# Chapter 13
# Parsing with Context-Free Grammars

> *There are and can exist but two ways of investigating and discovering truth. The one*
> *hurries on rapidly from the senses and particulars to the most general axioms, and from*
> *them…derives and discovers the intermediate axioms. The other constructs its axioms*
> *from the senses and particulars, by ascending continually and gradually, till it finally*
> *arrives at the most general axioms.*
>
> Francis Bacon, *Novum Organum* Book I.19 (1620)

We defined parsing in Ch. 3 as a combination of recognizing an input string and assigning a structure to it. Syntactic parsing, then, is the task of recognizing a sentence and assigning a syntactic structure to it. This chapter focuses on the kind of structures assigned by context-free grammars of the kind described in Ch. 12. However, since they are based on a purely declarative formalism, context-free grammars don't specify *how* the parse tree for a given sentence should be computed. We'll therefore need to specify algorithms that employ these grammars to produce trees. This chapter presents three of the most widely used parsing algorithms for automatically assigning a complete context-free (phrase structure) tree to an input sentence.

These kinds of parse trees are directly useful in applications such as **grammar checking** in word-processing systems: a sentence which cannot be parsed may have grammatical errors (or at least be hard to read). More typically, however, parse trees serve as an important intermediate stage of representation for **semantic analysis** (as we will see in Ch. 18), and thus play an important role in applications like **question answering** and **information extraction**. For example, to answer the question

> *What books were written by British women authors before 1800?*

we'll need to know that the subject of the sentence was *what books* and that the by-adjunct was *British women authors* to help us figure out that the user wants a list of books (and not a list of authors).

Before presenting any parsing algorithms, we begin by describing some of the factors that motivate the standard algorithms. First, we revisit the **search metaphor** for parsing and recognition, which we introduced for finite-state automata in Ch. 2, and talk about the **top-down** and **bottom-up** search strategies. We then discuss how the ambiguity problem rears its head again in syntactic processing, and how it ultimately makes simplistic approaches based on backtracking infeasible.

The sections that follow then present the Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967), the Earley algorithm (Earley, 1970), and the chart parsing approach (Kay, 1982; Kaplan, 1973). These approaches all combine insights from bottom-up and top-down parsing with dynamic programming to efficiently handle

| Grammar | Lexicon |
|---|---|
| $S \rightarrow NP\ VP$ | $Det \rightarrow that \mid this \mid a$ |
| $S \rightarrow Aux\ NP\ VP$ | $Noun \rightarrow book \mid flight \mid meal \mid money$ |
| $S \rightarrow VP$ | $Verb \rightarrow book \mid include \mid prefer$ |
| $NP \rightarrow Pronoun$ | $Pronoun \rightarrow I \mid she \mid me$ |
| $NP \rightarrow Proper\text{-}Noun$ | $Proper\text{-}Noun \rightarrow Houston \mid NWA$ |
| $NP \rightarrow Det\ Nominal$ | $Aux \rightarrow does$ |
| $Nominal \rightarrow Noun$ | $Preposition \rightarrow from \mid to \mid on \mid near \mid through$ |
| $Nominal \rightarrow Nominal\ Noun$ | |
| $Nominal \rightarrow Nominal\ PP$ | |
| $VP \rightarrow Verb$ | |
| $VP \rightarrow Verb\ NP$ | |
| $VP \rightarrow Verb\ NP\ PP$ | |
| $VP \rightarrow Verb\ PP$ | |
| $VP \rightarrow VP\ PP$ | |
| $PP \rightarrow Preposition\ NP$ | |

**Figure 13.1**    The $\mathscr{L}_1$ miniature English grammar and lexicon.

complex inputs. Recall that we've already seen several applications of dynamic pro-
gramming algorithms in earlier chapters — Minimum-Edit-Distance, Viterbi, Forward.
Finally, we discuss **partial parsing methods**, for use in situations where a superficial
syntactic analysis of an input may be sufficient.

# 13.1    Parsing as Search

Chapters 2 and 3 showed that finding the right path through a finite-state automaton,
or finding the right transduction for an input, can be viewed as a search problem. For
finite-state automata, the search is through the space of all possible paths through a
machine. In syntactic parsing, the parser can be viewed as searching through the space
of possible parse trees to find the correct parse tree for a given sentence. Just as the
search space of possible paths was defined by the structure of an automaton, so the
search space of possible parse trees is defined by a grammar. Consider the following
ATIS sentence:

(13.1)  Book that flight.

Fig. 13.1 introduces the $\mathscr{L}_1$ grammar, which consists of the $\mathscr{L}_0$ grammar from the
last chapter with a few additional rules. Given this grammar, the correct parse tree for
this example would be the one shown in Fig. 13.2.

