Left Recursion: (2.4.5 & 4.3.3)

expr -> expr + term | term

A -> Aα | B

A -> BA’

A’ -> ε | αA’

E -> E + T | T

Left Factoring: (4.3.4)

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt
\end{array}
$$

A -> αB | αC

A -> αA’

A’ -> B | C

$$
\begin{array}{l}
S \rightarrow i\ E\ t\ S \ \mid\ i\ E\ t\ S\ e\ S\ \mid\ a \\
E \rightarrow b
\end{array}
$$

FIRST-FOLLOW (4.4.2)

Define $FIRST(\alpha)$, where α is any string of grammar symbols, to be the set of terminals that begin strings derived from α. If $\alpha \overset{*}{\Rightarrow} \epsilon$, then ϵ is also in $\text{FIRST}(\alpha)$. For example, in Fig. 4.15, $A \overset{*}{\Rightarrow} c\gamma$, so c is in $\text{FIRST}(A)$.

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

$$
\begin{array}{lcl}
E & \to & T\ E' \\
E' & \to & +\ T\ E' \mid \epsilon \\
T & \to & F\ T' \\
T' & \to & *\ F\ T' \mid \epsilon \\
F & \to & (\ E\ ) \mid \textbf{id}
\end{array}
$$

To compute $\text{FOLLOW}(A)$ for all nonterminals A, apply the following rules until nothing can be added to any FOLLOW set.

1. Place \$ in $\text{FOLLOW}(S)$, where S is the start symbol, and \$ is the input right endmarker.
2. If there is a production $A \to \alpha B \beta$, then everything in $\text{FIRST}(\beta)$ except ϵ is in $\text{FOLLOW}(B)$.
3. If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$, where $\text{FIRST}(\beta)$ contains ϵ, then everything in $\text{FOLLOW}(A)$ is in $\text{FOLLOW}(B)$.

LL(1) Grammars:

Predictive parsers, that is, recursive-descent parsers needing no backtracking, can be constructed for a class of grammars called LL(1). The first "L" in LL(1) stands for scanning the input from left to right, the second "L" for producing a leftmost derivation, and the "1" for using one input symbol of lookahead at each step to make parsing action decisions.

The class of LL(1) grammars is rich enough to cover most programming constructs, although care is needed in writing a suitable grammar for the source language. For example, no left-recursive or ambiguous grammar can be LL(1).

A grammar G is LL(1) if and only if whenever $A \to \alpha \mid \beta$ are two distinct productions of G, the following conditions hold:

1. For no terminal a do both α and β derive strings beginning with a.

2. At most one of α and β can derive the empty string.

3. If $\beta \stackrel{*}{\Rightarrow} \epsilon$, then α does not derive any string beginning with a terminal in FOLLOW(A). Likewise, if $\alpha \stackrel{*}{\Rightarrow} \epsilon$, then β does not derive any string beginning with a terminal in FOLLOW(A).

The first two conditions are equivalent to the statement that FIRST(α) and FIRST(β) are disjoint sets. The third condition is equivalent to stating that if ϵ is in FIRST(β), then FIRST(α) and FOLLOW(A) are disjoint sets, and likewise if ϵ is in FIRST(α).