$$S \rightarrow \bullet\, VP,\ [0,0]$$
$$NP \rightarrow Det \bullet\, Nominal,\ [1,2]$$
$$VP \rightarrow V\, NP \bullet,\ [0,3]$$

The first state, with its dot to the left of its constituent, represents a top-down prediction for this particular kind of *S*. The first 0 indicates that the constituent predicted by this state should begin at the start of the input; the second 0 reflects the fact that the dot lies at the beginning as well. The second state, created at a later stage in the processing of this sentence, indicates that an *NP* begins at position 1, that a *Det* has been successfully parsed and that a *Nominal* is expected next. The third state, with its dot to the right of all its two constituents, represents the successful discovery of a tree corresponding to a *VP* that spans the entire input.

The fundamental operation of an Earley parser is to march through the $N+1$ sets of states in the chart in a left-to-right fashion, processing the states within each set in order. At each step, one of the three operators described below is applied to each state depending on its status. In each case, this results in the addition of new states to the end of either the current, or next, set of states in the chart. The algorithm always moves forward through the chart making additions as it goes; states are never removed and the algorithm never backtracks to a previous chart entry once it has moved on. The presence of a state $S \rightarrow \alpha\bullet,\ [0,N]$ in the list of states in the last chart entry indicates a successful parse. Fig. 13.13 gives the complete algorithm.

The following three sections describe in detail the three operators used to process states in the chart. Each takes a single state as input and derives new states from it. These new states are then added to the chart as long as they are not already present. The PREDICTOR and the COMPLETER add states to the chart entry being processed, while the SCANNER adds a state to the next chart entry.

Predictor

As might be guessed from its name, the job of PREDICTOR is to create new states representing top-down expectations generated during the parsing process. PREDICTOR is applied to any state that has a non-terminal immediately to the right of its dot that is not a part-of-speech category. This application results in the creation of one new state for each alternative expansion of that non-terminal provided by the grammar. These new states are placed into the same chart entry as the generating state. They begin and end at the point in the input where the generating state ends.

For example, applying PREDICTOR to the state $S \rightarrow \bullet\, VP,\ [0,0]$ results in the addition of the following five states

$$VP \rightarrow \bullet\, Verb,\ [0,0]$$
$$VP \rightarrow \bullet\, Verb\ NP,\ [0,0]$$
$$VP \rightarrow \bullet\, Verb\ NP\ PP,\ [0,0]$$
$$VP \rightarrow \bullet\, Verb\ PP,\ [0,0]$$
$$VP \rightarrow \bullet\, VP\ PP,\ [0,0]$$

to the first chart entry.

```
function EARLEY-PARSE(words, grammar) returns chart

  ADDTOCHART((γ → • S, [0,0]), chart[0])
  for i ← from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
          PREDICTOR(state)
      elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
          SCANNER(state)
      else
          COMPLETER(state)
    end
  end
  return(chart)

  procedure PREDICTOR((A → α • B β, [i, j]))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ADDTOCHART((B → • γ, [j, j]), chart[j])
    end

  procedure SCANNER((A → α • B β, [i, j]))
    if B ∈ PARTS-OF-SPEECH(word[jj]) then
        ADDTOCHART((B → word[j] •, [j, j+1]), chart[j+1])

  procedure COMPLETER((B → γ •, [j,k]))
    for each (A → α • B β, [i, j]) in chart[j] do
        ADDTOCHART((A → α B • β, [i,k]), chart[k])
    end

  procedure ADDTOCHART(state, chart-entry)
    if state is not already in chart-entry then
        PUSH-ON-END(state, chart-entry)
    end
```

Figure 13.13 The Earley algorithm

Scanner

When a state has a part-of-speech category to the right of the dot, SCANNER is called to examine the input and incorporate a state corresponding to the prediction of a word with a particular part-of-speech into the chart. This is accomplished by creating a new state from the input state with the dot advanced over the predicted input category. Note that unlike CKY, Earley uses top-down input to help deal with part-of-speech ambiguities; only those parts-of-speech of a word that are predicted by some existing state will find their way into the chart.

