NON RECURSIVE PREDICTIVE PARSING / (LL1 PARSING)

This top-down parsing algorithm is of non-recursive type. In this type a parsing table is built. For LL(1) the first 'L' means. "The input is scanned from Left to right." The second 'L' means "It uses left most derivation for input String" and the number '1' in the input symbol means it uses only one input symbol (Look ahead) to predict the parsing process.

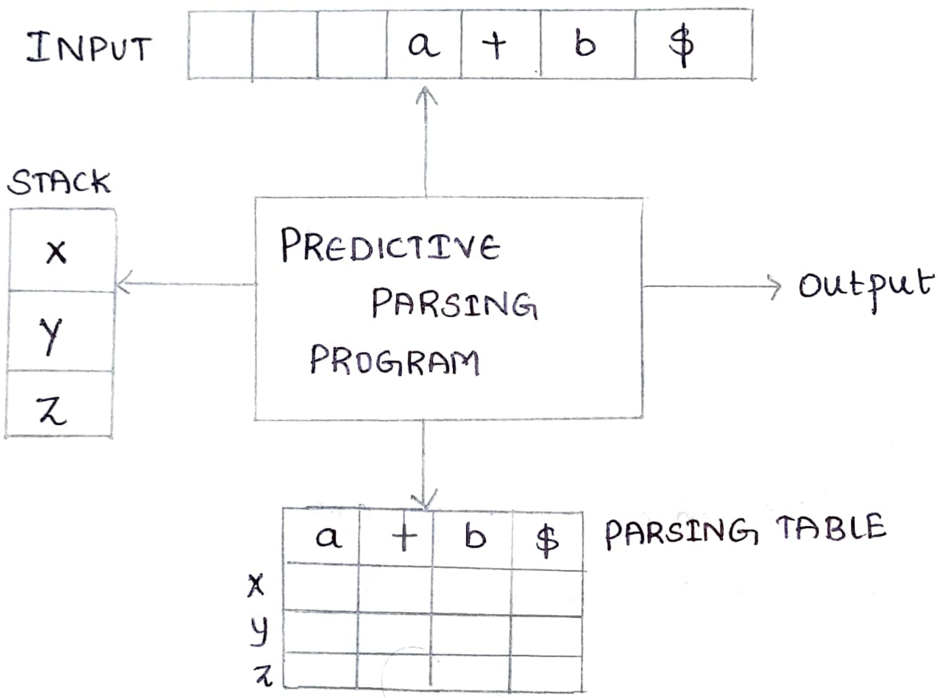

Fig:- Model of a table-driven predictive parser.

The data structures used by LL(1) are:

1. Input Buffer
2. Stack
3. Parsing table

1. Input Buffer: The LL(1) parser uses input buffer to store the input tokens

2. Stack:- The stack is used to hold the left sential form. The symbols in R.H.S of rule are pushed into the stack in reverse order i.e. from right to left. Thus use of stack makes this algorithm non-recursive.

3. Parsing table:- The table is basically a two dimensional array. The table has row for non-terminal and column for terminals.

Algorithm:-

Table driven predictive parsing.

```
Set ip to point to the first symbol of w
Set X to the top stack symbol.
while( X ≠ $) /* stack is not empty */
  {
  if (X is a) Pop the stack and advance ip;
  Else if (X is a terminal) error ();
  Else if (M [X,a] is an Error entry) error();
  Else if (M[X,a] = X → Y1 Y2 - - - Yk)
   {
  Output the production X → Y1 Y2 Y3 - - - Yk;
   Pop the stack
   Push Yk, Yk-1, - - -, Y1 onto the stack, with Y1 on top;
   }
   Set X to the top stack symbol;
}
```

Construction of predictive parser.

The construction of predictive LL(1) parser is based on two very important functions and those are,

FIRST and FOLLOW

For construction of predictive LL(1) parser we have to follow the following steps:-

1. Computation of FIRST and FOLLOW function.
2. Construct the predictive parsing table using FIRST and FOLLOW functions.
3. Parse the input string with the help of predictive parsing table.

FIRST Function :-

First (α) is a set of terminal symbols that are first symbols appearing at R.H.S in derivation of α.

Following are the rules used to compute the FIRST function.

