3.4 A Bottom-Up Chart Parser

The main difference between top-down and bottom-up parsers is the way the grammar rules are used. For example, consider the rule

NP $\rightarrow$ ART ADJ N

In a top-down system you use the rule to find an NP by looking for the sequence ART ADJ N. In a bottom-up parser you use the rule to take a sequence ART ADJ N that you have found and identify it as an NP. The basic operation in bottom-up parsing then is to take a sequence of symbols and match it to the right-hand side of the rules. You could build a bottom-up parser simply by formulating this matching process as a search process. The state would simply consist of a symbol list, starting with the words in the sentence. Successor states could be generated by exploring all possible ways to

- rewrite a word by its possible lexical categories
- replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol

1. S → NP VP
2. NP → ART ADJ N
3. NP → ART N
4. NP → ADJ N
5. VP → AUX VP
6. VP → V NP

Grammar 3.8 A simple context-free grammar

Unfortunately, such a simple implementation would be prohibitively expensive, as the parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily. To avoid this problem, a data structure called a **chart** is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Matches are always considered from the point of view of one constituent, called the **key**. To find rules that match a string involving the key, look for rules that start with the key, or for rules that have already been started by earlier keys and require the present key either to complete the rule or to extend the rule. For instance, consider Grammar 3.8.

Assume you are parsing a sentence that starts with an ART. With this ART as the key, rules 2 and 3 are matched because they start with ART. To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART. You denote this fact by writing the rule with a dot (∘), indicating what has been seen so far. Thus you record

2′. NP → ART ∘ ADJ N
3′. NP → ART ∘ N

If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2′ may be extended to give

2″. NP → ART ADJ ∘ N

The chart maintains the record of all the constituents derived from the sentence so far in the parse. It also maintains the record of rules that have matched partially but are not complete. These are called the **active arcs**. For example, after seeing an initial ART followed by an ADJ in the preceding example, you would have the chart shown in Figure 3.9. You should interpret this figure as follows. There are two completed constituents on the chart: ART1 from position 1 to 2 and ADJ1 from position 2 to 3. There are four active arcs indicating possible constituents. These are indicated by the arrows and are interpreted as follows (from top to bottom). There is a potential NP starting at position 1, which needs an ADJ starting at position 2. There is another potential NP starting at position 1, which needs an N starting at position 2. There is a potential NP

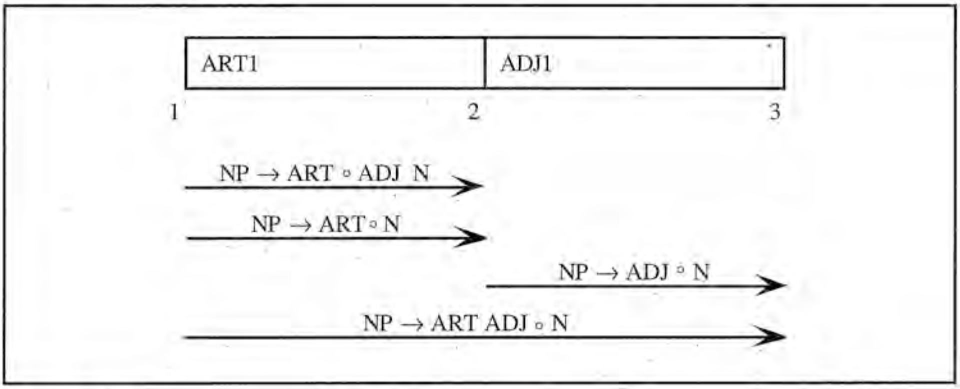


Figure 3.9 The chart after seeing an ADJ in position 2

```
To add a constituent C from position $p_1$ to $p_2$:
1.  Insert C into the chart from position $p_1$ to $p_2$.
2.  For any active arc of the form $X \rightarrow X_1 \ldots \circ C \ldots X_n$ from position $p_0$ to $p_1$,
    add a new active arc $X \rightarrow X_1 \ldots C \circ \ldots X_n$ from position $p_0$ to $p_2$.
3.  For any active arc of the form $X \rightarrow X_1 \ldots X_n \circ C$ from position $p_0$ to $p_1$,
    then add a new constituent of type X from $p_0$ to $p_2$ to the agenda.
```

Figure 3.10 The arc extension algorithm

starting at position 2 with an ADJ, which needs an N starting at position 3. Finally, there is a potential NP starting at position 1 with an ART and then an ADJ, which needs an N starting at position 3.

