_d$ and a set of equational constraints over the feature terms of types $X_0, \ldots, X_d$. A simple example of a rule, encoding agreement between the determiner and the noun in a noun phrase, is the following:

$$\begin{aligned} X_0 \rightarrow\ & X_1\ X_2 \\ & \langle X_0\ \mathit{category} \rangle = \mathsf{NP} \\ & \langle X_1\ \mathit{category} \rangle = \mathsf{Det} \\ & \langle X_2\ \mathit{category} \rangle = \mathsf{Noun} \\ & \langle X_1\ \mathit{agreement} \rangle = \langle X_2\ \mathit{agreement} \rangle \end{aligned}$$

Any such rule description can be represented as a phrase-structure rule where the symbols consist of feature terms. Below is a feature term rule corresponding to the previous rule (where $\boxed{1}$ indicates identity between the associated elements):

$$\left[\, \mathit{category} : \mathsf{NP} \,\right] \rightarrow \begin{bmatrix} \mathit{category} : & \mathsf{Det} \\ \mathit{agreement} : & \boxed{1} \end{bmatrix} \begin{bmatrix} \mathit{category} : & \mathsf{Noun} \\ \mathit{agreement} : & \boxed{1} \end{bmatrix}$$

The basic operation on feature terms is *unification*, which determines if two terms are compatible by merging them to the most general term compatible with both. As an example, the unification $A \sqcup B$ of the terms $A = \left[\, \mathit{agreement} : \left[\, \mathit{number} : \mathsf{plural} \,\right] \,\right]$ and $B = \left[\, \mathit{agreement} : \left[\, \mathit{gender} : \mathsf{neutr} \,\right] \,\right]$ suceeds with the result:

$$A \sqcup B = \left[\mathit{agreement} : \begin{bmatrix} \mathit{gender} : & \mathsf{neutr} \\ \mathit{number} : & \mathsf{plural} \end{bmatrix} \right]$$

However, neither A nor $A \sqcup B$ can be unified with:

$$C = \left[\, \mathit{agreement} : \left[\, \mathit{number} : \mathsf{singular} \,\right] \,\right]$$

since the atomic values plural and singular are distinct. The semantics of feature terms including the unification algorithm is described by Pereira and Shieber (1984), Kasper and Rounds (1986) and Shieber (1992). Unification of feature terms is an extension of Robinson's unification algorithm for first-order terms (Robinson 1965). More advanced grammar formalisms such as HPSG and LFG use further extensions of feature terms, such as type hierarchies and disjunction.

3.7.3 Tabular Parsing with Unification

Basically, the tabular parsers in Section 3.4 as well as the GLR parsers in Section 3.6can be adapted to constraint-based grammar by letting the symbols in the grammar rules be feature terms instead of atomic nonterminal symbols (Shieber 1985b, Tomita 1987, Nakazawa 1991, Samuelsson 1994). For example, an item in tabular parsing then still has the form $[\, i, j : A \rightarrow \alpha \bullet \beta \,]$, where A is a feature term and α, β are sequences of feature terms.

A problem in tabular parsing with constraint-based grammar as opposed to context-free grammar is that the item-redundancy test involves comparing complex feature terms instead of testing for equality between atomic symbols. For this reason, we need to make sure that no previously added item *subsumes* a new item to be added (Shieber 1985a; Pereira and Shieber 1987). Informally, an item e subsumes another item e' if e contains a subset of the information in e'. Since the input positions i, j are always fully instantiated, this amounts to checking if the feature terms in the dotted rule of e subsumes the corresponding feature terms in e'. The rationale for using this test is that we are only interested in adding edges that are less specific than the old ones, since everything we could do with a more specific edge, we can also do with a more general one.

The algorithm for implementing the deduction engine, presented in Section 3.5.1, only needs minor modifications to work on unification-based grammars: (i) instead of checking that the new item e is contained in the chart, we check that there is an item in the chart which subsumes e, and (ii) instead of testing whether two items e_j and e'_j matches, we try to perform the unification $e_j \sqcup e'_j$.

However, subsumption testing is not always sufficient for correct and efficient tabular parsing, since tabular CFG-based parsers are not fully specified in the order in which ambiguities are discovered (Lavie and Rosé 2004). Unification grammars may contain rules that lead to prediction of ever more specific feature terms that do not subsume each other, thereby resulting in infinite sequences of predictions. This kind of problem occurs in natural language grammars when keeping lists of, say, subcategorised constituents or gaps to be found. In logic programming, the *occurs check* is used for circumventing a corresponding circularity problem. In constraint-based grammar, Shieber (1985b) introduced the notion of *restriction* for the same purpose. A restrictor removes those portions of a feature term that could potentially lead to non-termination. This is in general done by replacing those portions with free (newly instantiated) variables, which typically removes some coreference. The purpose of restriction is to ensure that terms to be predicted are only instantiated to a certain depth, such that terms will eventually subsume each other.

3.8 Issues in Parsing

In the light of the previous exposition, this section re-examines the three fundamental challenges of parsing discussed in Section 3.1.

