FIRST and FOLLOW

- The construction of a predictive parser is aided by two functions associated with a grammar G. These functions, FIRST and FOLLOW, allow us to fill in the entries of a predictive parsing table for G, whenever possible. Sets of tokens yields by FOLLOW function can also be used as synchronizing tokens during panic-mode error recovery.

- If α is any string of grammar symbols, let FIRST(α) be the set of terminals that begin the strings derived from α. If $\alpha \Rightarrow^{*} \varepsilon$, then ε is also in FIRST(α).

Define FOLLOW(A), for nonterminal A, to be the set of terminals a that can appear immediately to the right of A in some sentential form, that is, the set of terminals a such that there exists a derivation of the form $S \stackrel{*}{\Rightarrow} \alpha A a \beta$ for some α and β. Note that there may, at some time during the derivation, have been symbols between A and a, but if so, they derived ε and disappeared. If A can be the rightmost symbol in some sentential form, then $ is in FOLLOW(A).

- To compute FIRST(X) for all grammar symbols X, apply the following rules until no more terminals or ε can be added to any FIRST set.

1. If X is terminal, then FIRST(X) is {X}.
2. If $X \rightarrow \varepsilon$ is a production, then add ε to FIRST(X).
3. If X is nonterminal and $X \rightarrow Y_1Y_2\ldots Y_k$ is a production, then place a in FIRST(X) if for some i, a is in FIRST(Y_i), and ε is in all of FIRST(Y_1),…, FIRST(Y_{i-1}); that is, $Y_1\ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$. If ε is in FIRST($Y_j$) for all j=1,2,…, k, then add ε to FIRST(X). For example, everything in FIRST(Y_1) is surely in FIRST(X). If Y_1 does not derive ε, then we add nothing more to FIRST(X), but if $Y_1 \overset{*}{\Rightarrow} \varepsilon$, then we add FIRST($Y_2$) and so on.

- Now, we can compute FIRST for any string $X_1X_2…X_n$ as follows. Add to FIRST($X_1X_2…X_n$) all the non-ε symbols of FIRST(X_1). Also add the non-ε symbols of FIRST(X_2) if ε is in FIRST(X_1), the non-ε symbols of FIRST(X_3) if ε is in both FIRST(X_1) and FIRST(X_2), and so on. Finally, add ε to FIRST($X_1X_2…X_n$) if, for all i, FIRST(X_i) contains ε.

- To compute FOLLOW(B) for all nonterminals B, apply the following rules until nothing can be added to any FOLLOW set.

1. Place \$ in FOLLOW(S), where S is the start symbol and \$ is the input right endmarker. (*!*)

2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) except for ε is placed in FOLLOW(B).

3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where FIRST(β) contains ε (i.e., $\beta \Rightarrow^* \varepsilon$), then everything in FOLLOW(A) is in FOLLOW(B).

- **Example 4.17.** Consider again grammar (4.11), repeated below:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E) \mid id$$

- Then:

$$FIRST(E) = FIRST(T) = FIRST(F) = \{(, id\}.$$
$$FIRST(E') = \{+, \varepsilon\}$$
$$FIRST(T') = \{*, \varepsilon\}$$
$$FOLLOW(E) = FOLLOW(E') = \{), \$\}$$
$$FOLLOW(T) = FOLLOW(T') = \{+,), \$\}$$
$$FOLLOW(F) = \{+, *,), \$\}$$

- For example, id and left parenthesis are added to FIRST(F) by rule (3) in the definition of FIRST with i=1 in each case, since FIRST(id) = {id} and FIRST('(') = { (} by rule (1). Then by rule (3) with i=1, the production $T \rightarrow FT'$ implies that id and left parenthesis are in FIRST(T) as well. As another example, ε is in FIRST(E') by rule (2).