How can we use $\mathscr{L}_1$ to assign the parse tree in Fig. 13.2 to this example? The
goal of a parsing search is to find all the trees whose root is the start symbol $S$ and
which cover exactly the words in the input. Regardless of the search algorithm we
choose, there are two kinds of constraints that should help guide the search. One set
of constraints comes from the data, that is, the input sentence itself. Whatever else
is true of the final parse tree, we know that there must be three leaves, and that they
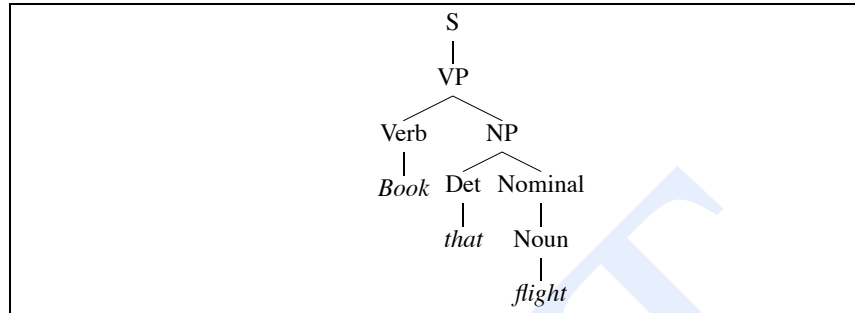
**Figure 13.2**    The parse tree for the sentence *Book that flight* according to grammar $\mathscr{L}_1$.

must be the words *book*, *that*, and *flight*. The second kind of constraint comes from the grammar. We know that whatever else is true of the final parse tree, it must have one root, which must be the start symbol *S*.

These two constraints, invoked by Bacon at the start of this chapter, give rise to the two search strategies underlying most parsers: **top-down** or **goal-directed search**, and **bottom-up** or **data-directed search**. These constraints are more than just search strategies. They reflect two important insights in the western philosophical tradition: the **rationalist** tradition, which emphasizes the use of prior knowledge, and the **empiricist** tradition, which emphasizes the data in front of us.

*Rationalist*
*Empiricist*

### 13.1.1    Top-Down Parsing

*Top-down*

A **top-down** parser searches for a parse tree by trying to build from the root node *S* down to the leaves. Let's consider the search space that a top-down parser explores, assuming for the moment that it builds all possible trees in parallel. The algorithm starts by assuming that the input can be derived by the designated start symbol *S*. The next step is to find the tops of all trees which can start with *S*, by looking for all the grammar rules with *S* on the left-hand side. In the grammar in Fig. 13.1, there are three rules that expand *S*, so the second **ply**, or level, of the search space in Fig. 13.1.1 has three partial trees.

*Ply*

We next expand the constituents in these three new trees, just as we originally expanded *S*. The first tree tells us to expect an *NP* followed by a *VP*, the second expects an *Aux* followed by an *NP* and a *VP*, and the third a *VP* by itself. To fit the search space on the page, we have shown in the third ply of Fig. 13.1.1 only a subset of the trees that result from the expansion of the left-most leaves of each tree. At each ply of the search space we use the right-hand sides of the rules to provide new sets of expectations for the parser, which are then used to recursively generate the rest of the trees. Trees are grown downward until they eventually reach the part-of-speech categories at the bottom of the tree. At this point, trees whose leaves fail to match all the words in the input can be rejected, leaving behind those trees that represent successful parses. In Fig. 13.1.1, only the fifth parse tree in the third ply (the one which has expanded the rule *VP → Verb NP*) will eventually match the input sentence *Book that flight*.
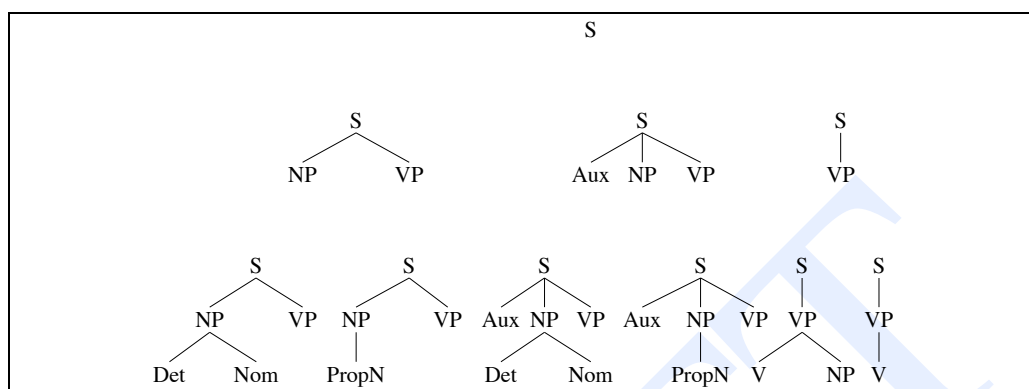
**Figure 13.3**   An expanding top-down search space. Each ply is created by taking each tree from the previous ply, replacing the leftmost non-terminal with each of its possible expansions, and collecting each of these trees into a new ply.