The basic operation of a chart-based parser involves combining an active arc with a completed constituent. The result is either a new completed constituent or a new active arc that is an extension of the original active arc. New completed constituents are maintained on a list called the **agenda** until they themselves are added to the chart. This process is defined more precisely by the arc extension algorithm shown in Figure 3.10. Given this algorithm, the bottom-up chart parsing algorithm is specified in Figure 3.11.

As with the top-down parsers, you may use a depth-first or breadth-first search strategy, depending on whether the agenda is implemented as a stack or a queue. Also, for a full breadth-first strategy, you would need to read in the entire input and add the interpretations of the words onto the agenda before starting the algorithm. Let us assume a depth-first search strategy for the following example.

Consider using the algorithm on the sentence *The large can can hold the water* using Grammar 3.8 with the following lexicon:

```
Do until there is no input left:
1.  If the agenda is empty, look up the interpretations for the next word in the
    input and add them to the agenda.
2.  Select a constituent from the agenda (let's call it constituent C from position
    $p_1$ to $p_2$).
3.  For each rule in the grammar of form $X \rightarrow C X_1 ... X_n$, add an active arc of
    form $X \rightarrow \circ C X_1 ... X_n$ from position $p_1$ to $p_2$.
4.  Add C to the chart using the arc extension algorithm above.
```

Figure 3.11 A bottom-up chart parsing algorithm

the:	ART
large:	ADJ
can:	N, AUX, V
hold:	N, V
water:	N, V

To best understand the example, draw the chart as it is extended at each step of the algorithm. The agenda is initially empty, so the word *the* is read and a constituent ART1 placed on the agenda.

Entering ART1: (*the* from 1 to 2)
Adds active arc NP $\rightarrow$ ART $\circ$ ADJ N from 1 to 2
Adds active arc NP $\rightarrow$ ART $\circ$ N from 1 to 2

Both these active arcs were added by step 3 of the parsing algorithm and were derived from rules 2 and 3 in the grammar, respectively. Next the word *large* is read and a constituent ADJ1 is created.

Entering ADJ1: (*large* from 2 to 3)
Adds arc NP $\rightarrow$ ADJ $\circ$ N from 2 to 3
Adds arc NP $\rightarrow$ ART ADJ $\circ$ N from 1 to 3

The first arc was added in step 3 of the algorithm. The second arc added here is an extension of the first active arc that was added when ART1 was added to the chart using the arc extension algorithm (step 4).

The chart at this point has already been shown in Figure 3.9. Notice that active arcs are never removed from the chart. For example, when the arc NP $\rightarrow$ ART $\circ$ ADJ N from 1 to 2 was extended, producing the arc from 1 to 3, both arcs remained on the chart. This is necessary because the arcs could be used again in a different way by another interpretation.

For the next word, *can*, three constituents, N1, AUX1, and V1 are created for its three interpretations.