1. If the terminal symbol a the FIRST(a) = {a}

   $X \rightarrow abc$

2. If there is a rule $X \rightarrow \epsilon$ then FIRST(X) = {ϵ}

3. For the rule $X \rightarrow AbC$

   $A \rightarrow mnq$

   First(X) = First(A) = First(m) = {m}

4. For the rule $A \rightarrow X_1 X_2 X_3 \dots X_K$

   First(A) = First(X_1) $\cup$ First(X_2) $\cup$ ..... First(X_K).

Consider the grammar shown below.

$E \rightarrow TE'$
$E' \rightarrow +TE' | \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' | \epsilon$
$F \rightarrow (E) | id$

$First(E) = First(T)$

$T \rightarrow FT'$

$First(T) = First(F)$

$F \rightarrow (E) | id$

$First(F) = First(() \cup First(id)$
$= \{(\} \cup \{id\}$
$= \{(, id\}$

$\therefore First(E) = First(T) = First(F) = \{(, id\}$

$First(E') = First(+) \cup First(\epsilon)$
$= \{+\} \cup \{\epsilon\}$
$= \{+ \epsilon\}$

$First(T') = First(*) \cup First(\epsilon)$
$= \{*, \epsilon\}$

FOLLOW Function

Follow(A) is defined as the set of terminal symbols that appear immediately to the right of A.

The rules for computing FOLLOW functions are as given below.

1. For the start symbol S place \$ in FOLLOW(S).
2. If there is a production $A \rightarrow \alpha B \beta$ then everything in First(β) without ϵ is to be placed in FOLLOW(B).

$$X \rightarrow aBc$$
$$A \rightarrow \alpha B \beta$$

$$\text{Follow}(B) = \text{First}(\beta)$$
$$= \text{First}(c) = \underline{\{c\}}$$

3. If there is a production $A \rightarrow \alpha B$, then, FOLLOW(B) = FOLLOW(A). That means everything in FOLLOW(A) is in FOLLOW(B).

$$X \rightarrow aB$$
$$A \rightarrow \alpha B$$
$$\text{FOLLOW}(B) = \text{FOLLOW}(X)$$

1) Construct FOLLOW Sets for the grammar

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

1] FOLLOW(E)

$A \rightarrow \alpha B \beta$

$F \rightarrow (E)$

Follow(B) = First(β)

Follow(E) = First())

= {) } = {), \$ }

2] FOLLOW(E')

$E \rightarrow TE'$

$A \rightarrow \alpha B$

Follow(E') = Follow(E)

= {), \$ }

3] FOLLOW(T)

$E \rightarrow TE'$

$A \rightarrow \alpha B \beta$

Follow(T) = First(E')

= { +, ϵ }

Follow(T) = First(E') − ϵ ∪ Follow(E')

= { +, ϵ } − ϵ ∪ {), \$ }

= { +,), \$ }

4] FOLLOW(T')

$T \rightarrow FT'$

$A \rightarrow \alpha B$

Follow(B) = Follow(A)

Follow(T') = Follow(T)

= { +,), \$ }

Follow(F)

$T \rightarrow FT'$

$A \rightarrow \alpha B \beta$

Follow(A) = First(β)

Follow(T) = First(T')

$= \{*, \epsilon\}$

$= \{*, \epsilon\} - \epsilon \cup$ Follow(T')

$= \{*, \epsilon\} - \epsilon \cup \{+,), \$\}$

$= \{*, +,), \$\}$

SYMBOL	FIRST	FOLLOW
E	{(, id}	{), $}
E'	{+, ε}	{), $}
T	{(, id}	{+,), $}
T'	{*, ε}	{+,), $}
F	{(, id}	{*, +,), $}

Algorithm for predictive parsing table.

For the rule $A \rightarrow \alpha$ of grammar G

1. For each a in First(α) create entry $M[A, a] = A \rightarrow \alpha$
   Where a is terminal symbol.

2. For ϵ in First(α) create entry $M[A, b] = A \rightarrow \alpha$
   Where b is the symbols from FOLLOW(A)

3. If ϵ is in FIRST(α) and \$ is in Follow(A) then create entry in the table $M[A, \$] = A \rightarrow \alpha$

1] Construct predictive parsing table for the following grammar and show the moves made by the predictive parser on the input id + id * id.