3.8.1 Robustness

Robustness can be seen as the ability to deal with input that somehow does not conform to what is normally expected (Menzel 1995). In grammar-driven parsing, it is natural to take "expected" input to correspond to those strings that are in the formal language $L(G)$ generated by the grammar G. However, as discussed in Section 3.1, the parser will always be exposed to some amount of input that is not in $L(G)$. One source of this problem is *undergeneration*, which is caused by lack of coverage of G relative to the natural language L. Another problem is that the input may contain errors; in other words, that it may be *ill-formed* (though the distinction between well-formed and ill-formed input is by no means clear-cut). But regardless of why the input is not in $L(G)$, it is usually desirable to try to recover as much meaningful information from it as possible, rather than returning no result at all. This is the problem of *robustness*, whose basic notion is to always return *some* analysis of the input. In a stronger sense, robustness means that small deviations from the expected input will only cause small impairments of the parse result, whereas large deviations may cause large impairments. Hence, robustness in this stronger sense amounts to *graceful degradation*.

Clearly, robustness requires methods that sacrifice something from the traditional ideal of recovering complete and exact parses using a linguistically motivated grammar. To avoid the situation where the parser can only stop and report failure in analysing the input, one option is to *relax* some of the grammatical constraints in such a way that a (potentially) ungrammatical sentence obtains a complete analysis (Jensen and Heidorn 1983; Mellish 1989). Put differently, by relaxing some constraints, a certain amount of *overgeneration* is achieved relative to the original grammar, and this is then hopefully sufficient to account for the input. The key problem of this approach is that, as the number of errors grows, the number of relaxation alternatives that are compatible with analyses of the whole input may explode, and that the search for a best solution is therefore very difficult to control.

One can then instead focus on the design of the *grammar*, making it less rich in the hope that this will allow for processing that is less brittle. The amount of information contained in the structural representations yielded by the parser is usually referred to as a distinction between *deep parsing* and *shallow parsing* (somewhat misleadingly, as this distinction does not necessarily refer to different parsing methods per se, but rather to the syntactic representations used). Deep parsing systems typically capture long-distance

dependencies or predicate–argument relations directly, as in LFG, HPSG or CCG (compare Section 3.2.4). In contrast, shallow parsing makes use of more skeletal representations. An example of this is Constraint Grammar (Karlsson et al. 1995). This works by first assigning all possible part-of-speech and syntactic labels to all words. It then applies pattern-matching rules (constraints) to disambiguate the labels, thereby reducing the number of parses. The result constitutes a dependency structure in the sense that it only provides relations between words, and may be ambiguous in that the identities of dependents are not fully specified.

A distinction which is sometimes used more or less synonymously with deep and shallow parsing is that between *full parsing* and *partial parsing*. Strictly speaking, however, this refers to the degree of completeness of the analysis with respect to a given target representation. Thus, partial parsing is often used to denote an initial, surface-oriented analysis ("almost parsing"), in which certain decisions, such as attachments, are left for subsequent processing. A radical form of partial parsing is *chunk parsing* (Abney 1991, 1997), which amounts to finding boundaries between basic elements, such as non-recursive clauses or low-level phrases, and analysing each of these elements using a finite-state grammar. Higher-level analysis is then left for processing by other means. One of the earliest approaches to partial parsing was *Fidditch* (Hindle 1989, 1994). A key idea of this is to leave constituents whose roles cannot be determined unattached, thereby always providing exactly one analysis for any given sentence. Another approach is *supertagging*, introduced by Bangalore and Joshi (1999) for the LTAG formalism as a means to reduce ambiguity by associating lexical items with rich descriptions (supertags) that impose complex constraints in a local context, but again without itself deriving a syntactic analysis. Supertagging has also been successfully applied within the CCG formalism (Clark and Curran 2004).

A second option is to sacrifice completeness with respect to covering the entire *input*, by parsing only fragments that are well-formed according to the grammar. This is sometimes referred to as *skip parsing*. Partial parsing is a means to achieve this, since leaving a fragment unattached may just as well be seen as a way of skipping that fragment. A particularly important case for skip parsing is noisy input, such as written text containing errors or output from a speech recogniser. (A word error rate around 20–40% is by no means unusual in recognition of spontaneous speech; see Chapter 15.) For parsing of spoken language in conversational systems, it has long been commonplace to use pattern-matching rules that trigger on domain-dependent subsets of the input (Ward 1989; Jackson et al. 1991; Boye and Wirén 2008). Other approaches have attempted to render deep parsing methods robust, usually by trying to connect the maximal subset of the original input that is covered by the grammar. For example, GLR* (Lavie 1996; Lavie and Tomita 1996), an extension of GLR (Section 3.6.3), can parse all subsets of the input that are licensed by the grammar by being able to skip over any words. Since many parsable subsets of the original input must then be analysed, the

amount of ambiguity is greatly exacerbated. To control the search space, GLR* makes use of statistical disambiguation similar to a method proposed by Carroll (1993) and Briscoe and Carroll (1993), where probabilities are associated directly with the actions in the precompiled LR parsing table (a method which in turn is an instance of the conditional history-based models discussed in Chapter 11). Other approaches in which subsets of the input can be parsed are Rosé and Lavie (2001), van Noord et al. (1999) and Kasper et al. (1999).

A third option is to sacrifice the traditional notion of *constructive parsing*, that is, analysing sentences by building syntactic representations imposed by the rules of a grammar. Instead one can use *eliminative parsing*, which works by initially setting up a maximal set of conditions, and then gradually reducing analyses that are illegal according to a given set of constraints, until only legal analyses remain. Thus, parsing is here viewed as a constraint satisfaction problem (or, put differently, as disambiguation), in which the set of constraints guiding the process corresponds to the grammar. Examples of this kind of approach are Constraint Grammar (Karlsson et al. 1995) and the system of Foth and Menzel (2005).