### 13.1.2   Bottom-Up Parsing

*Bottom-up*

**Bottom-up** parsing is the earliest known parsing algorithm (it was first suggested by Yngve (1955)), and is used in the shift-reduce parsers common for computer languages (Aho and Ullman, 1972). In bottom-up parsing, the parser starts with the words of the input, and tries to build trees from the words up, again by applying rules from the grammar one at a time. The parse is successful if the parser succeeds in building a tree rooted in the start symbol *S* that covers all of the input. Fig. 13.1.2 shows the bottom-up search space, beginning with the sentence *Book that flight*. The parser begins by looking up each input word in the lexicon and building three partial trees with the part-of-speech for each word. But the word *book* is ambiguous; it can be a noun or a verb. Thus the parser must consider two possible sets of trees. The first two plies in Fig. 13.1.2 show this initial bifurcation of the search space.

Each of the trees in the second ply is then expanded. In the parse on the left (the one in which *book* is incorrectly considered a noun), the *Nominal → Noun* rule is applied to both of the nouns (*book* and *flight*). This same rule is also applied to the sole noun (*flight*) on the right, producing the trees on the third ply.

In general, the parser extends one ply to the next by looking for places in the parse-in-progress where the right-hand side of some rule might fit. This contrasts with the earlier top-down parser, which expanded trees by applying rules when their left-hand side matched an unexpanded non-terminal.

Thus in the fourth ply, in the first and third parse, the sequence *Det Nominal* is recognized as the right-hand side of the *NP → Det Nominal* rule.

In the fifth ply, the interpretation of *book* as a noun has been pruned from the search space. This is because this parse cannot be continued: there is no rule in the grammar with the right-hand side *Nominal NP*. The final ply of the search space (not shown in Fig. 13.1.2) contains the correct parse (see Fig. 13.2).
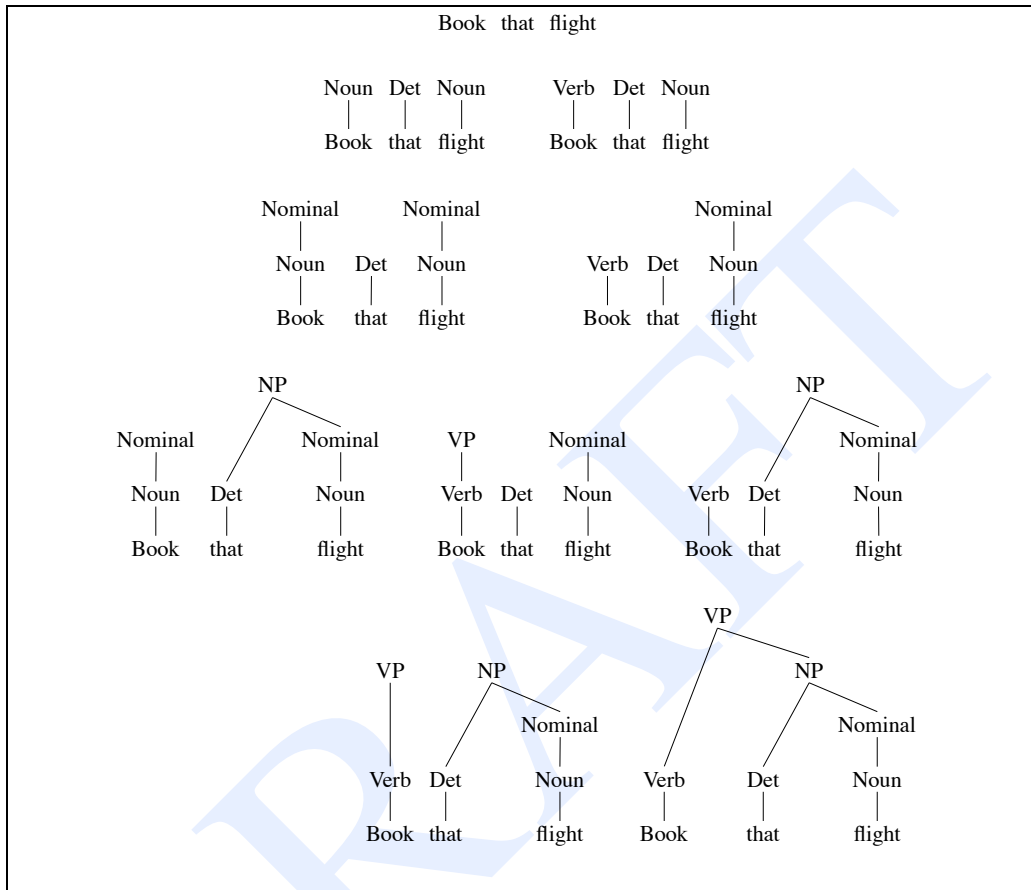
**Figure 13.4**    An expanding bottom-up search space for the sentence *Book that flight*. This figure does not show the final phase of the search with the correct parse tree (see Fig. 13.2). Make sure you understand how that final parse tree follows from the search space in this figure.