- To compute FOLLOW sets, we put $ in FOLLOW(E) by rule (1) for FOLLOW. By rule (2) applied to production $F \rightarrow (E)$, the right parenthesis is also in FOLLOW(E). By rule (3) applied to production $E \rightarrow TE'$, $ and right parenthesis are in FOLLOW(E'). Since $E' \underset{*}{\Rightarrow} \varepsilon$, they are also in FOLLOW(T). For a last example of how the FOLLOW rules are applied, the production $E \rightarrow TE'$ implies, by rule (2), that everything other than ε in FIRST(E') must be placed in FOLLOW(T). We have already seen that $ is in FOLLOW(T).

- **Example 4.18.** Let us apply Algorithm to grammar (4.11). Since FIRST(TE') = FIRST(T) = { **(** , **id**}, production $E \rightarrow TE'$ cause M[E, ( ] and M[E, id] to acquire the entry $E \rightarrow TE'$.

- Production $E' \rightarrow +TE'$ cause M[E', +] to acquire $E' \rightarrow +TE'$. Production $E' \rightarrow \varepsilon$ cause M[E',)] and M[E', $] to acquire $E' \rightarrow \varepsilon$ since FOLLOW(E') ={), $}

- The parsing table produced by Algorithm 4.4 for grammar (4.11) was shown in Fig.4.15.

Construction of Predictive Parsing Table

- The following algorithm can be used to construct a predictive parsing table for a grammar G. The idea behind the algorithm is the following. Suppose $A \rightarrow \alpha$ is a production with a in FIRST(α). Then only parser will expand A by α when the current input symbol is a. The only complication occurs when $\alpha = \varepsilon$ or $\alpha \stackrel{*}{\Rightarrow} \varepsilon$. In this case, we should again expand A by α if the current input symbol is in FOLLOW(A), or the \$ on the input has been reached and \$ is in FOLLOW(A).

- **Algorithm 4.4** Construction of a predictive parsing table.

  Input. Grammar G.

  Output. Parsing table M.

  Method.

  1. For each production $A \rightarrow \alpha$ of the grammar, do steps 2 and 3.
  2. For each terminal a in FIRST(α), add $A \rightarrow \alpha$ to M[A, a].
  3. If ε is in FIRST(α), add $A \rightarrow \alpha$ to M[A, b] for each terminal b in FOLLOW(A). If ε is in FIRST(α) and $ is in FOLLOW(A), add $A \rightarrow \alpha$ to M[A, $].
  4. Make each undefined entry of M be error.

NONTER-MINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

Fig. 4.15. Parsing table M for grammar (4.11)

STACK	INPUT	OUTPUT
\$E	id + id * id\$	
\$E'T	id + id * id\$	$E \rightarrow TE'$
\$E'T'F	id + id * id\$	$T \rightarrow FT'$
\$E'T'id	id + id * id\$	$F \rightarrow id$
\$E'T'	+ id * id\$	
\$E'	+ id * id\$	$T' \rightarrow \varepsilon$
\$E'T+	+ id * id\$	$E' \rightarrow +TE'$
\$E'T	id * id\$	
\$E'T'F	id * id\$	$T \rightarrow FT'$
\$E'T'id	id * id\$	$F \rightarrow id$
\$E'T'	* id\$	
\$E'T'F*	* id\$	$T' \rightarrow *FT'$
\$E'T'F	id\$	
\$E'T'id	id\$	$F \rightarrow id$
\$E'T'	\$	
\$E'	\$	$T' \rightarrow \varepsilon$
\$	\$	$E' \rightarrow \varepsilon$

Fig.4.16 Moves made by predictive parser on input id+id*id

```
Set ip to point to the first symbol of w$;
repeat
      let X be the top stack symbol and a the symbol
      pointed to by ip;
      if X is a terminal or $ then
                if X=a then
                          pop X from the stack and advance ip
                else error()
      else  /*X is a nonterminal */
                if M[X, a] = X → Y1Y2…Yk then begin
                          pop X from the stack;
                          push Yk, Yk-1, …, Y1 onto the stack,
with
                          Y1 on top;
                          output the production X → Y1Y2…Yk
                end
                else error()
until X = $  /* stack is empty */
```

Fig.4.14. Predictive parsing program.