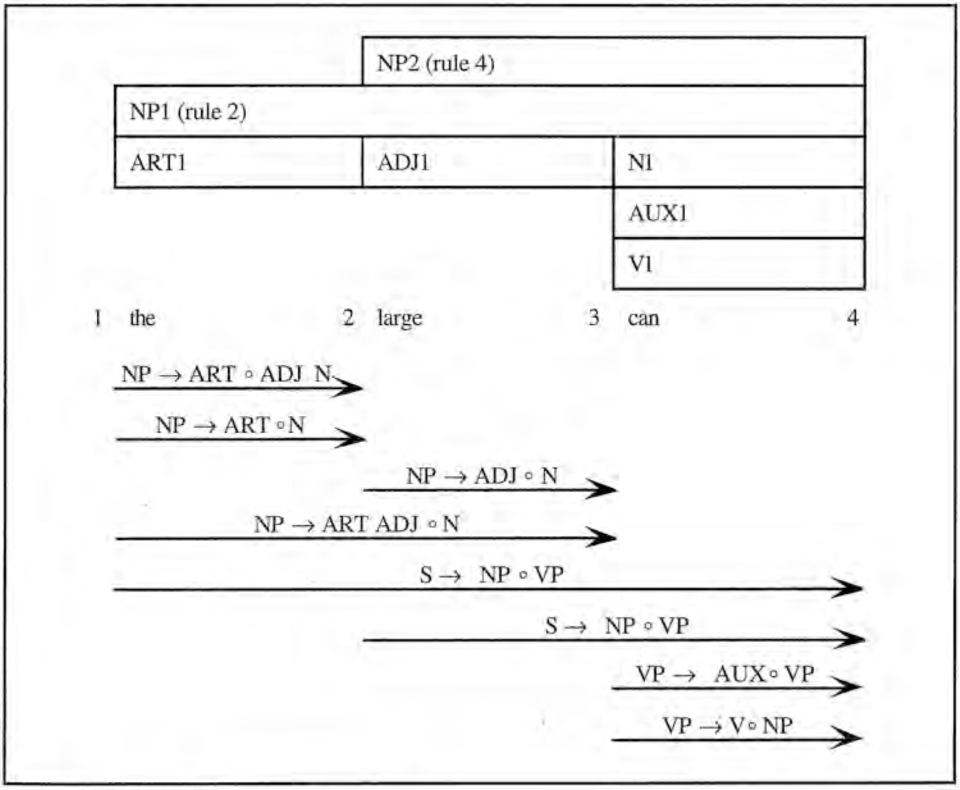


Figure 3.12 After parsing *the large can*

Entering N1: (*can* from 3 to 4)

No active arcs are added in step 2, but two are completed in step 4 by the arc extension algorithm, producing two NPs that are added to the agenda: The first, an NP from 1 to 4, is constructed from rule 2, while the second, an NP from 2 to 4, is constructed from rule 4. These NPs are now at the top of the agenda.

Entering NP1: an NP (*the large can* from 1 to 4)
    Adding active arc S → NP ∘ VP from 1 to 4
Entering NP2: an NP (*large can* from 2 to 4)
    Adding arc S → NP ∘ VP from 2 to 4
Entering AUX1: (*can* from 3 to 4)
    Adding arc VP → AUX ∘ VP from 3 to 4
Entering V1: (*can* from 3 to 4)
    Adding arc VP → V ∘ NP from 3 to 4

The chart is shown in Figure 3.12, which illustrates all the completed constituents (NP2, NP1, ART1, ADJ1, N1, AUX1, V1) and all the uncompleted active arcs entered so far. The next word is *can* again, and N2, AUX2, and V2 are created.