3.8.2 Disambiguation

The dual problem of undergeneration is that the parser produces superfluous analyses, for example, in the form of *massive ambiguity*, as illustrated in Section 3.1. Ultimately, we would like not just *some* analysis (robustness), but rather *exactly one* (disambiguation). Although not all information needed for disambiguation (such as contextual constraints) may be available during parsing, some pruning of the search space is usually possible and desirable. The parser may then pass on the n best analyses, if not a single one, to the next level of processing. A related problem, and yet another source of superfluous analyses, is that the grammar might be incomplete not only in the sense of undergeneration, but also by licensing constructions that do not belong to the natural language L. This problem is known as *overgeneration* or *leakage*, by reference to Sapir's famous statement that "[a]ll grammars leak" (Sapir 1921, p. 39).

A basic observation is that, although a general grammar will allow a large number of analyses of almost any non-trivial sentence, most of these analyses will be extremely implausible in the context of a particular domain. A simple approach which was pursued early on was then to code a new, specialised *semantic grammar* for each domain (Burton 1976; Hendrix et al. 1978).[6] A more advanced alternative is to tune the parser and/or grammar for each new domain. Grishman et al. (1984), Samuelsson and Rayner (1991) and Rayner

[6]Note that this early approach can be seen as a text-oriented and less robust variant of the domain-dependent pattern-matching systems of Ward (1989) and others aimed at spoken language, referred to in Section 3.8.1.

et al. (2000) make use of a method known as *grammar specialisation*, which takes advantage of actual rule usage in a particular domain. This method is based on the observation that, in a given domain, certain groups of grammar rules tend to combine frequently in some ways but not in others. On the basis of a sufficiently large corpus parsed by the original grammar, it is then possible to identify common combinations of rules of a (unification) grammar and to collapse them into single "macro" rules. The result is a specialised grammar which, compared to the original grammar, has a larger number of rules but a simpler structure, reducing ambiguity and allowing very fast processing using an LR parser. Another possibility is to use a hybrid method to rank a set of analyses according to their likelihood in the domain, based on data from supervised training (Rayner et al. 2000; Toutanova et al. 2002). Clearly, disambiguation leads naturally in one way or another to application of statistical inference; for a systematic exposition of this, we refer to Chapter 11.

3.8.3 Efficiency

Theoretical time complexity

The worst-case time complexity for parsing with context-free grammar is cubic, $O(n^3)$, in the length of the input sentence. This can most easily be seen for the algorithm CKY() in Section 3.3. The main part consists of three nested loops, all ranging over $O(n)$ input positions, giving cubic time complexity. This is not changed by the UNARY-CKY() algorithm. However, if we add inner loops for handling long right-hand sides, as discussed in Section 3.3.3, the complexity increases to $O(n^{d+1})$, where d is the length of the longest right-hand side in the grammar.

The time complexities of the tabular algorithms in Section 3.4 are also cubic, since using dotted rules constitutes an implicit transformation of the grammar into binary form. In general, assuming that we have a decent implementation of the deduction engine, the time complexity of a deductive algorithm is the complexity of the most complex inference rule. In our case this is the combine rule (3.10), which contains three variables i, j, k ranging over $O(n)$ input positions.

The worst-case time complexity of the optimised GLR algorithm, as formulated in Section 3.6.4, is $O(n^{d+1})$. This is because reduction pops the stack d times, for a rule with right-hand side length d. By binarizing the stack reductions it is possible to obtain cubic time complexity for GLR parsing (Kipps 1991; Nederhof and Satta 1996; Scott et al. 2007).

If the context-free grammar is lexicalised, as mentioned in Section 3.2.4, the time complexity of parsing becomes $O(n^5)$ rather than cubic. The reason for this is that the cubic parsing complexity also depends on the grammar size, which for a bilexical CFG depends quadratically on the size of the lexicon. And after filtering out the grammar rules that do not have a realization in the input sentence, we obtain a complexity of $O(n^2)$ (for the grammar size)

multiplied by $O(n^3)$ (for context-free parsing). Eisner and Satta (1999) and Eisner (2000) provide an $O(n^4)$ algorithm for bilexical CFG, and an $O(n^3)$ algorithm for a common restricted class of lexicalised grammars.

Valiant (1975) showed that it is possible to transform the CKY algorithm into the problem of boolean matrix multiplication (BMM), for which there are sub-cubic algorithms. Currently, the best BMM algorithm is approximately $O(n^{2.376})$ (Coppersmith and Winograd 1990). However, these sub-cubic algorithms all involve large constants making them inefficient in practice. Furthermore, since BMM can be reduced to context-free parsing (Lee 2002), there is not much hope in finding practical parsing algorithms with sub-cubic time complexity.

As mentioned in Section 3.2.4, mildly context-sensitive grammar formalisms all have polynomial parse time complexity. More specifically, TAG and CCG have $O(n^6)$ time complexity (Vijay-Shanker and Weir 1993), whereas for LCFRS, MCFG and RCG the exponent depends on the complexity of the grammar (Satta 1992).

In general, adding feature terms and unification to a phrase-structure backbone makes the resulting formalism undecidable. In practice, however, conditions are often placed on the phrase-structure backbone and/or possible feature terms to reduce complexity (Kaplan and Bresnan 1982, p. 266; Pereira and Warren 1983b, p. 142), sometimes even to the effect of retaining polynomial parsability (Joshi 1997). For a general exposition of computational complexity in connection with linguistic theories, see Barton et al. (1987).

