

# Compiler Design

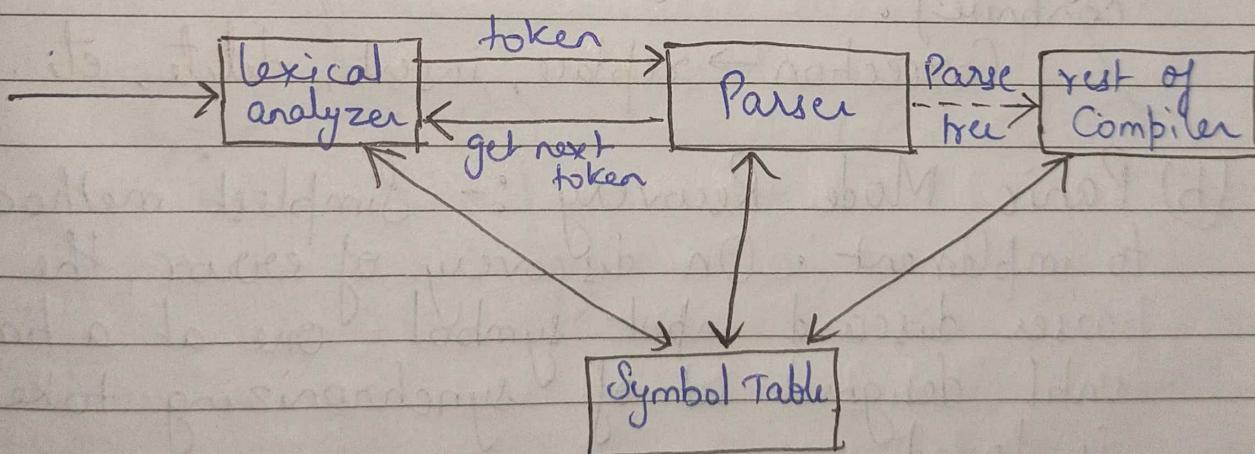
## Unit - 2

Syntax Analysis :- Checking if syntax is correct or not.

The syntax of programming language construct can be described by context-free grammars or BNF (Backus-Naur form) notation.

\* Role of Parser :-

The task of parser is to check syntax. The parser obtains a string of tokens from the lexical analyzer and verify that the string can be generated by grammar for the source language.



Syntax Error Handling :-

Types of Errors :-

- (i) Lexical → misspelling
- (ii) Syntactic → semicolon missing, extra bracket, bracket missing
- (iii) Logical → infinite loop
- (iv) Semantics → operator applied to incompatible operands.

Goals of error handler in a parser :-

- It should report the presence of error clearly and accurately.
- It should recover from each error quickly enough to be able to detect subsequent errors.
- It should not significantly slow down processing of correct program.

### \* Error Recovery Strategies :-

(a) Phrase level recovery :- On discovery of an error a parser may perform local correction on the remaining input, (i.e it may replace a prefix to some string so that parser will continue).

Correction → replace, insert, delete, etc.

(b) Panic Mode Recovery :- Simplest method to implement. On discovery of error, the parser discard input symbol one at a time until designed set of synchronizing token is found.

(c) Error production → Replace common errors.

(d) Global Correction :- For a given string  $X$  with grammar  $G$ , we take similar string  $Y$  and compare them to find error.

