# Syntax Analysis- III

# Predictive Parsers Design

## Introduction:

Predictive Parsers use a predictive parsing table to determine which production rule is to be used to expand a non-terminal depending on the input symbol. It has to be **LL(1) grammar** that is its *left most derivation* can be found by scanning *1 symbol* at a time.

A predictive parser is constituted of four parts:

i) An input string ending with $

ii) A stack

iii) a predictive parsing table

iv) An output

- **Stack-**
  The stack contains the input that has been scanned but not yet processed. The stack contains a $ at the end to demark the end of stack pointer. The stack is initialized by the start symbol.
- **Predictive Parsing Table:**
  It is a 2-D array which has columns corresponding to all terminals of the grammar and rows corresponding to all non-terminals of the grammar. A cell in the table is denoted as **M[A , a]** and contains either a production rule or an error entry.

## Algorithm for Predictive Parsing :-

Let a be the current input symbol scanned and X be the top element of stack. A Predictive Parser technique works in the following way:

**1) If X=a=$ (string is valid and accepted)**

**2) If X=a≠$ ( if a is terminal, pop it from stack and add to output)**

**3) If X= non-terminal (Go to entry in parsing table M[X,a] if a production rule exists, pop X, Push elements of production rule into stack otherwise call error handler routine )**

## How to generate Predictive Parsing Table

Compute value of two functions for all terminals and non-terminals of the grammar:

**1) FIRST( )**

**2) FOLLOW( )**

## Algorithm:

### 1) FIRST( ):-

*FIRST(n) returns the set of all terminals which can be the start of grammar symbol 'n' ( n can be a terminal or non-terminal)*

1. If n is a terminal, **First(n) = {n}**
2. If n is a non-terminal, **First(n)** is determined as follows:-
   a) If **n → aα|bβ|cγ** *where a,b,c are terminals,*
      **First(n) ={a,b,c}**
   b) If **n → Aα|Bα** *where α≠ϵ*
      **First(n) = First(A) U First(B)** *except ϵ*
   c) If **n→ ϵ or n→Aα** *where α= ϵ and First(A) contains ϵ*
      **First(n)={ ϵ }** ; *add ϵ to First(n)*

### 2) Follow( ):-

*Follow(n) returns the set of all terminals that can follow the non-terminal 'n'. Follow() can only be computed for non- terminals.*

1) Add **$** to follow of start symbol
2) For production rule **A→αBβ**
   add **First(β)** to **Follow(B)** *except ϵ*
3) For rule **A→ αB** or **A→ αBβ** *where First(β)= ϵ*
   add **Follow(A)** to **Follow(B)**

### Construction of Predictive 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 FIRST($A$), add $A \to \alpha$ to $M[A, a]$.
2. If $\epsilon$ is in FIRST($\alpha$), then for each terminal $b$ in FOLLOW($A$), add $A \to \alpha$ to $M[A, b]$. If $\epsilon$ is in FIRST($\alpha$) and \$ is in FOLLOW($A$), add $A \to \alpha$ to $M[A, \$]$ as well.

Ex: Create the predictive parsing table for the following grammar. Also write stack contents and draw syntax tree for given input string.

**E→TE'**
**E'→+TE'| ϵ**
**T→FT'**
**T'→*FT'| ϵ**
**F→id|(ϵ)**

**Solution:**

**Step 1: Compute First() and Follow():**

**First(+)={+}**
**First(*)={*}**
**First(id)={id}**
**First( ( )={ ( }**
**First( ) )={ ) }**
**First(E)={ ( , id }**
**First(E')={ +, ϵ }**
**First(T)={ (, id }**
**First(T')={ * , ϵ }**
**First(F)={ ( , id }**

**Follow(E)={ $,) }**
**Follow(E')={ $, ) }**
**Follow(T)={ $, +, ( }**
**Follow(T')={ $, +, ( }**
**Follow(F)={ * , +, (, $ }**

**Step 2: Construction of parsing table**

| M | Id | + | * | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | E→TE' | | | E→TE' | | |
| E' | | E'→+TE' | | | E'→ϵ | E'→ϵ |
| T | T→FT' | | | T→FT' | | |
| T' | | T'→ ϵ | T'→*FT' | | T'→ ϵ | T'→ ϵ |
| F | F→id | | | F→(E) | | |