First sets for all symbols:
for all terminals $a$, $\text{FIRST}(a) := \{a\}$
for all nonterminals $X$, $\text{FIRST}(X) := \emptyset$
for all productions $X \rightarrow \epsilon$, add $\epsilon$ to $\text{FIRST}(X)$
repeat
  (outer) for all productions $X \rightarrow Y_1 \, Y_2 \ldots Y_k$,
  (inner) for $i$ in $1 \ldots k$
  add $(\text{FIRST}(Y_i) \setminus \{\epsilon\})$ to $\text{FIRST}(X)$
  if $\epsilon \notin \text{FIRST}(Y_i)$ (yet)
       continue outer loop
  add $\epsilon$ to $\text{FIRST}(X)$
until no further progress

First set subroutine for string $X_1 \, X_2 \ldots X_n$, similar to inner loop above:
return value := $\emptyset$
for $i$ in $1 \ldots n$
  add $(\text{FIRST}(X_i) \setminus \{\epsilon\})$ to return value
  if $\epsilon \notin \text{FIRST}(X_i)$
      return
  add $\epsilon$ to return value

Follow sets for all symbols:
$\text{FOLLOW}(S) := \{\epsilon\}$, where $S$ is the start symbol
for all other symbols $X$, $\text{FOLLOW}(X) := \emptyset$
repeat
  for all productions $A \rightarrow \alpha$ $B$ $\beta$,
    add $(\text{FIRST}(\beta) \setminus \{\epsilon\})$ to $\text{FOLLOW}(B)$
  for all productions $A \rightarrow \alpha$ $B$
    or $A \rightarrow \alpha$ $B$ $\beta$, where $\epsilon \in \text{FIRST}(B)$,
    add $\text{FOLLOW}(A)$ to $\text{FOLLOW}(B)$
until no further progress

Predict sets for all productions:
for all productions $A \rightarrow \alpha$
$\text{PREDICT}(A \rightarrow \alpha) := (\text{FIRST}(\alpha) \setminus \{\epsilon\})$
  $\cup$ (if $\epsilon \in \text{FIRST}(\alpha)$ then $\text{FOLLOW}(A)$ else $\emptyset$)

Figure 2.23 Algorithm to calculate first, follow, and predict sets. The grammar is LL(1) if and only if the predict sets are disjoint.

(i.e., converge on a solution), because the sizes of the sets are bounded by the number of terminals in the grammar.

If in the process of calculating predict sets we find that some token belongs to the predict set of more than one production with the same left-hand side, then the grammar is not LL(1), because we will not be able to choose which of the productions to employ when the left-hand side is at the top of the parse stack (or we are in the left-hand side's subroutine in a recursive descent parser) and we see the token coming up in the input. This sort of ambiguity is known as a predict-predict conflict; it can arise either because the same token can begin