### 13.1.3    Comparing Top-Down and Bottom-Up Parsing

Each of these two architectures has its own advantages and disadvantages. The top-down strategy never wastes time exploring trees that cannot result in an *S*, since it begins by generating just those trees. This means it also never explores subtrees that cannot find a place in some *S*-rooted tree. In the bottom-up strategy, by contrast, trees that have no hope of leading to an *S*, or fitting in with any of their neighbors, are generated with wild abandon.

The top-down approach has its own inefficiencies. While it does not waste time with trees that do not lead to an *S*, it does spend considerable effort on *S* trees that are not consistent with the input. Note that the first four of the six trees in the third ply in Fig. 13.1.1 all have left branches that cannot match the word *book*. None of these trees
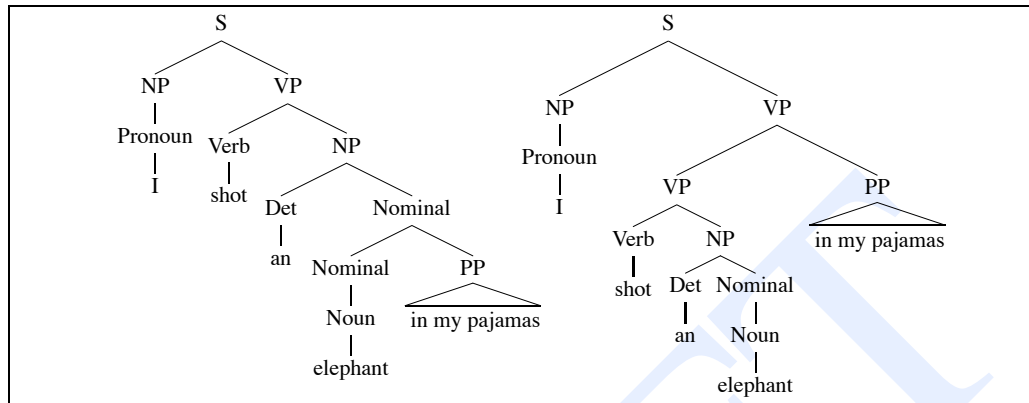
**Figure 13.5**    Two parse trees for an ambiguous sentence. Parse (a) corresponds to the humorous reading in which the elephant is in the pajamas, parse (b) to the reading in which Captain Spaulding did the shooting in his pajamas.

could possibly be used in parsing this sentence. This weakness in top-down parsers arises from the fact that they generate trees before ever examining the input. Bottom-up parsers, on the other hand, never suggest trees that are not at least locally grounded in the input.

## 13.2    Ambiguity

> *One morning I shot an elephant in my pajamas. How he got into my paja-mas I don't know.*

Groucho Marx, *Animal Crackers*, 1930

Ambiguity is perhaps the most serious problem faced by parsers. Ch. 5 introduced the notions of **part-of-speech ambiguity** and **part-of-speech disambiguation**. In this section we introduce a new kind of ambiguity, which arises in the syntactic structures used in parsing, called **structural ambiguity**. Structural ambiguity occurs when the grammar assigns more than one possible parse to a sentence. Groucho Marx's well-known line as Captain Spaulding is ambiguous because the phrase *in my pajamas* can be part of the *NP* headed by *elephant* or of the verb-phrase headed by *shot*.

*Structural ambiguity*

Structural ambiguity, appropriately enough, comes in many forms. Two common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**.

*Attachment ambiguity*

A sentence has an **attachment ambiguity** if a particular constituent can be attached to the parse tree at more than one place. The Groucho Marx sentence above is an example of *PP*-attachment ambiguity. Various kinds of adverbial phrases are also subject to this kind of ambiguity. For example in the following example the gerundive-*VP flying to Paris* can be part of a gerundive sentence whose subject is *the Eiffel Tower* or it can be an adjunct modifying the *VP* headed by *saw*:

(13.2)  We saw the Eiffel Tower flying to Paris.

*Coordination ambiguity*

In **coordination ambiguity** there are different sets of phrases that can be conjoined