Practical efficiency

The complexity results above represent theoretical worst cases, which in actual practice may occur only under very special circumstances. Hence, to assess the practical behaviour of parsing algorithms, empirical evaluations are more informative. As an illustration of this, in a comparison of three unification-based parsers using a wide-coverage grammar of English, Carroll (1993) found parsing times for exponential-time algorithms to be approximately quadratic in the length of the input for sentence lengths of 1–30 words.

Early work on empirical parser evaluation, such as Pratt (1975), Slocum (1981), Tomita (1985), Wirén (1987) and Billot and Lang (1989), focused on the behaviour of specific algorithms. However, reliable comparisons require that the same grammars and test data are used across different evaluations. Increasingly, the availability of common infrastructure in the form of grammars, treebanks and test suites has facilitated this, as illustrated by Carroll (1994), van Noord (1997), Oepen et al. (2000), Oepen and Carroll (2002) and Kaplan et al. (2004), among others. A more difficult problem is that reliable comparisons also require that parsing times can be normalised across different implementations and computing platforms. One way of trying to handle this would be to have standard implementations of reference algorithms in all implementation languages of interest, as suggested by Moore (2000).

3.9 Historical Notes and Outlook

With the exception of machine translation, parsing is probably the area with the longest history in natural language processing. Victor Yngve has been credited with describing the first method for parsing, conceived of as one component of a system for machine translation, and proceeding bottom-up (Yngve 1955). Subsequently, top-down algorithms were provided by, among others, Kuno and Oettinger (1962). Another early approach was the Transformation and Discourse Analysis Project (TDAP) of Zellig Harris 1958–59, which in effect used cascades of finite-state automata for parsing (Harris 1962; Joshi and Hopely 1996).

During the next decade, the focus shifted to parsing algorithms for context-free grammar. In 1960, John Cocke invented the core dynamic-programming parser which was independently generalised and formalised by Kasami (1965) and Younger (1967), thus evolving into the CKY algorithm. This allowed for parsing in cubic time with grammars in Chomsky normal form. Although Cocke's original algorithm was never published, it remains a highly significant achievement in the history of parsing (for sources to this, see Hays 1966 and Kay 1999). In 1968, Earley then presented the first algorithm for parsing with general CFG in no worse than cubic time (Earley 1970). In independent work, Kay (1967, 1973, 1986) and Kaplan (1973) generalised Cocke's algorithm into what they coined chart parsing. A key idea of this is to view tabular parsing algorithms as instances of a general algorithm schema, with specific parsing algorithms arising from different instantiations of inference rules and the agenda (see also Thompson 1981, 1983, and Wirén 1987).

However, with the growing dominance of Transformational Grammar, particularly with the *Aspects* ("Standard Theory") model introduced by Chomsky (1965), there was a diminished interest in context-free phrase-structure grammar. On the other hand, Transformational Grammar was not itself amenable to parsing in any straightforward way. The main reason for this was the inherent directionality of the transformational component, in the sense that it maps from deep structure to surface word string.

A solution to this problem was the development of Augmented Transition Networks (ATNs), which started in the late 1960s and which became the dominating framework for natural language processing during the 1970s (Woods et al. 1972; Woods 1970, 1973). Basically, the appeal of the ATN was that it constituted a formalism of the same power as Transformational Grammar, but one whose operational claims could be clearly stated, and which provided an elegant (albeit procedural) way of linking the surface structure encoded by the network path with the deep structure built up in registers.

Beginning around 1975, there was a revival of interest in phrase-structure grammars (Joshi et al. 1975; Joshi 1985), later augmented with complex features whose values were typically matched using unification (Shieber 1986).

One reason for this revival was that some of the earlier arguments against the use of CFG had been refuted, resulting in several systematically restricted formalisms (see Section 3.2.4). Another reason was a movement towards declarative (constraint-based) grammar formalisms which typically used a phrase-structure backbone, and whose parsability and formal properties could be rigorously analysed. This allowed parsing to be formulated in ways that abstracted from implementational detail, as demonstrated most elegantly in the parsing-as-deduction paradigm (Pereira and Warren 1983a).

Another development during this time was the generalization of Knuth's deterministic LR parsing algorithm (Knuth 1965) to handling nondeterminism (ambiguous CFGs), leading to the notion of GLR parsing (Lang 1974; Tomita 1985). Eventually, the relation of this framework to tabular (chart) parsing was also illuminated (Nederhof and Satta 1996, 2004b). Finally, in contrast to the work based on phrase-structure grammar, there was a renewed interest in more restricted and performance-oriented notions of parsing, such as finite-state (Church 1980; Ejerhed 1988) and deterministic parsing (Marcus 1980; Shieber 1983).

In the late 1980s and during the 1990s, two interrelated developments were particularly apparent: on the one hand, an interest in robust parsing, motivated by an increased involvement with unrestricted text and spontaneous speech (see Section 3.8.1), and on the other hand the revival of empiricism, leading to statistics-based methods being applied both on their own and in combination with grammar-driven parsing (see Chapter 11). These developments have continued during the first decade in the new millenium, along with a gradual closing of the divide between grammar-driven and statistical methods (Baldwin et al. 2007).

