* CONSTRUCTION OF PREDICTIVE PARSING TABLE

PROCEDURE :-

For each production $A \to \alpha$ apply the following steps to fill the entries of the predictive parsing table, $M[A, a]$:-

1. Add $A \to \alpha$ to the $M[A, a]$ $\forall$ $a \in$ FIRST(α)
2. Add $A \to \alpha$ to $M[A, a]$ $\forall$ $b \in$ FOLLOW(A) if FIRST(α) contains ε.
3. make all other entries as error.

PREDICTIVE PARSING ALGORITHM :-

Let X be the symbol on Top of the Stack
a be the symbol under input pointer
then the parser behaves as follows :-

1. If $x = a \neq \$$, then pops off x from the stack ; and advance ip pointer to the next symbol.
2. If $x = a = \$$, parser annouces successful completion of parsing.
3. If x is a Non Terminal, the parsing program contacts Predictive parsing Table Entry $M[X, a]$.

Suppose $M[x, a] = X \to UVW$ then parser replaces X by WVU with U on top.

Suppose $M[X, a]$ = error, the parser calls error recovery unit.

Q. Construct predictive parsing Table for the following grammar G.

(1) $S \to AB$
$A \to aBb \mid \varepsilon$
$B \to b.$

X	first(X)	follow(X)
S	a, b.	$
A	a, ε	b
B	b	$, b

Predictive Table :-

	a	b	$
S	S → AB	S → AB	ERR
A	A → a	A → ε	ERR
B	ERR	B → b.	ERR

(2). $S \to ABe$
$A \to aBb \mid \varepsilon$
$B \to b \mid \varepsilon.$

Predictive Table :-

Moves in this way.

	a	b	e	$
S	S → ABe	S → ABe	S → ABe	ERR
A	A → aBb	A → ε	A → ε	ERR
B	ERR	B → b B → ε	B → ε	ERR

X	FIRST(X)	FOLLOW(X)
S	{a, b, e}	{$}
A	{a, ε}	{b, e}
B	{b, ε}	{b, e}

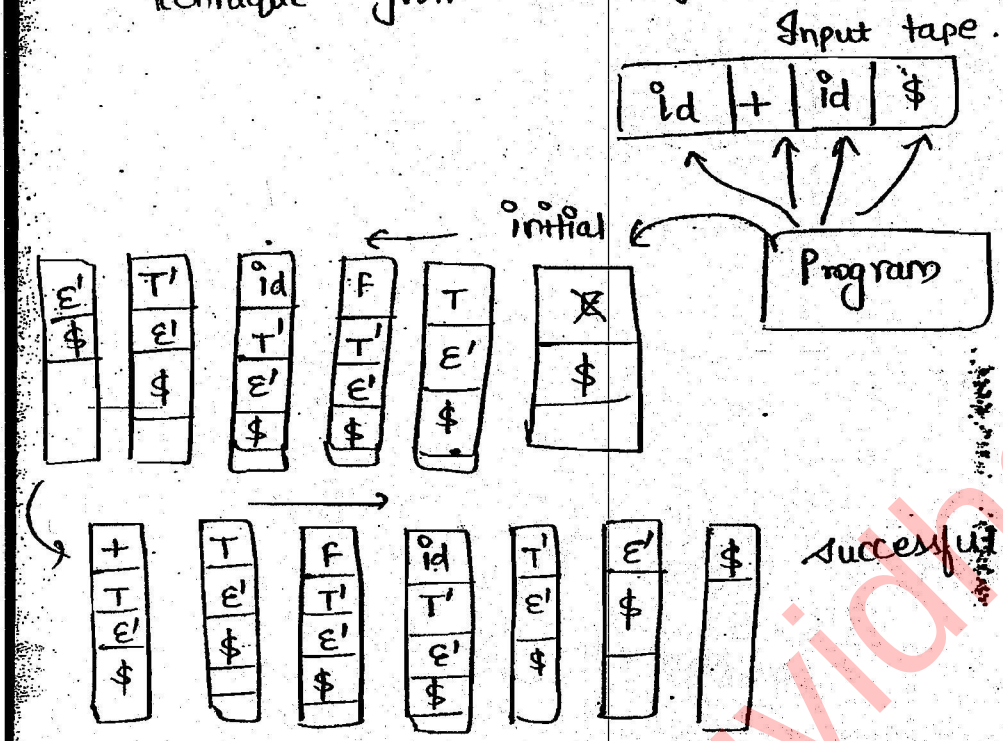
(3). $E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid id$

X	FIRST(X)	FOLLOW(X)
E	(, id	$,)
E'	+, ε	$,)
T	(, id	+, $,)
T'	*, ε	+, $,)
F	(, id	*, +, $,)

Predictive Table -

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

Trace the string id + id using predictive passing technique from above grammar.