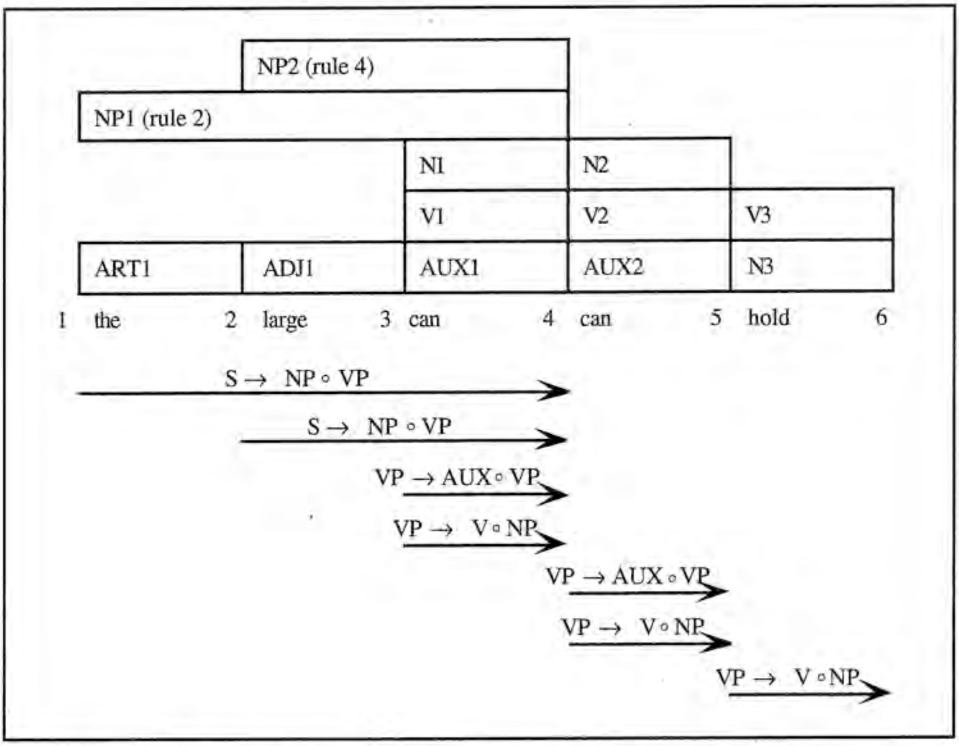


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

Entering N2: (*can* from 4 to 5, the second *can*)
  Adds no active arcs
Entering AUX2: (*can* from 4 to 5)
  Adds arc VP → AUX ∘ VP from 4 to 5
Entering V2: (*can* from 4 to 5)
  Adds arc VP → V ∘ NP from 4 to 5

The next word is *hold*, and N3 and V3 are created.

Entering N3: (*hold* from 5 to 6)
  Adds no active arcs
Entering V3: (*hold* from 5 to 6)
  Adds arc VP → V ∘ NP from 5 to 6

The chart in Figure 3.13 shows all the completed constituents built so far, together with all the active arcs, except for those used in the first NP.

Entering ART2: (*the* from 6 to 7)
  Adding arc NP → ART ∘ ADJ N from 6 to 7
  Adding arc NP → ART ∘ N from 6 to 7

