Let $X$ be a grammar symbol, then $\text{First}(X)$:

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 $\epsilon$ is in all of $\text{First}(Y_1), \ldots, \text{First}(Y_{i-1})$; that is, $Y_1 \cdots Y_{i-1} \Rightarrow \epsilon$. If $\epsilon$ is in $\text{First}(Y_j)$ for all $j = 1, 2, \ldots, k$, then add $\epsilon$ to $\text{First}(X)$. For example, everything in $\text{First}(Y_1)$ is surely in $\text{First}(X)$. If $Y_1$ does not derive $\epsilon$, then we add nothing more to $\text{First}(X)$, but if $Y_1 \Rightarrow \epsilon$, then we add $\text{First}(Y_2)$, and so on.

3. If $X \to \epsilon$ is a production, then add $\epsilon$ to $\text{First}(X)$. 

$$
\begin{align*}
E & \to E + T \mid T \\
T & \to T * F \mid F \\
F & \to (E) \mid \text{id} \\
E' & \to T E' \\
E' & \to + T E' \mid \epsilon \\
T' & \to F T' \\
T' & \to * F T' \mid \epsilon \\
F & \to (E) \mid \text{id}
\end{align*}
$$
Follow(X):

1. Place $\$ \in \text{FOLLOW}(S)$, where $S$ is the start symbol, and $\$ \in \text{FIRST}(S)$ is the input right endmarker.

2. If there is a production $A \rightarrow \alpha B \beta$, then everything in $\text{FIRST}(\beta)$ except $\epsilon$ is in $\text{FOLLOW}(B)$.

3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$, where $\text{FIRST}(\beta)$ contains $\epsilon$, then everything in $\text{FOLLOW}(A)$ is in $\text{FOLLOW}(B)$.

For the Example:

$$
\begin{align*}
E & \rightarrow \ T \ E' \\
E' & \rightarrow \ + \ T \ E' \mid \epsilon \\
T & \rightarrow \ F \ T' \\
T' & \rightarrow \ * \ F \ T' \mid \epsilon \\
F & \rightarrow \ ( \ E ) \mid \text{id}
\end{align*}
$$

1. $\text{FIRST}(F) = \text{FIRST}(T) = \text{FIRST}(E) = \{(, \ \text{id}\}$. To see why, note that the two productions for $F$ have bodies that start with these two terminal symbols, $\text{id}$ and the left parenthesis. $T$ has only one production, and its body starts with $F$. Since $F$ does not derive $\epsilon$, $\text{FIRST}(T)$ must be the same as $\text{FIRST}(F)$. The same argument covers $\text{FIRST}(E)$.

2. $\text{FIRST}(E') = \{+, \ \epsilon\}$. The reason is that one of the two productions for $E'$ has a body that begins with terminal $+$, and the other's body is $\epsilon$. Whenever a nonterminal derives $\epsilon$, we place $\epsilon$ in $\text{FIRST}$ for that nonterminal.

3. $\text{FIRST}(T') = \{*, \epsilon\}$. The reasoning is analogous to that for $\text{FIRST}(E')$. 
Now for the Follow:

\[
\begin{align*}
E & \rightarrow T \ E' \\
E' & \rightarrow + T \ E' | \epsilon \\
T & \rightarrow F T' \\
T' & \rightarrow * F T' | \epsilon \\
F & \rightarrow ( E ) | \text{id}
\end{align*}
\]

4. \( \text{FOLLOW}(E) = \text{FOLLOW}(E') = \{[,], \text{\$}\} \). Since \( E \) is the start symbol, \( \text{FOLLOW}(E) \) must contain \( \text{\$} \). The production body \( (E) \) explains why the right parenthesis is in \( \text{FOLLOW}(E) \). For \( E' \), note that this nonterminal appears only at the ends of bodies of \( E \)-productions. Thus, \( \text{FOLLOW}(E') \) must be the same as \( \text{FOLLOW}(E) \).

5. \( \text{FOLLOW}(T) = \text{FOLLOW}(T') = \{+, [,], \text{\$}\} \). Notice that \( T \) appears in bodies only followed by \( E' \). Thus, everything except \( \epsilon \) that is in \( \text{FIRST}(E') \) must be in \( \text{FOLLOW}(T) \); that explains the symbol \( + \). However, since \( \text{FIRST}(E') \) contains \( \epsilon \) (i.e., \( E' \rightarrow^* \epsilon \)), and \( E' \) is the entire string following \( T \) in the bodies of the \( E \)-productions, everything in \( \text{FOLLOW}(E) \) must also be in \( \text{FOLLOW}(T) \). That explains the symbols \( \text{\$} \) and the right parenthesis. As for \( T' \), since it appears only at the ends of the \( T \)-productions, it must be that \( \text{FOLLOW}(T') = \text{FOLLOW}(T) \).