In sum, grammar-driven parsing is one of the oldest areas within natural language processing, and one whose methods continue to be a key component of much of what is carried out in the field. Grammar-driven approaches are essential when the goal is to achieve the precision and rigour of deep parsing, or when annotated corpora for supervised statistical approaches are unavailable. The latter situation holds both for the majority of the world's languages and frequently when systems are to be engineered for new application domains. But also in shallow and partial parsing, some of the most successful systems in terms of accuracy and efficiency are rule-based. However, the best performing broad-coverage parsers for theoretical frameworks such as CCG, HPSG, LFG, TAG and dependency grammar increasingly use statistical components for preprocessing (for example, tagging) and/or postprocessing (by ranking competing analyses for the purpose of disambiguation). Thus, although grammar-driven approaches remain a basic framework for syntactic parsing, it appears that we can continue to look forward to an increasingly symbiotic relationship between grammar-driven and statistical methods.

Acknowledgements

We want to thank the reviewers, Alon Lavie and Mark-Jan Nederhof, for detailed and constructive comments on an earlier version of this chapter. We also want to thank Joakim Nivre for helpful comments and for discussions about the organization of the two parsing chapters (Chapter 3 and Chapter 11).

Bibliography

Abney, S. (1991). Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-Based Parsing*, pp. 257–278. Kluwer Academic Publishers.

Abney, S. (1997). Part-of-speech tagging and partial parsing. In S. Young and G. Bloothooft (Eds.), *Corpus-based Methods in Language and Speech Processing*, pp. 118–136. Kluwer Academic Publishers.

Aho, A., M. Lam, R. Sethi, and J. Ullman (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Addison-Wesley.

Aycock, J. and N. Horspool (2002). Practical Earley parsing. *The Computer Journal 45*(6), 620–630.

Aycock, J., N. Horspool, J. Janoušek, and B. Melichar (2001). Even faster generalized LR parsing. *Acta Informatica 37*(9), 633–651.

Backus, J. W., F. L. Bauer, J. Green, C. Katz, J. Mccarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger (1963). Revised report on the algorithm language ALGOL 60. *Communications of the ACM 6*(1), 1–17.

Baldwin, T., M. Dras, J. Hockenmaier, T. H. King, and G. van Noord (2007). The impact of deep linguistic processing on parsing technology. In *Proc. IWPT'07, 10th International Conference on Parsing Technologies*, Prague, Czech Republic, pp. 36–38.

Bangalore, S. and A. K. Joshi (1999). Supertagging: An approach to almost parsing. *Computational Linguistics 25*(2), 237–265.

Bar-Hillel, Y., M. Perles, and E. Shamir (1964). On formal properties of simple phrase structure grammars. In Y. Bar-Hillel (Ed.), *Language and Information: Selected Essays on their Theory and Application*, Chapter 9, pp. 116–150. Addison-Wesley.

Barton, G. E., R. C. Berwick, and E. S. Ristad (1987). *Computational Complexity and Natural Language*. MIT Press.

Billot, S. and B. Lang (1989). The structure of shared forests in ambiguous parsing. In *Proc. ACL'89, 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, pp. 143–151.

Boullier, P. (2004). Range concatenation grammars. In H. Bunt, J. Carroll, and G. Satta (Eds.), *New developments in parsing technology*, pp. 269–289. Kluwer Academic Publishers.

Boye, J. and M. Wirén (2008). Robust parsing and spoken negotiative dialogue with databases. *Natural Language Engineering 14*(3), 289–312.

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

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

Burton, R. R. (1976). Semantic grammar: An engineering technique for constructing natural language understanding systems. BBN Report 3453, Bolt, Beranek, and Newman, Inc., Cambridge, Massachusetts.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. New York: Cambridge University Press.

Carroll, J. (1993). *Practical Unification-based Parsing of Natural Language*. Ph. D. thesis, University of Cambridge, UK. Computer Laboratory Technical Report 314.

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proc. ACL'94, 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, New Mexico, pp. 287–294.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory 2*(3), 113–124.

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

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press.

Church, K. W. (1980). On memory limitations in natural language processing. Report MIT/LCS/TM-216, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Church, K. W. and R. Patil (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics 8*(3–4), 139–149.

Clark, S. and J. R. Curran (2004). The importance of supertagging for wide-coverage CCG parsing. In *Proc. COLING'04, 20th International Conference on Computational Linguistics*, Geneva, Switzerland.

Coppersmith, D. and S. Winograd (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9*(3), 251–280.

Daniels, M. W. and D. Meurers (2004). A grammar formalism and parser for linearization-based HPSG. In *Proc. COLING'04, 20th International Conference on Computational Linguistics*, Geneva, Switzerland, pp. 169–175.

de Groote, P. (2001). Towards abstract categorial grammars. In *Proc. ACL'01, 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, France.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM 13*(2), 94–102.

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In H. Bunt and A. Nijholt (Eds.), *New Developments in Natural Language Parsing*. Kluwer Academic Publishers.

Eisner, J. and G. Satta (1999). Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. ACL'99, 37th Annual Meeting of the Association for Computational Linguistics*, pp. 457–464.

Ejerhed, E. (1988). Finding clauses in unrestricted text by finitary and stochastic methods. In *Proc. 2nd Conference on Applied Natural Language Processing*, Austin, Texas, pp. 219–227.

Foth, K. and W. Menzel (2005). Robust parsing with weighted constraints. *Natural Language Engineering 11*(1), 1–25.

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

Grishman, R., N. T. Nhan, E. Marsh, and L. Hirschman (1984). Automated determination of sublanguage syntactic usage. In *Proc. COLING-ACL'84, 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics*, Stanford, California, pp. 96–100.

Grosz, B., K. Sparck Jones, and B. L. Webber (Eds.) (1986). *Readings in Natural Language Processing*. Morgan Kaufmann Publishers.

Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton.

Hays, D. G. (1966). Parsing. In D. G. Hays (Ed.), *Readings in automatic language processing*, pp. 73–82. American Elsevier Publishing Company.

Hendrix, G. G., E. D. Sacerdoti, and D. Sagalowicz (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems 3*(2), 105–147.

Hindle, D. (1989). Acquiring disambiguation rules from text. In *Proc. ACL'89, 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, British Columbia, Canada, pp. 118–125.

Hindle, D. (1994). A parser for text corpora. In A. Zampolli (Ed.), *Computational Approaches to the Lexicon*. Oxford University Press.

Hopcroft, J., R. Motwani, and J. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Addison-Wesley.

Jackson, E., D. Appelt, J. Bear, R. Moore, and A. Podlozny (1991). A template matcher for robust NL interpretation. In *Proc. HLT'91, Workshop on Speech and Natural Language*, Pacific Grove, California, pp. 190–194.

Jensen, K. and G. E. Heidorn (1983). The fitted parse: 100 % parsing capability in a syntactic grammar of English. In *Proc. 1st Conference on Applied Natural Language Processing*, Santa Monica, California, pp. 93–98.

Joshi, A. (1997). Parsing techniques. pp. 351–356.

Joshi, A. K. (1985). How much context-sensitivity is necessary for characterizing structural descriptions – tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky (Eds.), *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, pp. 206–250. Cambridge University Press.

Joshi, A. K. and P. Hopely (1996). A parser from antiquity: an early application of finite state transducers to natural language parsing. *Natural Language Engineering 2*(4), 291–294.

Joshi, A. K., L. S. Levy, and M. Takahashi (1975). Tree adjunct grammars. *Journal of Computer and System Sciences 10*(1), 136–163.

Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages. Vol 3: Beyond Words*, Chapter 2, pp. 69–123. Springer-Verlag.

Kaplan, R. and J. Bresnan (1982). Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations*, pp. 173–281. MIT Press.

Kaplan, R. M. (1973). A general syntactic processor. See Rustin (1973), pp. 193–241.

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

Kaplan, R. M. and A. Zaenen (1995). Long-distance dependencies, constituent structure, and functional uncertainty. In R. M. Kaplan, M. Dalrymple, J. T. Maxwell, and A. Zaenen (Eds.), *Formal issues in Lexical-Functional Grammar*, Chapter 3, pp. 137–165. CSLI Publications.

Karlsson, F., A. Voutilainen, J. Heikkilä, and A. Anttila (Eds.) (1995). *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter.

Karttunen, L. (1986). D-PATR: A development environment for unification-based grammars. In *Proc. COLING'86, 11th International Conference on Computational Linguistics*.

Karttunen, L. and A. M. Zwicky (1985). Introduction. In D. Dowty, L. Karttunen, and A. Zwicky (Eds.), *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, pp. 1–25. Cambridge University Press.

Kasami, T. (1965). An efficient recognition and syntax algorithm for context-free languages. Technical Report AFCLR-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.

Kasper, R. T. and W. C. Rounds (1986). A logical semantics for feature structures. In *Proc. ACL'86, 24th Annual Meeting of the Association for Computational Linguistics*, New York, pp. 257–266.

Kasper, W., B. Kiefer, H. U. Krieger, C. J. Rupp, and K. L. Worm (1999). Charting the depths of robust speech parsing. In *Proc. ACL'99, 37th Annual Meeting of the Association for Computational Linguistics*, College Park, Maryland, pp. 405–412.

Kay, M. (1967). Experiments with a powerful parser. In *Proc. COLING'67, 2nd International Conference on Computational Linguistics [2ème conférence internationale sur le traitement automatique des langues]*, Grenoble, France.

Kay, M. (1973). The MIND system. See Rustin (1973), pp. 155–188.

Kay, M. (1986). Algorithm schemata and data structures in syntactic processing. See Grosz, Sparck Jones, and Webber (1986), pp. 35–70. Originally published as Report CSL-80-12, Xerox PARC, Palo Alto, California, 1980.

Kay, M. (1989). Head-driven parsing. In *Proc. IWPT'89, 1st International Workshop on Parsing Technologies*, Pittsburgh, Pennsylvania.

Kay, M. (1999). Chart translation. In *Proc. MT Summit VII*, Singapore, pp. 9–14.

Kipps, J. R. (1991). GLR parsing in time $O(n^3)$. In M. Tomita (Ed.), *Generalized LR Parsing*, Chapter 4, pp. 43–59. Kluwer Academic Publishers.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control 8*, 607–639.

Kuno, S. and A. G. Oettinger (1962). Multiple-path syntactic analyzer. In *Proc. IFIP Congress*, pp. 306–312.

Lang, B. (1974). Deterministic techniques for efficient non-deterministic parsers. In J. Loeckx (Ed.), *Proc. 2nd Colloquium on Automata, Languages and Programming*, Volume 14 of *LNCS*, pp. 255–269. Springer-Verlag.

Lavie, A. (1996). GLR*: A robust parser for spontaneously spoken language. In *Proc. ESSLLI'96 Workshop on Robust Parsing*, Prague.

Lavie, A. and C. P. Rosé (2004). Optimal ambiguity packing in context-free parsers with interleaved unification. In H. Bunt, J. Carroll, and G. Satta (Eds.), *New Developments in Parsing Technology*, pp. 307–321. Kluwer Academic Publishers.