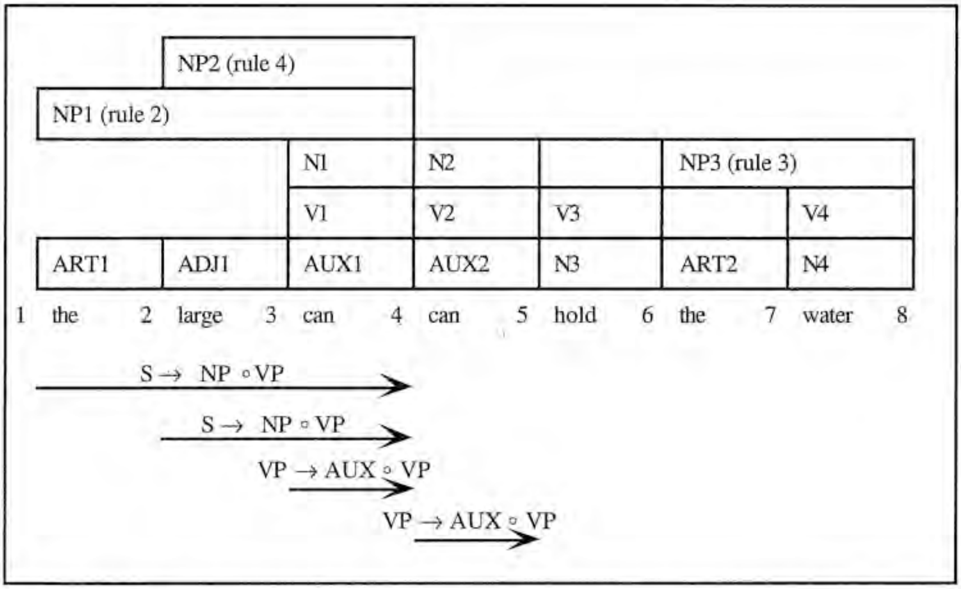


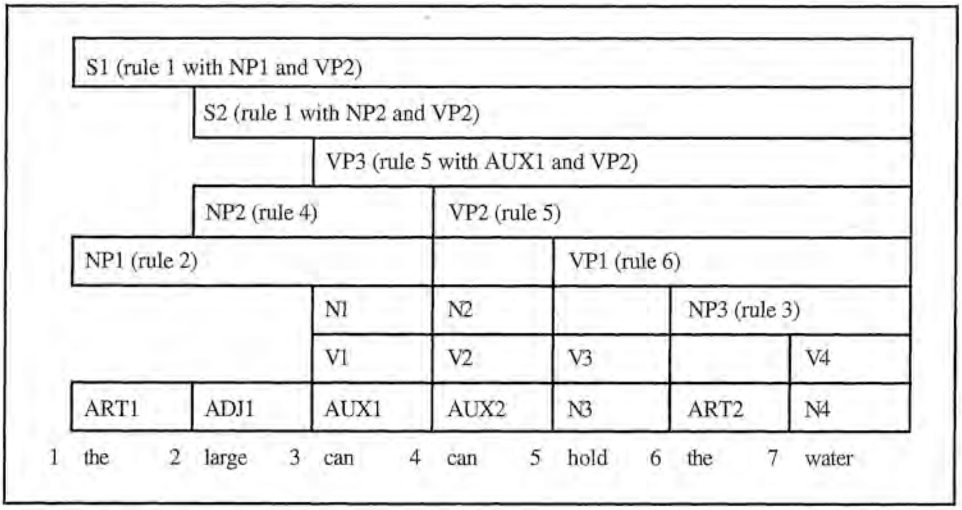
Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

Entering N4: (*water* from 7 to 8)
  No active arcs added in step 3
  An NP, NP3, from 6 to 8 is pushed onto the agenda, by completing
    arc NP $\rightarrow$ ART ∘ N from 6 to 7
Entering NP3: (*the water* from 6 to 8)
  A VP, VP1, from 5 to 8 is pushed onto the agenda, by completing
    VP $\rightarrow$ V ∘ NP from 5 to 6
  Adds arc S $\rightarrow$ NP ∘ VP from 6 to 8

The chart at this stage is shown in Figure 3.14, but only the active arcs to be used in the remainder of the parse are shown.

Entering VP1: (*hold the water* from 5 to 8)
  A VP, VP2, from 4 to 8 is pushed onto the agenda, by completing
    VP $\rightarrow$ AUX ∘ VP from 4 to 5
Entering VP2: (*can hold the water* from 4 to 8)
  An S, S1, is added from 1 to 8, by completing
    arc S $\rightarrow$ NP ∘ VP from 1 to 4
  A VP, VP3, is added from 3 to 8, by completing
    arc VP $\rightarrow$ AUX ∘ VP from 3 to 4
  An S, S2, is added from 2 to 8, by completing
    arc S $\rightarrow$ NP ∘ VP from 2 to 4

Since you have derived an S covering the entire sentence, you can stop successfully. If you wanted to find all possible interpretations for the sentence,



S1 (rule 1 with NP1 and VP2)
S2 (rule 1 with NP2 and VP2)
VP3 (rule 5 with AUX1 and VP2)
NP2 (rule 4)
VP2 (rule 5)
NP1 (rule 2)
VP1 (rule 6)
N1
N2
NP3 (rule 3)
V1
V2
V3
V4
ART1
ADJ1
AUX1
AUX2
N3
ART2
N4
1 the 2 large 3 can 4 can 5 hold 6 the 7 water

Figure 3.15 The final chart

you would continue parsing until the agenda became empty. The chart would then contain as many S structures covering the entire set of positions as there were different structural interpretations. In addition, this representation of the entire set of structures would be more efficient than a list of interpretations, because the different S structures might share common subparts represented in the chart only once. Figure 3.15 shows the final chart.

Efficiency Considerations

Chart-based parsers can be considerably more efficient than parsers that rely only on a search because the same constituent is never constructed more than once. For instance, a pure top-down or bottom-up search strategy could require up to C^n operations to parse a sentence of length n, where C is a constant that depends on the specific algorithm you use. Even if C is very small, this exponential complexity rapidly makes the algorithm unusable. A chart-based parser, on the other hand, in the worst case would build every possible constituent between every possible pair of positions. This allows us to show that it has a worst-case complexity of $K*n^3$, where n is the length of the sentence and K is a constant depending on the algorithm. Of course, a chart parser involves more work in each step, so K will be larger than C. To contrast the two approaches, assume that C is 10 and that K is a hundred times worse, 1000. Given a sentence of 12 words, the brute force search might take 10^{12} operations (that is, 1,000,000,000,000), whereas the chart parser would take $1000 * 12^3$ (that is, 1,728,000). Under these assumptions, the chart parser would be up to 500,000 times faster than the brute force search on some examples!