- **Example.** Consider the following simple grammar G1. Let us compute $FIRST_k$ and $FOLLOW_k$ for its nonterminal, for k=1:

1. $G \rightarrow E \perp$
2. $E \rightarrow TE'$
3. $E' \rightarrow +E$
4. $E' \rightarrow \varepsilon$
5. $T \rightarrow FT'$
6. $T' \rightarrow *T$
7. $T' \rightarrow \varepsilon$
8. $F \rightarrow (E)$
9. $F \rightarrow a$

- $FIRST_1(F) = \{ (, a\}$ from productions 8 and 9
- $FIRST_1 (T') = \{*, \varepsilon\}$ from productions 6 and 7
- $FIRST_1 (T)=FIRST_1 (FT')=FIRST_1 (F\ FIRST_1 (T'))=\{(, a\}$
- $FIRST_1 (E') = \{+, \varepsilon\}$ from productions 3 and 4
- $FIRST_1 (E) = FIRST_1 (TE') = \{ (, a\}$
- $FIRST_1 (G) = FIRST_1 (E\perp) = \{ (,a\}$
- $FOLLOW_1 (E) = \{\perp,)\} \cup FOLLOW_1 (E')$ from productions 1, 3, and 8
- $FOLLOW_1 (G) = \{ \$ \}$ using property (6)
- $FOLLOW_1 (E') = FOLLOW_1 (E)$ from production 2
- $FOLLOW_1 (T) = FIRST_1 (E'FOLLOW1(E)) \cup FOLLOW_1 (T')$ from productions 2 and 6
- $FOLLOW_1 (T') = FOLLOW_1 (T)$ from production 5
- $FOLLOW_1 (F) = FIRST_1 (T'\ FOLLOW_1 (T))$ from production 5

- By using these relations repeatedly, we obtain the following table of FOLLOW1.

	1	2	3	4	5	6
G	$					
E	⊥	)				
E’			⊥	)		
T			⊥	)	+	
T’			⊥	)	+	
F			⊥	)	+	*

- Columns 1 and 2 are the contents of $FOLLOW_1$ (G) and $FOLLOW_1$ (E) known directly from the productions. The remaining columns are determined by inference from these and the $FOLLOW_1$ and $FIRST_1$ relations given.

```
Set R[X] = ∅ for all nonterminal X in G;
Repeat                    (* First *)
    For every nonterminal X in G do begin
        For every production X → ω do begin
            Let x1 x2 … xr = ω;
            rx := 1
            more := true
            while more do begin
                if rx > r then begin
                    R[X] := R[X] + [ ε ];
                    more := false
                    end
                else if is_terminal(x_rx) then begin
                    R[X] := R[X] + [ x_rx ];
                    more := false
                    end
```

```
                else begin
                   R[X] := R[X] + (R[x_rx] – [ ε ])       (*!*)
                   if not(nullable(x_rx) then more := false
                   end
                rx := rx +1;
                end {while}
          end {for}
     end; {for}
until no member of R[X] has been augmented;
```

For all tokens X in G do F[X] := [];
Let S be the start token of G;
F[S] := [$]; **(* Follow *)**
repeat
for every token X in G do
if not(X in [ε,$]) then begin
for (every production $Z \rightarrow \omega$ such that
X appears in ω) do
for (every appearance of X in ω) do begin
let $\omega = \alpha\ X\ b_1 b_2 \ldots b_r$;
{where $\alpha \in (N \cup \Sigma)$;
and $b_i \in N$ for $1 \leq i \leq r$ }
let p := 1; { p = position in $b_1 b_2 \ldots b_r$ }
let more := true;

```
                while more do begin
                        if p ≥ r then begin
                                F[X] := F[X] + F[Z];
                                more := false;
                                end
                        else begin
                          F[X] := F[X] + (FIRST[bp] – [ ε ]);
                                If nullable(bp) then
                                        p := p + 1
                                else more:= false;
                        end {if}
                end {while}
        end {for}
        end {if}
until (no F[S] has been augmented)
```