## Derivation :-

$$E \rightarrow E + E$$

$$E \rightarrow id$$

① Left-Most Derivation

$$E \rightarrow E + E$$

$$E \rightarrow (E+E) + E$$

$$E \rightarrow id + E + E$$

$$E \rightarrow id + id + E$$

$$E \rightarrow id + id + id$$

② Right-Most Derivation

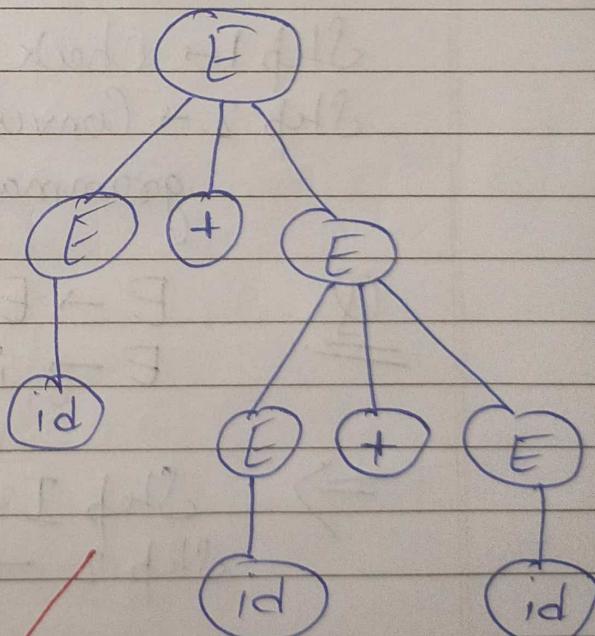
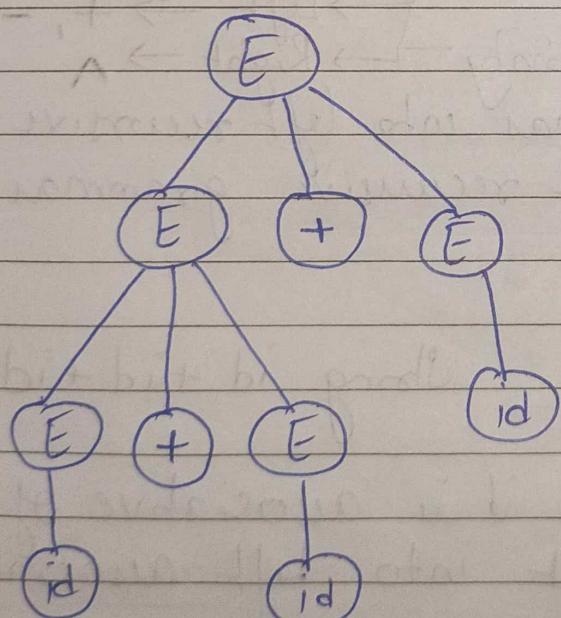
$$E \rightarrow E + E$$

$$E \rightarrow E + (E+E)$$

$$E \rightarrow E + E + id$$

$$E \rightarrow E + id + id$$

$$E \rightarrow id + id + id$$



2-Different Parse tree  
 $\Rightarrow$  Ambigous

When such situation of ambiguity happens and we have to choose which one to take. We follow following rules :-

- (a) Associativity → (when symbols are same)  
e.g. id + id + id
- (b) Precedence → (when symbols are different)  
e.g. id + id \* id

So basically, to remove ambiguity we check for whether or not it is associative or Precedence and then convert accordingly.

Step 1 → Check associativity       $\begin{cases} \text{Left} & \rightarrow +, -, *, / \\ \text{Right} & \rightarrow \wedge \end{cases}$   
 Step 2 → Convert grammar into left recursive grammar / right recursive grammar.

$$\begin{array}{l} \text{Q.} \\ \hline \hline \end{array} \quad E \rightarrow E + E \\ E \rightarrow id$$

String: id + id + id

⇒ Step 1 → Since it is associative of '+'  
 Step 2 → Convert into left associative

$$E \rightarrow E + E \quad \Rightarrow \quad E \rightarrow E + id \\ E \rightarrow id \quad \quad \quad E \rightarrow id$$

Q

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

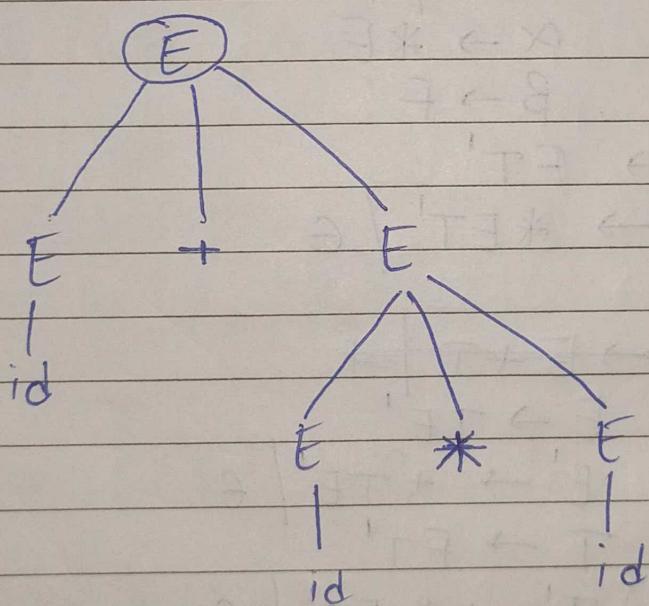
$$E \rightarrow id$$

String  $id + id * id$

$\Rightarrow$  Since string contains different symbols so precedence will follow.