3.6 Top-Down Chart Parsing

So far, you have seen a simple top-down method and a bottom-up chart-based method for parsing context-free grammars. Each of the approaches has its advantages and disadvantages. In this section a new parsing method is presented that actually captures the advantages of both. But first, consider the pluses and minuses of the approaches.

Top-down methods have the advantage of being highly predictive. A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered. For example, consider Grammar 3.8 in a top-down parse of the sentence *The can holds the water,* where *can* may be an AUX, V, or N, as before.

The top-down parser would rewrite (S) to (NP VP) and then rewrite the NP to produce three possibilities, (ART ADJ N VP), (ART N VP), and (ADJ N VP). Taking the first, the parser checks if the first word, *the*, can be an ART, and then if the next word, *can*, can be an ADJ, which fails. Trying the next possibility, the parser checks *the* again, and then checks if *can* can be an N, which succeeds. The interpretations of *can* as an auxiliary and a main verb are never considered because no syntactic tree generated by the grammar would ever predict an AUX or V in this position. In contrast, the bottom-up parser would have considered all three interpretations of *can* from the start—that is, all three would be added to the chart and would combine with active arcs. Given this argument, the top-down approach seems more efficient.

On the other hand, consider the top-down parser in the example above needed to check that the word *the* was an ART twice, once for each rule. This reduplication of effort is very common in pure top-down approaches and becomes a serious problem, and large constituents may be rebuilt again and again as they are used in different rules. In contrast, the bottom-up parser only checks the input once, and only builds each constituent exactly once. So by this argument, the bottom-up approach appears more efficient.

You can gain the advantages of both by combining the methods. A small variation in the bottom-up chart algorithm yields a technique that is predictive like the top-down approaches yet avoids any reduplication of work as in the bottom-up approaches.

As before, the algorithm is driven by an agenda of completed constituents and the arc extension algorithm, which combines active arcs with constituents when they are added to the chart. While both use the technique of extending arcs with constituents, the difference is in how new arcs are generated from the grammar. In the bottom-up approach, new active arcs are generated whenever a completed constituent is added that could be the first constituent of the right-hand side of a rule. With the top-down approach, new active arcs are generated whenever a new active arc is added to the chart, as described in the top-down arc introduction algorithm shown in Figure 3.22. The parsing algorithm is then easily stated, as is also shown in Figure 3.22.

Consider this new algorithm operating with the same grammar on the same sentence as in Section 3.4, namely *The large can can hold the water.* In the initialization stage, an arc labeled S → ∘ NP VP is added. Then, active arcs for each rule that can derive an NP are added: NP → ∘ ART ADJ N, NP → ∘ ART N,

Top-Down Arc Introduction Algorithm

To add an arc $S \rightarrow C_1 \ldots \circ C_i \ldots C_n$ ending at position j, do the following:

For each rule in the grammar of form $C_i \rightarrow X_1 \ldots X_k$, recursively add the new arc $C_i \rightarrow \circ X_1 \ldots X_k$ from position j to j.

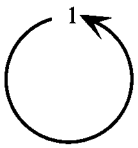
Top-Down Chart Parsing Algorithm

Initialization: For every rule in the grammar of form $S \rightarrow X_1 \ldots X_k$, add an arc labeled $S \rightarrow \circ X_1 \ldots X_k$ using the arc introduction algorithm.

Parsing: Do until there is no input left:

1. If the agenda is empty, look up the interpretations of the next word and add them to the agenda.
2. Select a constituent from the agenda (call it constituent C).
3. Using the arc extension algorithm, combine C with every active arc on the chart. Any new constituents are added to the agenda.
4. For any active arcs created in step 3, add them to the chart using the top-down arc introduction algorithm.

