Construction of Predictive Parse Table

- After finding the FIRST and FOLLOW, we can construct predictive parse table \( M[A, a] \), where \( A_i \)'s are non-terminal of the given grammar and \( a \)'s are terminals including $$. The entries in this two-dimensional table are made according to following rule.

- Here ‘A’ represents row and ‘a’ represents columns.
Construction of Predictive Parse Table

Algorithm for construction of Predictive Parse Table

1. Compare each production of the grammar with $A \rightarrow \alpha$, i.e. everything on right side is taken as $\alpha$, apply step 2 and 3, for each of them

2. Find FIRST($\alpha$), for each $a$ in FIRST($\alpha$)
   
   \[ \text{add } M[A, a] = A \rightarrow \alpha \]
   
   which means for each $a$ in FIRST($\alpha$), corresponding to non-terminal $A$ and terminal $a$, the current production $A \rightarrow \alpha$ is added to table.

3. If $\epsilon$ is in FIRST($\alpha$), then
   
   \[ \text{add, } M[A, a] = A \rightarrow \alpha \text{ for each } b \text{ in FOLLOW}(A) \]
   
   If $\epsilon$ is in FIRST($\alpha$) and $\$$ is in FOLLOW($A$) then
   
   \[ \text{add, } M[A,\$$] = A \rightarrow \alpha \]

4. Make each undefined entry of $M$, as error.
Construction of Predictive Parse Table

Consider the grammar

- $E \rightarrow E + T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow (E) \mid \text{id}$

First we check left-recursion in grammar, because this grammar is left recursive, so we will remove left-recursion.

- $E \rightarrow TE'$
- $E' \rightarrow +TE' \mid \varepsilon$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \varepsilon$
- $F \rightarrow (E) \mid \text{id}$
Construction of Predictive Parse Table

- Its FIRST is
  - \( \text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{(, \text{ id}\} \)
  - \( \text{FIRST}(E') = \{+, \varepsilon\} \)
  - \( \text{FIRST}(T') = \{*, \varepsilon\} \)

- Its FOLLOW is
  - \( \text{FOLLOW}(E) = \text{FOLLOW}(E') = \{), \$\} \)
  - \( \text{FOLLOW}(T) = \text{FOLLOW}(T') = \{+, ), \$\} \)
  - \( \text{FOLLOW}(F) = \{+, *, ), \$\} \)

- Now for constructing predictive parse table, comparing each production with \( A \rightarrow \alpha \)
Entries for Predictive Parser table

- \( M[E,()] = E \rightarrow TE' \)
- \( M[E,id] = E \rightarrow TE' \)
- \( M[E',+] = E' \rightarrow +TE' \)
- \( M[E',\text{,}id] = E' \rightarrow \varepsilon, \quad M[E',\$] = E' \rightarrow \varepsilon \)
- \( M[T,()] = E \rightarrow FT' \)
- \( M[T,id] = E \rightarrow FT' \)
- \( M[T',\*] = T' \rightarrow *FT' \)
- \( M[T',+] = T' \rightarrow \varepsilon, \quad M[T',\text{,}id] = T' \rightarrow \varepsilon, \quad M[T',\$] = T' \rightarrow \varepsilon \)
- \( M[F,()] = F \rightarrow (E) \)
- \( M[F,id] = F \rightarrow id \)
Predictive Parsing

- We have constructed the predictive parse table, now our next step is how to use this table to declare that an input string has been accepted by the grammar or not, for this first we see one algorithm for this purpose.
**Algo. for predictive parsing**

Push start symbol into stack  
Repeat  
 Begin  
 Make X to be the top stack symbol and a the next symbol  
 If X is a terminal on $ then  
 If X=a, then  
 Pop X from stack and remove a from input  
 else  
 Error()  
 else /* i.e. X is a non-terminal */  

If \( M[X,a] = X \rightarrow Y_1,Y_2,..Y_k \) then  
 Begin  
 Pop X from stack  
 Push \( Y_k Y_{k-1} -----Y_3 Y_2 Y_1 \) onto stack such that \( Y_1 \) is on top  
 End  
 Else  
 error()  
 End  
 Until X = $ /* i.e. stack empties */
Predictive Parsing

- This algorithm is quite simple. First push start symbol onto top of stack, call it X. Now look for the given input string, in the If condition, if they match Pop X and remove current input a.

- If it is a non-terminal then Pop this non-terminal from stack and look for production corresponding to this non-terminal X and current input symbol a i.e. M[X,a], then after popping X, push the right hand side of production in reverse order into stack \((Y_k \ Y_{k-1} \ldots \ Y_3 \ Y_2 \ Y_1)\). Repeat this process till \(X = \$\). Note that we always call the top stack symbol as X.

- Now we will see this parsing algorithm for the input string id+id*id
### Predictive Parsing

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input (a)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>id+id*id$</td>
<td>--</td>
</tr>
<tr>
<td>$E/T$</td>
<td>id+id*id$</td>
<td>E $\rightarrow$ TE' from M[E, id] - [Push in reverse order]</td>
</tr>
<tr>
<td>$E/T/F$</td>
<td>id+id*id$</td>
<td>T $\rightarrow$ FT' from M[T, id] - [Push in reverse order]</td>
</tr>
<tr>
<td>$E/T/id$</td>
<td>id+id*id$</td>
<td>F $\rightarrow$ id from M[F, id] [Push]</td>
</tr>
</tbody>
</table>
| $E/T'$ | +id*id$ | X=id, and matches with a=id
So, Pop X and remove a |
| $E'$ | +id*id$ | T' $\rightarrow$ $\epsilon$ from M[T',+] [Push] |
| $E/T+$ | +id*id$ | E' $\rightarrow$ +TE' from M[E',+] [Push in reverse order] |
| $E/T$ | id*id$ | X=+, and matches with a=+
So, Pop X and remove a |
| $E/T/F$ | id*id$ | E $\rightarrow$ FT' from M[T,id] [Push in reverse order] |
| $E/T/id$ | id*id$ | F $\rightarrow$ id from M[F, id] [Push] |
| $E/T'$ | *id$ | X=id, and matches with a=id
So, Pop X and remove a |
## Predictive Parsing

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input (a)</th>
<th>Output</th>
</tr>
</thead>
</table>
| $E/T'$ | *id$      | X=id, and matches with a=id  
|        |           | So, Pop X and remove a       |
| $E/T/F^*$ | *id$   | T' $\rightarrow$ *FT' from M[T',*] [Push in reverse order] |
| $E/T/F$ | id$      | X=*, and matches with a=*  
|        |           | So, Pop X and remove a       |
| $E/T/id$ | id$   | F $\rightarrow$ id from M[F, id] [Push] |
| $E/T'$ | $        | X=id, and matches with a=id  
|        |           | So, Pop X and remove a       |
| $E/'$  | $        | T' $\rightarrow$ $\varepsilon$ from M[T',$] [Push] |
| $      | $        | E' $\rightarrow$ $\varepsilon$ from M[E',$] [Push] |

So, $ matches and the input string “id+id*id” is accepted by the grammar.