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow (E) \mid id$$

Consider the grammar,

$$E \rightarrow E+T \mid T$$

Map this grammar using the rule $\boxed{A \rightarrow A\alpha \mid \beta}$

$$\underbrace{E}_{A} \rightarrow \underbrace{E}_{A}\underbrace{+T}_{\alpha} \mid \underbrace{T}_{\beta}$$

$$\boxed{A \rightarrow \beta A'}$$

$$E \rightarrow TE'$$

$$\boxed{A' \rightarrow \alpha A' \mid \epsilon}$$

$$E' \rightarrow +TE' \mid \epsilon$$

Consider the grammar

$$T \rightarrow T*F \mid F$$

Map this grammar using the rule $A \rightarrow A\alpha \mid \beta$

$$T \rightarrow T*F \mid F$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \epsilon$$

∴ The grammar is

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

Constructing the FIRST and FOLLOW sets for the above grammar

FIRST sets:-

Consider the grammar $E \to TE'$

$First(E) = First(T)$

$T \to FT'$

$First(T) = First(F)$

$F \to (E) | id$

$First(F) = First(() \cup First(id)$

$= \{(\} \cup \{id\}$

$= \{(\; id\}$

$\therefore$ $First(E) = First(T) = First(F) = \{(, id\}$

consider the grammar

$E' \to +TE' | \epsilon$

$First(E') = First(+) \cup First(\epsilon)$

$= \{+\} \cup \{\epsilon\}$

$= \{+ \; \epsilon\}$

Consider the grammar

$T' \to *FT' | \epsilon$

$First(T') = First(*) \cup First(\epsilon)$

$= \{*\} \cup \{\epsilon\}$

$= \{* \; \epsilon\}$

FOLLOW Sets:-

Follow(E)

$A \to \alpha B \beta$

$F \to (E) | id$

$Follow(B) = First(\beta)$

$Follow(E) = First())$

$= \{)\} = \{), \$\}$ //

Follow(E')

$E \rightarrow TE'$

$A \rightarrow \alpha B$

Follow(B) = Follow(A)

Follow(E') = Follow(E)

= {), $ }

Follow(T)

$E \rightarrow TE'$

$A \rightarrow \alpha B \beta$

Follow(T) = First(E')

= { +, ∈ }

= { +, ∈ } − ∈ ∪ Follow(E')

= { +, ∈ } − ∈ ∪ {), $ }

= { +,), $ }

Follow(T')

$T \rightarrow FT'$

$A \rightarrow \alpha B \beta$

Follow(T') = Follow(T)

= { +,), $ }

Follow(F)

$T \rightarrow FT'$

$A \rightarrow \alpha B \beta$

Follow(F) = First(T')

= { *, ∈ }

= { *, ∈ } − ∈ ∪ Follow(T')

= { *, ∈ } − ∈ ∪ { +,), $ }

= { *, +,), $ }

SYMBOLS	FIRST	FOLLOW
E	{ (, id }	{), $ }
E'	{ +, ϵ }	{), $ }
T	{ (, id }	{ +,), $ }
T'	{ *, ϵ }	{ +,), $ }
F	{ (, id }	{ *, +,), $ }

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'		T'→ϵ	T'→*FT'			
F	F→id			F→(E)		

MATCHED	STACK	INPUT	ACTION
	E$	id+id*id$	
	TE'$	id+id*id$	E→TE'
	FT'E'$	id+id*id$	T→FT'
	idT'E'$	id+id*id$	F→id
id	T'E'$	+id*id$	matched id
id	E'$	+id*id$	T'→ε
	+TE'$	+id*id$	E'→+TE'
id+	TE'$	id*id$	Match +
id+	FT'E'$	id*id$	T→FT'
id+	idT'E'$	id*id$	F→id
id+id	T'E'$	*id$	match id
id+id	*FT'E'$	*id$	T'→*FT'
id+id*	FT'E'$	id$	match *
id+id*	idT'E'$	id$	F→id
id+id*id	T'E'$	$	match id
id+id*id	E'$	$	T'→ε
id+id*id	$	$	E'→ε