Figure 3.22 The top-down arc introduction and chart parsing algorithms



$S \rightarrow \circ$ NP VP
$NP \rightarrow \circ$ ADJ N
$NP \rightarrow \circ$ ART N
$NP \rightarrow \circ$ ART ADJ N

Figure 3.23 The initial chart

and NP $\rightarrow \circ$ ADJ N are all added from position 1 to 1. Thus the initialized chart is as shown in Figure 3.23. The trace of the parse is as follows:

Entering ART1 (*the*) from 1 to 2
  Two arcs can be extended by the arc extension algorithm
    NP $\rightarrow$ ART $\circ$ N from 1 to 2
    NP $\rightarrow$ ART $\circ$ ADJ N from 1 to 2
Entering ADJ1 (*large*) from 2 to 3
  One arc can be extended
    NP $\rightarrow$ ART ADJ $\circ$ N from 1 to 3
Entering AUX1 (*can*) from 3 to 4
  No activity, constituent is ignored
Entering V1 (*can*) from 3 to 4
  No activity, constituent is ignored

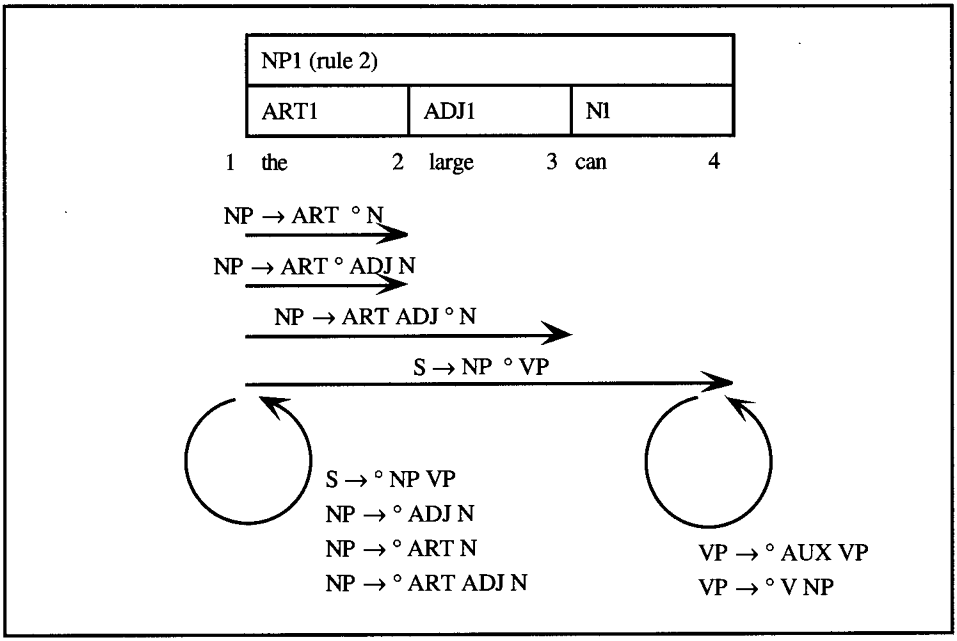


Figure 3.24 The chart after building the first NP

Entering N1 (*can*) from 3 to 4
    One arc extended and completed yielding
        NP1 from 1 to 4 (*the large can*)
Entering NP1 from 1 to 4
    One arc can be extended
        S → NP ∘ VP from 1 to 4
    Using the top-down rule (step 4), new active arcs are added for VP
        VP → ∘ AUX VP from 4 to 4
        VP → ∘ V NP from 4 to 4

At this stage, the chart is as shown in Figure 3.24. Compare this with Figure 3.10. It contains fewer completed constituents since only those that are allowed by the top-down filtering have been constructed.

The algorithm continues, adding the three interpretations of *can* as an AUX, V, and N. The AUX reading extends the VP → ∘ AUX VP arc at position 4 and adds active arcs for a new VP starting at position 5. The V reading extends the VP → ∘ V NP arc and adds active arcs for an NP starting at position 5. The N reading does not extend any arc and so is ignored. After the two readings of *hold* (as an N and V) are added, the chart is as shown in Figure 3.25. Again, compare with the corresponding chart for the bottom-up parser in Figure 3.13. The rest of the sentence is parsed similarly, and the final chart is shown in Figure 3.26. In comparing this to the final chart produced by the bottom-up parser (Figure 3.15), you see that the number of constituents generated has dropped from 21 to 13.

