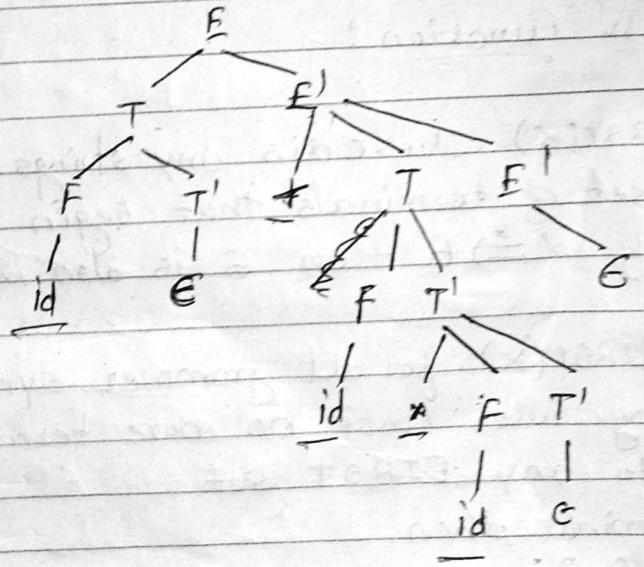
* Top-down parsing :-
unambiguous
left recursion elimination
left factoring
Top down follows leftmost derivation (like preorder)

① $E \to TE'$
$E' \to +TE' / \epsilon$
$T \to FT'$
$T' \to *FT' / \epsilon$
$F \to (E) / id$

id+id*id

$\Rightarrow$

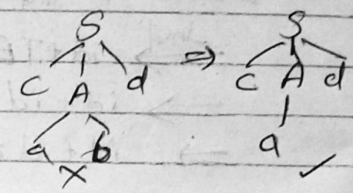
$E \xRightarrow{lmd} TF' \xRightarrow{lmd} FT'E' \xRightarrow{lmd} idT'E' \xRightarrow{lmd} idE'$
$\xRightarrow{lmd} id+TE' \Longrightarrow id+FT'E' \Rightarrow id+idT'E'$
$\Longrightarrow id+id*FT'E' \Rightarrow id+id*idT'E'$
$\Longrightarrow id+id*idE' \Rightarrow id+id*id$



* Recursive-Descent parsing :-
↳ with Backtracking
↳ without backtracking

```
void A(){
1) Choose an A-production, A → X1, X2 - - - Xk
2) For (i=1 to k) {
     if(Xi is a nonterminal)
          Call procedure Xi();
     else if(Xi equals the current i/p symbol a)
          advance the input to the next symbol;
     else
          /* an error has occured */
   }
}
```

Ex:- $S \to cAd$, $A \to ab/a$ $\quad X_1\ X_2\ X_3$ $\quad w = cad$



Imp

* FIRST & FOLLOW Function :-

* FIRST function :-

Define FIRST(α) where α is any strings 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 FIRST(α)

To compute FIRST(X) for all grammer symbol X apply the following rules until no more terminal or ϵ can be added to any FIRST set.

1) If X is a terminal then
   FIRST(X) = {X}

2) If $X \to \epsilon$ is production then add ϵ to FIRST of X
   FIRST(X) = {ϵ}

3)

{ϵ}

3) If X is non terminal & $X \to Y_1 Y_2 \ldots Y_K$ is a production for some $K \geq 1$ then place 'a' in ~~first~~ FIRST(X) if for some i, a is in FIRST(Y_i) & ϵ is in all of FIRST(Y_1), FIRST(Y_2) ... FIRST(Y_{i-1})
i.e. $Y_1, Y_2, \ldots Y_{i-1} \overset{*}{\Rightarrow} \epsilon$

4) ~~Compute FIRST~~ If ϵ is in FIRST(Y_j) $\forall j = 1, 2 \ldots K$ then add ϵ to FIRST(S)

* Compute FIRST for the following grammer.

(1)
$S \to aBDH$
$B \to cC$
$C \to bC \mid \epsilon$
$D \to EF$
$E \to g \mid \epsilon$
$F \to f \mid \epsilon$

$\Rightarrow$ FIRST(A) = {a}
FIRST(B) = {c}
FIRST(E) = {g, ϵ}
FIRST(C) = {b, ϵ}
FIRST(D) = {g, f, ϵ}
FIRST(F) = {f, ϵ}