6. \( \text{FOLLOW}(F) = \{+, *, [,], \text{\$}\} \). The reasoning is analogous to that for \( T \) in point (5).
LL(1) Parsing Table

**INPUT:** Grammar $G$.

**OUTPUT:** Parsing table $M$.

**METHOD:** For each production $A \to \alpha$ of the grammar, do the following:

1. For each terminal $a$ in $\text{FIRST}(\alpha)$, add $A \to \alpha$ to $M[A, a]$.

2. If $\epsilon$ is in $\text{FIRST}(\alpha)$, then for each terminal $b$ in $\text{FOLLOW}(A)$, add $A \to \alpha$ to $M[A, b]$. If $\epsilon$ is in $\text{FIRST}(\alpha)$ and $\$$ is in $\text{FOLLOW}(A)$, add $A \to \alpha$ to $M[A, \$$]$ as well.

---

If, after performing the above, there is no production at all in $M[A, a]$, then set $M[A, a]$ to error (which we normally represent by an empty entry in the table).

<table>
<thead>
<tr>
<th>NON-TERMINAL</th>
<th>id</th>
<th>+</th>
<th>*</th>
<th>(</th>
<th>)</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>$E \to TE'$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E'$</td>
<td>$E' \to +TE'$</td>
<td>$E' \to \epsilon$</td>
<td>$E' \to \epsilon$</td>
<td>$E' \to \epsilon$</td>
<td>$E' \to \epsilon$</td>
<td>$E' \to \epsilon$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T \to FT'$</td>
<td></td>
<td>$T' \to \epsilon$</td>
<td>$T' \to \epsilon$</td>
<td>$T' \to \epsilon$</td>
<td>$T' \to \epsilon$</td>
</tr>
<tr>
<td>$T'$</td>
<td>$T' \to *FT'$</td>
<td></td>
<td></td>
<td>$T' \to \epsilon$</td>
<td>$T' \to \epsilon$</td>
<td>$T' \to \epsilon$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F \to id$</td>
<td>$F \to (E)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.19: Model of a table-driven predictive parser

**Method:** Initially, the parser is in a configuration with $w\$ in the input buffer and the start symbol $S$ of $G$ on top of the stack, above $\$. The program in Fig. 4.20 uses the predictive parsing table $M$ to produce a predictive parse for the input. □

```python
let a be the first symbol of $w$;
let X be the top stack symbol;
while (X $\neq$ $\$) { /* stack is not empty */
    if (X = a) pop the stack and let a be the next symbol of $w$;
    else if (X is a terminal) error();
    else if (M[X, a] is an error entry) error();
    else if (M[X, a] = X $\rightarrow$ Y$_1$Y$_2$ $\ldots$ Y$_k$) {
        output the production X $\rightarrow$ Y$_1$Y$_2$ $\ldots$ Y$_k$;
        pop the stack;
        push Y$_k$, Y$_{k-1}$, $\ldots$, Y$_1$ onto the stack, with Y$_1$ on top;
    }
    let X be the top stack symbol;
}
<table>
<thead>
<tr>
<th>Matched</th>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>id + id * id$</td>
<td>$TE'$</td>
<td>output $E \rightarrow TE'$</td>
</tr>
<tr>
<td>$TE'$</td>
<td>id + id * id$</td>
<td>$FT'E'$</td>
<td>output $T \rightarrow FT'$</td>
</tr>
<tr>
<td>$FT'E'$</td>
<td>id + id * id$</td>
<td>id $T'E'$</td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id</td>
<td>$T'E'$</td>
<td>+ id * id$</td>
<td>match id</td>
</tr>
<tr>
<td>id</td>
<td>$E'$</td>
<td>+ id * id$</td>
<td>output $T' \rightarrow \epsilon$</td>
</tr>
<tr>
<td>id</td>
<td>+ $TE'$</td>
<td>+ id * id$</td>
<td>output $E' \rightarrow + TE'$</td>
</tr>
<tr>
<td>id +</td>
<td>$TE'$</td>
<td>id * id$</td>
<td>match +</td>
</tr>
<tr>
<td>id +</td>
<td>$FT'E'$</td>
<td>id * id$</td>
<td>output $T \rightarrow FT'$</td>
</tr>
<tr>
<td>id +</td>
<td>id $T'E'$</td>
<td>id * id$</td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id + id</td>
<td>$T'E'$</td>
<td>* id$</td>
<td>match *</td>
</tr>
<tr>
<td>id + id</td>
<td>* $FT'E'$</td>
<td>* id$</td>
<td>output $T' \rightarrow * FT'$</td>
</tr>
<tr>
<td>id + id</td>
<td>* $FT'E'$</td>
<td>id$</td>
<td>match *</td>
</tr>
<tr>
<td>id + id</td>
<td>* id $T'E'$</td>
<td>id$</td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id + id</td>
<td>* id $T'E'$</td>
<td>id$</td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id + id</td>
<td>* id $E'$</td>
<td>$$$</td>
<td>output $T' \rightarrow \epsilon$</td>
</tr>
<tr>
<td>id + id</td>
<td>* id $E'$</td>
<td>$$$</td>
<td>output $E' \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

Figure 4.21: Moves made by a predictive parser on input id + id * id