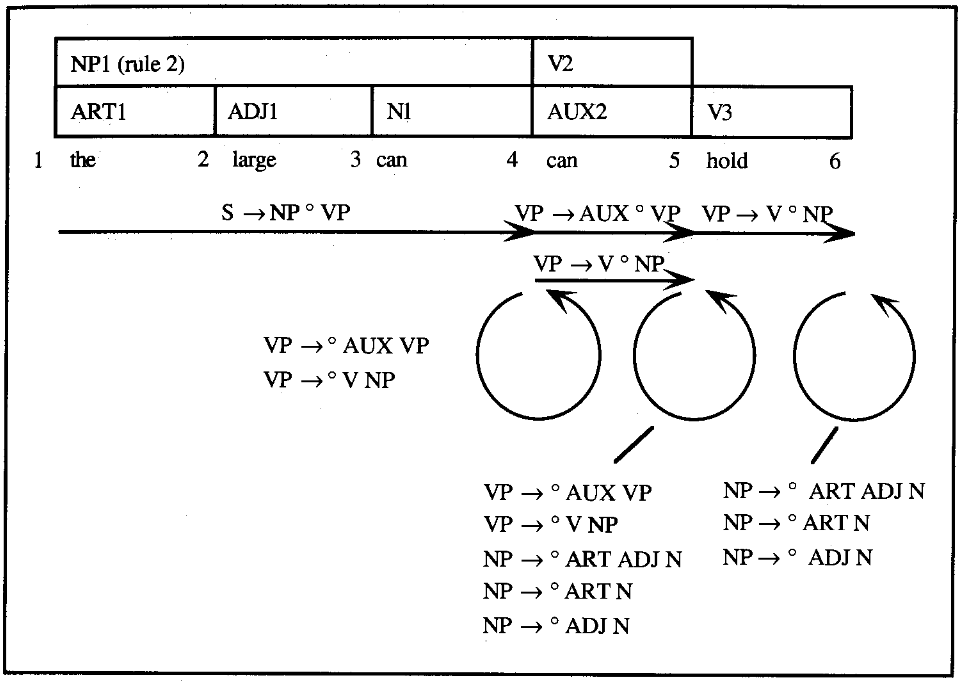


Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

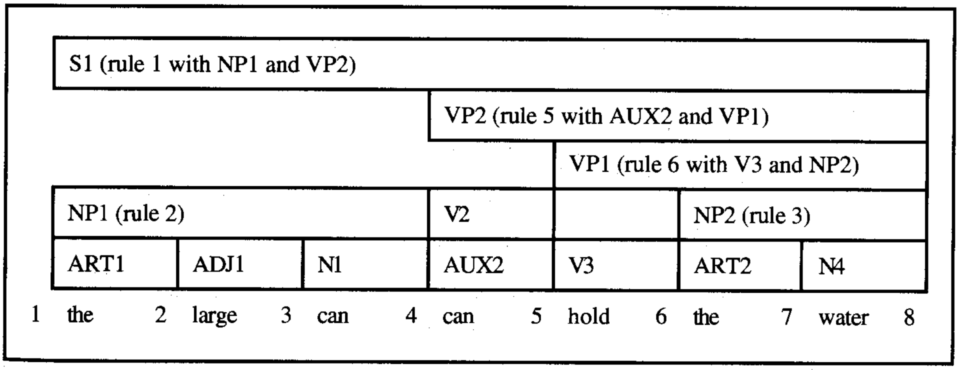

S1 (rule 1 with NP1 and VP2)
VP2 (rule 5 with AUX2 and VP1)
VP1 (rule 6 with V3 and NP2)
NP1 (rule 2)
V2
NP2 (rule 3)
ART1
ADJ1
N1
AUX2
V3
ART2
N4
1 the 2 large 3 can 4 can 5 hold 6 the 7 water 8

Figure 3.26 The final chart for the top-down filtering algorithm

While it is not a big difference here with such a simple grammar, the difference can be dramatic with a sizable grammar.

It turns out in the worst-case analysis that the top-down chart parser is not more efficient that the pure bottom-up chart parser. Both have a worst-case complexity of $K{*}n^3$ for a sentence of length n. In practice, however, the top-down method is considerably more efficient for any reasonable grammar.