Returning to our example, when the state $VP \rightarrow \bullet\ Verb\ NP,\ [0,0]$ is processed, SCANNER consults the current word in the input since the category following the dot is

a part-of-speech. It then notes that *book* can be a verb, matching the expectation in the current state. This results in the creation of the new state $Verb \rightarrow book\bullet, [0,1]$. This new state is then added to the chart entry that *follows* the one currently being processed. The noun sense of *book* never enters the chart since it is not predicted by any rule at this position in the input.

Completer

COMPLETER is applied to a state when its dot has reached the right end of the rule. The presence of such a state represents the fact that the parser has successfully discovered a particular grammatical category over some span of the input. The purpose of COMPLETER is to find, and advance, all previously created states that were looking for this grammatical category at this position in the input. New states are then created by **copying** the older state, advancing the dot over the expected category, and installing the new state in the current chart entry.

In the current example, when the state $NP \rightarrow Det\ Nominal\bullet, [1,3]$ is processed, COMPLETER looks for incomplete states ending at position 1 and expecting an *NP*. It finds the states $VP \rightarrow Verb\bullet NP, [0,1]$ and $VP \rightarrow Verb\bullet NP\ PP, [0,1]$. This results in the addition of the new complete state $VP \rightarrow Verb\ NP\bullet, [0,3]$, and the new incomplete state $VP \rightarrow Verb\ NP\bullet PP, [0,3]$ to the chart.

A Complete Example

Fig. 13.14 shows the sequence of states created during the complete processing of Ex. 13.7; each row indicates the state number for reference, the dotted rule, the start and end points, and finally the function that added this state to the chart. The algorithm begins by seeding the chart with a top-down expectation for an *S*. This is accomplished by adding a dummy state $\gamma \rightarrow \bullet\ S, [0,0]$ to Chart[0]. When this state is processed, it is passed to PREDICTOR leading to the creation of the three states representing predictions for each possible type of *S*, and transitively to states for all of the left-corners of those trees. When the state $VP \rightarrow \bullet\ Verb, [0,0]$ is reached, SCANNER is called and the first word is read. A state representing the verb sense of *Book* is added to the entry for Chart[1]. Note that when the subsequent sentence initial *VP* states are processed, SCANNER will be called again. However, new states are not added since they would be identical to the *Verb* state already in the chart.

When all the states of Chart[0] have been processed, the algorithm moves on to Chart[1] where it finds the state representing the verb sense of *book*. This is a complete state with its dot to the right of its constituent and is therefore passed to COMPLETER. COMPLETER then finds the four previously existing *VP* states expecting a *V*erb at this point in the input. These states are copied with their dots advanced and added to Chart[1]. The completed state corresponding to an intransitive *VP* then leads to the creation of an *S* representing an imperative sentence. Alternatively, the dot in the transitive verb phrase leads to the creation of the three states predicting different forms of *NP*s. The state $NP \rightarrow \bullet\ Det\ Nominal, [1,1]$ causes SCANNER to read the word *that* and add a corresponding state to Chart[2].

Moving on to Chart[2], the algorithm finds the state representing the determiner sense of *that*. This complete state leads to the advancement of the dot in the *NP* state