★ LL(1) grammars

A grammar G is said to be LL(1) if predictive parsing table do not contains multiple define entries for given G. The LL(1) stands for :-

1. The first L represents in LL(1) represents left to right scan of the input string.
2. The second L represents leftmost derivation.
3. 1 represents use one symbol as a lookahead at a time.

Q. Check whether the following grammar is LL(1) or not :-

(1) S → ABe
A → Aa | b
B → e.

Left Recursive grammar.

Kuch yha pr first (X) mai no ε is then so no problem, it is a LL(1)

X	first(X)
S	b
A	b
B	e

	a	b	e	$
S				
A		A→Aa, A→b		
B			B→e	

Multiple entries so not LL(1)

Note :- Left Recursive grammars can have multiple entries in predictive parsing Table! So, Left Recursive grammar can not be LL(1)

(2) S → AB
A → ab | ad
B → b.

Left factored

so not LL(1)

	a	b	d	$
S				
A	A→ab, A→ad			
B		B→b		

Note :- Left factored grammar can not be LL(1) as they contain multiple entries in Predictive parsing Table.

(3). $S \to iCtSS'$
$S' \to eS \mid \varepsilon$
$C \to b$.
→ Ambiguous grammar.

	i	t	e	b	$
S	$S \to iCtSS'$				
S'			$S' \to eS$ $S' \to \varepsilon$		$S' \to \varepsilon$
C				$C \to b$	

multiple entries.
not LL(1).

Ambiguous grammar can not be LL(1). since they contain multiple entries.

follow(S') = follow(S)
⇒ $ ∈ follow(S)

CONCLUSION

Following grammars are not LL(1)
1. Left Recursive grammar
2. Left factored grammar
3. Ambiguous grammar.

SHORT CUT :-

To verify whether a given grammar G is LL(1) or not we apply following conditions

1. for each non-ε productions,
$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \dots \mid \alpha_n$, LL(1) grammar must hold
$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$
$\forall\, i \neq j$.

② for all ε production of the form
$A \to \alpha | \varepsilon$
Be LL(1) grammar must hold,
$FIRST(\alpha) \cap FOLLOW(A) = \phi$

Dont worry about single productions (from a variab

④. $S \to AB | BA$
$A \to aBb | \varepsilon \Rightarrow FT(aBb) \cap FOLLOW(A) \neq \phi$
$a \cap \{\$, a, b\}$
$B \to Aa | b. \Rightarrow FT(Aa) \cap FT(b)$
$a \cap b = \phi$
do directly,

So not LL(1).
~~FOLLOW(A) = FT(B)~~

⑤. $E \to E+T | T$
$T \to T*F | F$
$F \to id | (E)$.

Gate perspective k acc. hume just given gram ko dekh kar bta ki hai. remove rgrh kik nai dekhna.

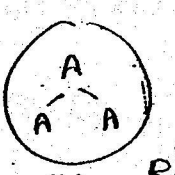
Left recursive, so not LL(1).

agar hum issi grammar ko mai se left recursion remove kar dege tho ye LL(1) ho shi hai (piche kia tha).

But it is not necessary that, if we remove left recursion and left factoring from a grammar, then it becomes LL(1), because then decision depends on ambigious or not.
↳ If ambigious → not LL
If not ambigious → LL.

Q. $A \rightarrow AA \mid (A) \mid \varepsilon$ is not suitable for predictive parsing because the grammar is

(a). ambiguous
(b). Left Recursive
(c). Right Recursive
(d). An operator grammar.



as it is left Recursive as well as Right Recursive, so it is definitely ambiguous so we always give priority to ambiguous.

Q. Which of the following is sufficient to convert an arbitrary CFG to an LL(1) grammar.

(a) Removing left Recursion alone
(b). factoring the grammar alone
(c). Removing the left Recursion and factoring the grammar both.
(d). None of the above.
↳ as it can be ambiguous still.

Q. (2017)
Consider the following grammar :-

$P \rightarrow xQRS$
$Q \rightarrow yz \mid z$
$R \rightarrow w \mid \varepsilon$
$S \rightarrow y$

what is the follow of Q.