(2)
$S \to A$
$A \to aB \mid Ad$
$B \to b$
$C \to g$

FIRST(S)
FIRST(A)
$\longrightarrow$ FIRST(B) = {b}
$\Rightarrow$ FIRST(C) = {g}

$\Rightarrow$
$S \to A$
$A \to aBA'$
$A' \to dA' \mid \epsilon$

FIRST(S) = {a}
FIRST(A) = {a}
FIRST(A') = {d, ϵ}

Imp

(3) $S \to (L) \mid a$

$L \to SL'$

$L' \to ,SL' \mid \epsilon$

$\Rightarrow$ FIRST(S) = {(, a}

FIRST(L) = {(, a}

FIRST(L') = {, , ε}

(4) $S \to AaAb \mid BbBa$

$A \to \epsilon$

$B \to \epsilon$

$\Rightarrow$ FIRST(S) = {a, b}

FIRST(A) = {ε}

FIRST(B) = {ε}

(5) $E \to TE'$

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

$T \to FT'$

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

$F \to (E) \mid id$

$\Rightarrow$ FIRST(E') = {+, ε}

FIRST(T') = {*, ε}

FIRST(F) = {(, id}

FIRST(T) = {(, id}

FIRST(E) = {(, id}

(6) $S \to ACB \mid CbB \mid Ba$

$A \to da \mid BC$

$B \to g \mid \epsilon$

$C \to h \mid \epsilon$

$\Rightarrow$ FIRST(B) = {g, ε}

FIRST(C) = {h, ε}

FIRST(A) = {d, g, h, ε}

FIRST(S) = {d, g, h, ε, a, b}

{First(A) − ε} ∪ {First(B) − ε} ∪ {First(C)}

{g, d, h, ε}

* FOLLOW Function:-

Define FOLLOW(A) for non-terminal 'A' to be the set of terminals 'a' that can appear immediately to the right of 'A' in some sentential form i.e.

$S \overset{*}{\Longrightarrow} \alpha A a \beta$

To compute FOLLOW(A) for all non-terminal A apply the following rules until nothing can be added to any FOLLOW set.

) Place $ in FOLLOW(S) where S is start symbol & $ is input right end marker

) If there is a production $A \to \alpha B \beta$ then everything in FIRST(β) except ϵ is in FOLLOW(B)

3) If there is a production $A \to \alpha B$ or $A \to \alpha B \beta$ where FIRST(β) contains ϵ then, everything in FOLLOW(B) is in FOLLOW(A)

* Compute FOLLOW for following grammer.

Note :- 3a) For any production rule $A \to \alpha B \beta$ if $\epsilon \in$ FIRST(β), then FOLLOW(B) = {FIRST(β) - ϵ} $\cup$ FOLLOW(A)

1) $S \to aBDh$
   $B \to cC$
   $C \to bC \mid \epsilon$
   $D \to EF$
   $E \to g \mid \epsilon$
   $F \to f \mid \epsilon$

FIRST(E) = {g, ϵ}
FIRST(F) = {f, ϵ}
FIRST(S) = {a}
FIRST(B) = {c}
FIRST(C) = {b, ϵ}
FIRST(D) = {g, f, ϵ}

FOLLOW(S) = {$}
FOLLOW(D) = {h}
FOLLOW(B) = {FIRST(D) - ϵ} $\cup$ FIRST(h)
= {g, f, h}
FOLLOW(F) = FOLLOW(D) = {h}
FOLLOW(E) = {FIRST(F) - ϵ} $\cup$ FOLLOW(D)
= {f, h}
FOLLOW(C) = FOLLOW(B)
= {g, f, h}

*Compute FOLLOW for following

(2)

$S \rightarrow A$
$A \rightarrow aBA'$
$A' \rightarrow dA' \mid \epsilon$
$B \rightarrow b$
$C \rightarrow g$

FIRST(S) = {a}
FIRST(A) = {a}
FIRST(B) = {b}
FIRST(C) = {g}
FIRST(A') = {d, ε}

$\Rightarrow$ FOLLOW(S) = {$}
FOLLOW(A) = ~~FIRST(B)~~ {$}
FOLLOW(B) = {FIRST(A') − ε} ∪ FOLLOW(A)
= {d, $}
FOLLOW(A') = FOLLOW(A) = {$}
FOLLOW(C) = {NA}

(3) $S \rightarrow (L) \mid a$
$L \rightarrow SL'$
$L' \rightarrow ,SL' \mid \epsilon$

FIRST(S) = {(, a}
FIRST(L) = {(, a}
FIRST(L') = {, , ε}

$\Rightarrow$

~~FOLLOW(S)~~ = ~~FOLLOW('L'}~~ = ~~FIRST(L') = {)$}~~
FOLLOW(L') = {)}
FOLLOW(L) = {)}
FOLLOW(S) = {FIRST(L') − ε} ∪ FOLLOW(L) ∪ {
= {[redacted],), $}

(4) $S \rightarrow AaAb \mid BaBb$
$A \rightarrow \epsilon$
$B \rightarrow \epsilon$

FIRST(A) = {ε}
FIRST(B) = {ε}
FIRST(S) = {a, b}

FOLLOW(S) = {$}
FOLLOW(A) = FIRST(a) ∪ FIRST(b) = {a, b}
FOLLOW(B) = {a, b}

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

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

⇒ FOLLOW(E) = {$,)}
FOLLOW(E') = FOLLOW(E) = {$,)}
FOLLOW(T) = {FIRST(E') − ε} ∪ FOLLOW(E')
= {*, $,)}
FOLLOW(T') = FOLLOW(T) = {*, $,)}
FOLLOW(F) = (FIRST(T') − ε) ∪ FOLLOW(T)
= {*, +, $,)}

(6) $S \to ACB | CbB | Ba$
$A \to da | BC$
$B \to g | \epsilon$
$C \to h | \epsilon$