Lavie, A. and M. Tomita (1996). GLR* – an efficient noise-skipping parsing algorithm for context-free grammars. In H. Bunt and M. Tomita (Eds.), *Recent Advances in Parsing Technology*, Chapter 10, pp. 183–200. Kluwer Academic Publishers.

Lee, L. (2002). Fast context-free grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM 49*(1), 1–15.

Leiss, H. (1990). On Kilbury's modification of Earley's algorithm. *ACM Trans. Program. Lang. Syst. 12*(4), 610–640.

Ljunglöf, P. (2004). *Expressivity and Complexity of the Grammatical Framework*. Ph. D. thesis, University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden.

Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. MIT Press.

Mellish, C. S. (1989). Some chart-based techniques for parsing ill-formed input. In *Proc. ACL'89, 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, British Columbia, Canada, pp. 102–109.

Menzel, W. (1995). Robust processing of natural language. In *In Proc. 19th Annual German Conference on Artificial Intelligence*.

Moore, R. C. (2000). Time as a measure of parsing efficiency. In *Proc. COLING'00 Workshop on Efficiency in Large-Scale Parsing Systems*, Luxemburg.

Moore, R. C. (2004). Improved left-corner chart parsing for large context-free grammars. In H. Bunt, J. Carroll, and G. Satta (Eds.), *New Developments in Parsing Technology*, pp. 185–201. Kluwer Academic Publishers.

Nakazawa, T. (1991). An extended LR parsing algorithm for grammars using feature-based syntactic categories. In *Proc. EACL'91, 5th Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, Germany.

Nederhof, M.-J. and J. Sarbo (1996). Increasing the applicability of LR parsing. In H. Bunt and M. Tomita (Eds.), *Recent Advances in Parsing Technology*, pp. 35–58. Kluwer Academic Publishers.

Nederhof, M.-J. and G. Satta (1996). Efficient tabular LR parsing. In *Proc. ACL'96, 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, California, pp. 239–246.

Nederhof, M.-J. and G. Satta (2004a). IDL-expressions: A formalism for representing and parsing finite languages in natural language processing. *Journal of Artificial Intelligence Research 21*, 287–317.

Nederhof, M.-J. and G. Satta (2004b). Tabular parsing. In C. Martin-Vide, V. Mitrana, and G. Paun (Eds.), *Formal Languages and Applications*, Volume 148 of *Studies in Fuzziness and Soft Computing*, pp. 529–549. Springer-Verlag.

Nivre, J. (2006). *Inductive Dependency Parsing*. Springer-Verlag.

Nozohoor-Farshi, R. (1991). GLR parsing for ϵ-grammars. In M. Tomita (Ed.), *Generalized LR Parsing*. Kluwer Academic Publishers.

Oepen, S. and J. Carroll (2002). Efficient parsing for unification-based grammars. In H. U. D. Flickinger, S. Oepen and J.-I. Tsujii (Eds.), *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Processing*, pp. 195–225. CSLI Publications.

Oepen, S., D. Flickinger, H. Uszkoreit, and J.-I. Tsujii (2000). Introduction to this special issue. *Natural Language Engineering 6*(1), 1–14.

Pereira, F. C. N. and S. M. Shieber (1984). The semantics of grammar formalisms seen as computer languages. In *Proc. COLING'84, 10th International Conference on Computational Linguistics*, Stanford, California, pp. 123–129.

Pereira, F. C. N. and S. M. Shieber (1987). *Prolog and Natural-Language Analysis*, Volume 4 of *CSLI lecture notes*. CSLI Publications. Reissued in 2002 by Microtome Publishing.

Pereira, F. C. N. and D. H. D. Warren (1983a). Parsing as deduction. In *Proc. ACL'83, 21st Annual Meeting of the Association for Computational Linguistics*, pp. 137–144.

Pereira, F. C. N. and D. H. D. Warren (1983b). Parsing as deduction. In *Proc. ACL'83, 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, Massachusetts, pp. 137–144.

Pollard, C. and I. Sag (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Pratt, V. R. (1975). LINGOL – a progress report. In *Proc. 4th International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, pp. 422–428.

Ranta, A. (1994). *Type-Theoretical Grammar*. Oxford University Press.

Ranta, A. (2004). Grammatical Framework, a type-theoretical grammar formalism. *Journal of Functional Programming 14*(2), 145–189.

Rayner, M., D. Carter, P. Bouillon, V. Digalakis, and M. Wirén (2000). *The Spoken Language Translator*. Cambridge University Press.

Rayner, M., B. A. Hockey, and P. Bouillon (2006). *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Publications.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM 12*(1), 23–49.

Rosé, C. P. and A. Lavie (2001). Balancing robustness and efficiency in unification-augmented context-free parsers for large practical applications. In J.-C. Junqua and G. van Noord (Eds.), *Robustness in Language and Speech Technology*. Kluwer Academic Publishers.

Rustin, R. (Ed.) (1973). *Natural Language Processing*. Algorithmics Press.

Samuelsson, C. (1994). Notes on LR parser design. In *Proc. 15th International Conference on Computational Linguistics*, Kyoto, Japan, pp. 386–390.

Samuelsson, C. and M. Rayner (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proc. 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp. 609–615.

Sapir, E. (1921). *Language: An introduction to the study of speech*. Harcourt Brace & Co.

Satta, G. (1992). Recognition of linear context-free rewriting systems. In *Proc. ACL'92, 30th Annual Meeting of the Association for Computational Linguistics*, Newark, Delaware, pp. 89–95.