Chart[0]	S0	$\gamma \rightarrow \bullet\, S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet\, NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet\, Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet\, VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet\, Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet\, Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet\, Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet\, Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet\, Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet\, Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet\, Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet\, VP\ PP$	[0,0]	Predictor
Chart[1]	S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
	S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
	S14	$VP \rightarrow Verb \bullet\, NP$	[0,1]	Completer
	S15	$VP \rightarrow Verb \bullet\, NP\ PP$	[0,1]	Completer
	S16	$VP \rightarrow Verb \bullet\, PP$	[0,1]	Completer
	S17	$S \rightarrow VP \bullet$	[0,1]	Completer
	S18	$VP \rightarrow VP \bullet\, PP$	[0,1]	Completer
	S19	$NP \rightarrow \bullet\, Pronoun$	[1,1]	Predictor
	S20	$NP \rightarrow \bullet\, Proper\text{-}Noun$	[1,1]	Predictor
	S21	$NP \rightarrow \bullet\, Det\ Nominal$	[1,1]	Predictor
	S22	$PP \rightarrow \bullet\, Prep\ NP$	[1,1]	Predictor
Chart[2]	S23	$Det \rightarrow that \bullet$	[1,2]	Scanner
	S24	$NP \rightarrow Det \bullet\, Nominal$	[1,2]	Completer
	S25	$Nominal \rightarrow \bullet\, Noun$	[2,2]	Predictor
	S26	$Nominal \rightarrow \bullet\, Nominal\ Noun$	[2,2]	Predictor
	S27	$Nominal \rightarrow \bullet\, Nominal\ PP$	[2,2]	Predictor
Chart[3]	S28	$Noun \rightarrow flight \bullet$	[2,3]	Scanner
	S29	$Nominal \rightarrow Noun \bullet$	[2,3]	Completer
	S30	$NP \rightarrow Det\ Nominal \bullet$	[1,3]	Completer
	S31	$Nominal \rightarrow Nominal \bullet\, Noun$	[2,3]	Completer
	S32	$Nominal \rightarrow Nominal \bullet\, PP$	[2,3]	Completer
	S33	$VP \rightarrow Verb\ NP \bullet$	[0,3]	Completer
	S34	$VP \rightarrow Verb\ NP \bullet\, PP$	[0,3]	Completer
	S35	$PP \rightarrow \bullet\, Prep\ NP$	[3,3]	Predictor
	S36	$S \rightarrow VP \bullet$	[0,3]	Completer
	S37	$VP \rightarrow VP \bullet\, PP$	[0,3]	Completer

Figure 13.14 Chart entries created during an Earley parse of *Book that flight.* Each entry shows the state, its start and end points, and the function that placed it in the chart.

predicted in Chart[1], and also to the predictions for the various kinds of *Nominal*. The first of these causes SCANNER to be called for the last time to process the word *flight*.

Finally moving on to Chart[3], the presence of the state representing *flight* leads in quick succession to the completion of an *NP*, transitive *VP*, and an *S*. The presence of the state $S \rightarrow VP\bullet$, $[0,3]$ in the last chart entry signals the discovery of a successful

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	(S28)
	S30	*NP* → *Det Nominal* •	[1,3]	(S23, S29)
	S33	*VP* → *Verb NP* •	[0,3]	(S12, S30)
	S36	*S* → *VP* •	[0,3]	(S33)

Figure 13.15 States that participate in the final parse of *Book that flight*, including structural parse information.

parse.

It is useful to contrast this example with the CKY example given earlier. Although Earley managed to avoid adding an entry for the noun sense of *book*, its overall behavior is clearly much more promiscuous than CKY. This promiscuity arises from the purely top-down nature of the predictions that Earley makes. Exercise 13.6 asks you to improve the algorithm by eliminating some of these unnecessary predictions.

Retrieving Parse Trees from a Chart

As with the CKY algorithm, this version of the Earley algorithm is a recognizer not a parser. Valid sentences will simply leave the state $S \rightarrow \alpha\bullet$, $[0,N]$ in the chart. To retrieve parses from the chart the representation of each state must be augmented with an additional field to store information about the completed states that generated its constituents.

The information needed to fill these fields can be gathered by making a simple change to the COMPLETER function. Recall that COMPLETER creates new states by advancing existing incomplete states when the constituent following the dot has been discovered in the right place. The only change necessary is to have COMPLETER add a pointer to the older state onto a list of constituent-states for the new state. Retrieving a parse tree from the chart is then merely a matter of following pointers starting with the state (or states) representing a complete S in the final chart entry. Fig. 13.15 shows the chart entries produced by an appropriately updated COMPLETER that participate in the final parse for this example.