FIRST(C) = {h, ε}
FIRST(B) = {g, ε}
FIRST(A) = {d, g, h, ε}
FIRST(S) = {d, g, h, ε, a, b}

⇒ FOLLOW(S) = {$}
FOLLOW(B) = FIRST(a) ∪

* LL(1) Grammer :-

* Construct predective parsing table for following gram

① $S \to (L) | a$ ⇒
$L \to SL'$
$L' \to ,SL' | G$

Non Terminal	Input Symbol				
	a	(	)	,	$
S	S→a	S→(L)			
L	L→SL'	L→SL'			
L'			L'→ε	L'→,SL'	

*LL(1) :- A grammer is LL(1) if & only if following cond" holds

① for ~~no terminal~~ whenever $A \to \alpha / \beta$ then following are distiction production then following cond" holds.

① For no terminal 'a' do both α & β derives string begining with a

② Atmost one of α & β derives ϵ

③ If β derives ϵ then FIRST(α) & FOLLOW(A) are disjoints sets. Similarly

*Non-recursive predective Parsing:-

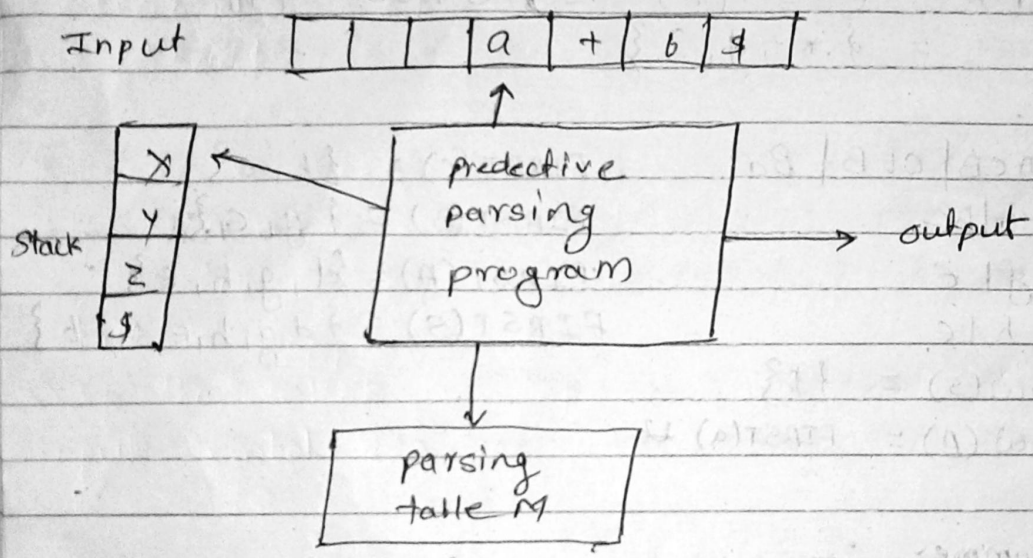


Fig: Table driven predictive parsing

*Algorithm:-

- Input :- A string 'w' & parsing table M
- Output :- If 'w' is in L(G) then left most derivation of 'w' otherwise & error indication.
- Method :- Initially the parser is in configuration with w$ in the input buffer & start symbol S(G) on top of the stack above '$'

① Let 'a' be the first symbol of 'w';

② Let 'X' be the top stack symbol;

```
while (X ≠ $) {
    if (X = a) pop the stack & let 'a' be next symbol of w;
    else if (X is a terminal) then error();
    else if (M[X, a] is an error entry) then error();
    else if (M[X, a] = X → Y1, Y2, ..... YK)
    {
        output the production X → Y1, Y2 ... YK;
        pop the stack;
        push YK, YK-1, ..... Y1 on to the stack
        with Y1 on top;
    }
    Let X be the stack top symbol;
}
```

* in error recovery fill follow of production with synch

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'		T' → ϵ	T' → ϵ
F	F → id			F → (E)		

Matched	STACK	INPUT	ACTION
	E $	id + id * id $	
	TE' $	id + id * id $	output E → TE'
	FT'E' $	id + id * id $	output T → FT'
	id T' E' $	id + id * id $	output F → id
id	T' E' $	+ id * id $	match id
id	E' $	+ id * id $	output T' → ϵ
id	+ TE' $	+ id * id $	output E' → +TE'
id +	TE' $	id * id $	match +

id+	FT'E'$	id*id$	Output T→FT'
id+	idT'E'$	id*id$	output F→id
id+id	T'E'$	*id$	match id
id+id	*FT'E'$	*id$	output T'→*FT'
id+id*	FT'E'$	id$	match *
id+id*	idT'E'$	id$	output F→id
id+id*id	T'E'$	$	match id
id+id*id	ϵE'$	$	output T'→ϵ
id+id*id$	E'$	$	match $ output
id+id*id	$	$	output E'→ϵ
id+id*id$	-	-	match $

* Error recovery in predective parsing :-

1) If M(A, a) is blanks then input symbol a is skiped.
2) If entry is synch then non-terminal on top of stack is popped.
3) If the token on top of the stack doesnot match input symbol then pop token from the stack.