(a). R → Never possible (∵ NT). ∞,
(b) w
(c) w, y
(d). w, $

Q. Consider the following grammar shown below:-

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \varepsilon$

$E \rightarrow b$.

In the predictive parsing Table M of this grammar the entries $M[S', e]$ and $M[S', \$]$ respectively are ____ ?

(a). $\{S' \rightarrow eS\}$, $\{S' \rightarrow \varepsilon\}$

(b). $\{S' \rightarrow eS\}$, $\{\}$

(c). $\{S' \rightarrow \varepsilon\}$, $\{S' \rightarrow \varepsilon\}$

(d). $\{S' \rightarrow eS, S' \rightarrow \varepsilon\}$, $\{S' \rightarrow \varepsilon\}$.

By applying shortcut method, check directly.

Q. Consider the following grammar
GATE 2006

$S \rightarrow FR$

$R \rightarrow *S \mid \varepsilon$

$F \rightarrow id$

In the predictive parsing table M of this grammar, the entries $M[S, id]$ and $M[R, \$]$ respectively

STATE RULE

____ ?

$M[S, id] = S \rightarrow FR$

$M[R, \$] = R \rightarrow \varepsilon$.

Q. Consider the following grammar shown below:-

$S \rightarrow cC$

$C \rightarrow aC \mid d$.

This grammar is -:

(a). LL(1)

(b). SLR(1) but not LL(1)

(c). LALR(1) but not SLR(1)

(d). LR(1) but not LALR(1).

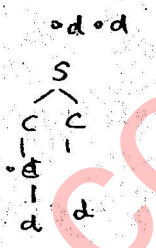
This G is valid for all parses (It is unambiguous

Q. Consider the following grammar -:

$E \to E - T \mid T$

$T \to T + F \mid F$

$F \to (E) \mid id.$



Which of the following grammar is not left Recursive but is equal to G.

Sol.

$$\begin{cases} E \to TE' \\ E' \to -TE' \mid \varepsilon \\ T \to FT' \\ T' \to +FT' \mid \varepsilon \\ F \to (\varepsilon) \mid id. \end{cases} \Rightarrow \begin{array}{l} E \to TX \\ X \to -TX \mid \varepsilon \\ T \to FY \\ Y \to +FY \mid \varepsilon \\ F \to (E) \mid id. \end{array}$$

Q. Consider the following grammar.

$S \to aAbB \mid bAaB \mid \varepsilon$

$A \to S$

$B \to S$

(1) first and follow sets for the non terminals A and B.

X	first(X)	follow(X)
A	a, b, ε	a, b
B	a, b, ε	a, b, $

Q. $S \to asa \mid bS \mid c$

LL(1) or not ?

⇒ LL(1).

Q. $S \to ictSS_1 \mid a$

$S' \to eS_1 \mid \varepsilon$

$C \to b$.

The grammar is not LL1 because.

it is ambiguous.

Q. A student wrote 2 context free grammars G_1 and G_2 for generating a single C– like array declaration. The dimensions of array is atleast 1. for eg. int a[10][3] The grammars uses d as the start symbol and uses the 6 terminal ~~d~~ symbol *, ;, id, (,), no. int id[num][num

(GATE 2016 Set 2)

G_1.

$D \to int\ L;$

$L \to id[E$

$E \to num]$

$E \to num][E$

G_2.

$D \to int\ L;$

$L \to id\ E$

$E \to E[num]$

$E \to [num]$

Which of the grammar correctly contains the declaration maintained above

(a) only G_1

(b) only G_2

(c) @ G_1 & G_2

(d) Neither G_1 Nor G_2.

Q. Which one of the following is Top down parser?

(a). Recursive Descent parser
(b). Operator ~~pe~~ precedence parser
(c). LR(K)
(d). LALR(K).

BOTTOM - UP PARSER

In bottom up parsing we attempt to construct a parse tree starting from leaf nodes and we work towards root node.
The primary objective of bottom up parsing is reducing the given string to start symbol.
In each step of the parsing, we identify the substring in a w that matches with right side string of some production (called Handle) & whenever we encounter a handle, we replace it by left side non-terminal of the production, this process is called reduction.
A bottom up parser is also called as shift reduce parser (SRP) in which the parser peform 4 actions namely shift, reduce, accept & error.
A shift reduce parser can be viewed as Right most derivation in reverse order.