13.4.3 Chart Parsing

In both the CKY and Earley algorithms, the order in which events occur (adding entries to the table, reading words, making predictions, etc.) is statically determined by the procedures that make up these algorithms. Unfortunately, dynamically determining the order in which events occur based on the current information is often necessary for a variety of reasons. Fortunately, an approach advanced by Martin Kay and his colleagues (Kaplan, 1973; Kay, 1986) called **Chart Parsing** facilitates just such dynamic determination of the order in which chart entries are processed. This is accomplished through the introduction of an *agenda* to the mix. In this scheme, as states (called **edges**

CHART PARSING

in this approach) are created they are added to an agenda that is kept ordered according to a policy that is specified *separately* from the main parsing algorithm. This can be viewed as another instance of state-space search that we've seen several times before. The FSA and FST recognition and parsing algorithms in Chs. 2 and 3 employed agendas with simple static policies, while the A* decoding algorithm described in Ch. 9 is driven by an agenda that is ordered probabilistically.

Fig. 13.16 presents a generic version of a parser based on such a scheme. The main part of the algorithm consists of a single loop that removes a edge from the front of an agenda, processes it, and then moves on to the next entry in the agenda. When the agenda is empty, the parser stops and returns the chart. The policy used to order the elements in the agenda thus determines the order in which further edges are created and predictions are made.

```
function CHART-PARSE(words, grammar, agenda-strategy) returns chart

  INITIALIZE(chart, agenda, words)
  while agenda
      current-edge ← POP(agenda)
      PROCESS-EDGE(current-edge)
  return(chart)

procedure PROCESS-EDGE(edge)
    ADD-TO-CHART(edge)
    if INCOMPLETE?(edge)
        FORWARD-FUNDAMENTAL-RULE(edge)
    else
        BACKWARD-FUNDAMENTAL-RULE(edge)
    MAKE-PREDICTIONS(edge)

procedure FORWARD-FUNDAMENTAL((A → α • B β, [i,j]))
  for each(B → γ •, [j,k]) in chart
      ADD-TO-AGENDA(A → α B • β, [i,k])

procedure BACKWARD-FUNDAMENTAL((B → γ •, [j,k]))
  for each(A → α • B β, [i,j]) in chart
      ADD-TO-AGENDA(A → α B • β, [i,k])

procedure ADD-TO-CHART(edge)
  if edge is not already in chart then
      Add edge to chart

procedure ADD-TO-AGENDA(edge)
  if edge is not already in agenda then
      APPLY(agenda-strategy, edge, agenda)
```

Figure 13.16 A Chart Parsing Algorithm

FUNDAMENTAL RULE

The key principle in processing edges in this approach is what Kay termed the **fundamental rule** of chart parsing. The fundamental rule states that when the chart contains two contiguous edges where one of the edges provides the constituent that

the other one needs, a new edge should be created that spans the original edges and incorporates the provided material. More formally, the fundamental rule states the following: if the chart contains two edges $A \rightarrow \alpha \bullet B\,\beta, [i,j]$ and $B \rightarrow \gamma \bullet, [j,k]$ then we should add the new edge $A \rightarrow \alpha\, B \bullet \beta\, [i,k]$ to the chart. It should be clear that the fundamental rule is a generalization of the basic table-filling operations found in both the CKY and Earley algorithms.

The fundamental rule is triggered in Fig. 13.16 when an edge is removed from the agenda and passed to the PROCESS-EDGE procedure. Note that the fundamental rule itself does not specify which of the two edges involved has triggered the processing. PROCESS-EDGE handles both cases by checking to see whether or not the edge in question is complete. If it is complete than the algorithm looks earlier in the chart to see if any existing edge can be advanced; if it is incomplete than it looks later in the chart to see if it can be advanced by any pre-existing edge later in the chart.