Scott, E. and A. Johnstone (2006). Right nulled GLR parsers. *ACM Trans. Program. Lang. Syst. 28*(4), 577–618.

Scott, E., A. Johnstone, and R. Economopoulos (2007). BRNGLR: a cubic Tomita-style GLR parsing algorithm. *Acta Informatica 44*(6), 427–461.

Seki, H., T. Matsumara, M. Fujii, and T. Kasami (1991). On multiple context-free grammars. *Theoretical Computer Science 88*, 191–229.

Shieber, S. M. (1983). Sentence disambiguation by a shift-reduce parsing technique. In *Proc. ACL'83, 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, Massachusetts, pp. 113–118.

Shieber, S. M. (1985a). Evidence against the context-freeness of natural language. *Linguistics and Philosophy 8*(3), 333–343.

Shieber, S. M. (1985b). Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proc. ACL'85, 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, Illinois, pp. 145–152.

Shieber, S. M. (1986). *An Introduction to Unification-based Approaches to Grammar*. Number 4 in CSLI Lecture Notes. University of Chicago Press.

Shieber, S. M. (1992). *Constraint-Based Grammar Formalisms*. MIT Press.

Shieber, S. M., Y. Schabes, and F. C. N. Pereira (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming 24*(1–2), 3–36.

Shieber, S. M., H. Uszkoreit, F. C. N. Pereira, J. J. Robinson, and M. Tyson (1983). The formalism and implementation of PATR-II. In B. J. Grosz and M. E. Stickel (Eds.), *Research on Interactive Acquisition and Use of Knowledge*, Final Report, SRI project number 1894, pp. 39–79. SRI International.

Sikkel, K. (1998). Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science 199*, 87–103.

Sikkel, K. and A. Nijholt (1997). Parsing of context-free languages. In G. Rozenberg and A. Salomaa (Eds.), *The Handbook of Formal Languages*, Volume II, pp. 61–100. Springer-Verlag.

Slocum, J. (1981). A practical comparison of parsing strategies. In *Proc. ACL'81, 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, pp. 1–6.

Steedman, M. (1985). Dependency and coordination in the grammar of Dutch and English. *Language 61*, 523–568.

Steedman, M. (1986). Combinators and grammars. In R. Oehrle, E. Bach, and D. Wheeler (Eds.), *Categorial Grammars and Natural Language Structures*, pp. 417–442. Foris Publications.

Steedman, M. J. (1983). Natural and unnatural language processing. In K. Sparck Jones and Y. Wilks (Eds.), *Automatic Natural Language Parsing*, pp. 132–140. Ellis Horwood.

Tesnière, L. (1959). *Éléments de Syntaxe Structurale*. Libraire C. Klincksieck.

Thompson, H. S. (1981). Chart parsing and rule schemata in GPSG. In *Proc. ACL'81, 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, pp. 167–172.

Thompson, H. S. (1983). MCHART: A flexible, modular chart parsing system. In *Proc. Third National Conference on Artificial Intelligence*, Washington, DC, pp. 408–410.

Tomita, M. (1985). *Efficient Parsing for Natural Language*. Kluwer Academic Publishers.

Tomita, M. (1987). An efficient augmented context-free parsing algorithm. *Computational Linguistics 13*(1–2), 31–46.

Tomita, M. (1988). Graph-structured stack and natural language parsing. In *Proc. ACL'88, 26th Annual Meeting of the Association for Computational Linguistics*.

Toutanova, K., C. D. Manning, S. M. Shieber, D. Flickinger, and S. Oepen (2002). Parse disambiguation for a rich HPSG grammar. In *Proc. 1st Workshop on Treebanks and Linguistic Theories*, Sozopol, Bulgaria, pp. 253–263.

Valiant, L. (1975). General context-free recognition in less than cubic time. *Journal of Computer and Systems Sciences 10*(2), 308–315.

van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics 23*(3), 425–456.

van Noord, G., G. Bouma, R. Koeling, and M.-J. Nederhof (1999). Robust grammatical analysis for spoken dialogue systems. *Natural Language Engineering 5*(1), 45–93.

Vijay-Shanker, K. and D. Weir (1993). Parsing some constrained grammar formalisms. *Computational Linguistics 19*(4), 591–636.

Vijay-Shanker, K. and D. Weir (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory 27*(6), 511–546.

Vijay-Shanker, K., D. Weir, and A. K. Joshi (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. ACL'87, 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, California.

Ward, W. (1989). Understanding spontaneous speech. In *Proc. HLT '89, Workshop on Speech and Natural Language*, Philadelphia, Pennsylvania, pp. 137–141.

Wirén, M. (1987). A comparison of rule-invocation strategies in context-free chart parsing. In *Proc. EACL'87, 3rd Conference of the European Chapter of the Association for Computational Linguistics*.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM 13*(10), 591–606. Also in Grosz et al. (1986, pp. 71–87).

Woods, W. A. (1973). An experimental parsing system for transition network grammars. See Rustin (1973), pp. 111–154.

Woods, W. A., R. M. Kaplan, and B. Nash-Webber (1972). The lunar sciences natural language information system: final report. BBN Report 2378, Bolt, Beranek, and Newman, Inc., Cambridge, Massachusetts.

Yngve, V. H. (1955). Syntax and the problem of multiple meaning. In W. N. Locke and A. D. Booth (Eds.), *Machine Translation of Languages*, pp. 208–226. MIT Press.

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