The next piece of the algorithm that needs to be filled in is the method for making predictions based on the edge being processed. There are two key components to making predictions in chart parsing: the events that trigger predictions, and the nature of a predictions. The nature of these components varies depending on whether we are pursuing a top-down or bottom-up strategy. As in Earley, top-down predictions are triggered by expectations that arise from incomplete edges that have been entered into the chart; bottom-up predictions are triggered by the discovery of completed constituents. Fig. 13.17 illustrates how these two strategies can be integrated into the chart parsing algorithm.

```
procedure MAKE-PREDICTIONS(edge)
  if Top-Down and INCOMPLETE?(edge)
    TD-PREDICT(edge)
  elsif Bottom-Up and COMPLETE?(edge)
    BU-PREDICT(edge)

procedure TD-PREDICT((A → α • B β, [i,j]))
  for each(B → γ) in grammar do
    ADD-TO-AGENDA(B → • γ, [j,j])

procedure BU-PREDICT((B → γ •, [i,j]))
  for each(A → B β) in grammar
    ADD-TO-AGENDA(A → B • β, [i,j])
```

Figure 13.17 A Chart Parsing Algorithm

Obviously we've left out many of the bookkeeping details that would have to be specified to turn this approach into a real parser. Among the details that have to be worked out are how the INITIALIZE procedure gets things started, how and when words are read, the organization of the chart, and specifying an agenda strategy. Indeed, in describing the approach here, Kay (1986) refers to it as an **algorithm!schema** rather than an algorithm, since it more accurately specifies an entire family of parsers rather than any particular parser. Exercise 13.7 asks you to explore some of the available

ALGORITHM!SCHEMA

choices by implementing various chart parsers.

13.5 Partial Parsing

PARTIAL PARSE
SHALLOW PARSE

Many language-processing tasks simply do not require complex, complete parse trees for all inputs. For these tasks, a **partial parse**, or **shallow parse**, of input sentences may be sufficient. For example, **information extraction** systems generally do not extract *all* the possible information from a text; they simply identify and classify the segments in a text that are likely to contain valuable information. Similarly, information retrieval systems may choose to index documents based on a select subset of the constituents found in a text.

Not surprisingly, there are many different approaches to partial parsing. Some approaches make use of cascades of FSTs, of the kind discussed in Ch. 3, to to produce representations that closely approximate the kinds of trees we've been assuming in this chapter and the last. These approaches typically produce flatter trees than the ones we've been discussing. This flatness arises from the fact that such approaches generally defer decisions that may require semantic or contextual factors, such as prepositional phrase attachments, coordination ambiguities, and nominal compound analyses. Nevertheless the intent is to produce parse-trees that link all the major constituents in an input.

CHUNKING

An alternative style of partial parsing is known as **chunking**. Chunking is the process of identifying and classifying the flat non-overlapping segments of a sentence that constitute the basic non-recursive phrases corresponding to the major parts-of-speech found in most wide-coverage grammars. This set typically includes noun phrases, verb phrases, adjective phrases, and prepositional phrases; in other words, the phrases that correspond to the content-bearing parts-of-speech. Of course, not all applications require the identification of all of these categories; indeed the most common chunking task is to simply find all the base noun phrases in a text.

Since chunked texts lack a hierarchical structure, a simple bracketing notation is sufficient to denote the location and the type of the chunks in a given example. The following example illustrates a typical bracketed notation.

(13.8) [$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived.]

This bracketing notation makes clear the two fundamental tasks that are involved in chunking: finding the non-overlapping extents of the chunks, and assigning the correct label to the discovered chunks.

Note that in this example all the words are contained in some chunk. This will not be the case in all chunking applications. In many settings, a good number of the words in any input will fall outside of any chunk. This is, for example, the norm in systems that are only interested in finding the base-NPs in their inputs, as illustrated by the following example.

(13.9) [$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived.

The details of what constitutes a syntactic base-phrase for any given system varies according to the syntactic theories underlying the system and whether the phrases