### \* Precedence of operators :-

- 1) Higher level will have operator of less priority.
- 2) Lower level will have operator of high priority.

 $\Rightarrow$ 

### Elimination of Left Recursion :-

$$A \rightarrow A\alpha | \beta$$

$\Downarrow$

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' | \epsilon$$

$$\underline{Q} \quad \begin{array}{l} E \rightarrow E + T \mid T \\ T \rightarrow T * E \mid F \\ F \rightarrow (E) \mid id \end{array}$$

$$\Rightarrow \quad \begin{array}{l} E \rightarrow E + T \mid T \\ \alpha \rightarrow +T \\ \beta \rightarrow T \\ E' \rightarrow TE' \\ E' \rightarrow +TE' / \epsilon \end{array}$$

$$\begin{array}{l} T \rightarrow T * E \mid F \\ \alpha \rightarrow *E \\ \beta \rightarrow F \\ T' \rightarrow FT' \\ T' \rightarrow *ET' / \epsilon \end{array}$$

$$\Rightarrow \quad \begin{array}{l} E \rightarrow E + T \mid + \\ E \rightarrow TE' \\ E' \rightarrow +TE' / \epsilon \\ T \rightarrow FT' \\ T' \rightarrow *ET' / \epsilon \\ F \rightarrow (E) \mid id \end{array}$$

$$\underline{Q} \quad \begin{array}{l} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Sd \mid \epsilon \end{array}$$

$$\Rightarrow \quad \begin{array}{l} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Aad \mid bd \mid \epsilon \end{array}$$

$$\begin{array}{l} A \rightarrow A c \\ \alpha \rightarrow c \\ \beta \rightarrow b d / \epsilon \end{array}$$

$$\begin{array}{l} A \rightarrow A a d \\ \alpha \rightarrow a d \\ \beta \rightarrow b d / \epsilon \end{array}$$

$$\begin{array}{l} A' \rightarrow b d A' / c A' \\ A' \rightarrow a d A' / c A' / \epsilon \end{array} \Rightarrow A \rightarrow b d A' / A'$$

Left factoring :- (Common Prefix Problem) :-

$$A \rightarrow \alpha \beta_1 / \alpha \beta_2 / \alpha \beta_3 / Y$$

$$\begin{array}{l} A \rightarrow \alpha \beta'_1 / Y \\ A' \rightarrow \beta_1 / \beta_2 / \beta_3 \end{array}$$

$$\textcircled{Q}. S \rightarrow i E t S / i E t S e S / \alpha$$

$$\begin{aligned} \Rightarrow \quad \alpha &\rightarrow i E t S \\ \beta_1 &\rightarrow \epsilon \\ \beta_2 &\rightarrow e S \\ Y &\rightarrow \alpha \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad S &\rightarrow i E t S S' / \alpha \\ S' &\rightarrow e / e S \end{aligned}$$

Q.  $S \rightarrow assbss / asasb / abb / b$

$\Rightarrow \alpha \rightarrow a$

$\beta_1 \rightarrow ssbss$

$\beta_2 \rightarrow sasb$

$\beta_3 \rightarrow bb$

$\gamma \rightarrow b$

$S \rightarrow as' / b$

$s' \rightarrow ssbss / sasb / bb$

$\alpha \rightarrow S$

$\beta_1 \rightarrow sbss$

$\beta_2 \rightarrow asb$

$\gamma \rightarrow bb$

$s' \rightarrow ss'' / bb$

$s'' \rightarrow sbss / asb$

①

②

③

For Practice :-

Q.  $S \rightarrow bSSaaS / bSSasb / bsB / a$

Parsing Methods

① Top Down Parsing

→ Starts from Start symbol

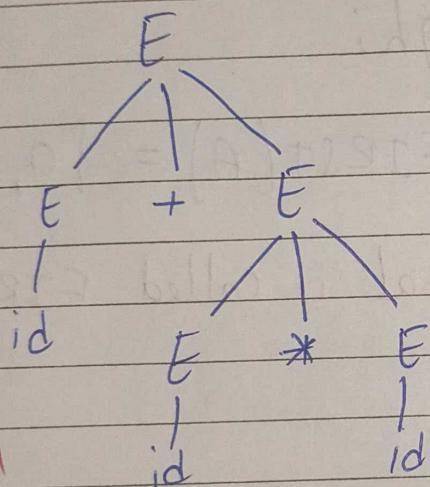
② Bottom-up Parsing

→ Just opposite

Q  $E \rightarrow E+E / E*E / id$   
 Generate  $id + id * id$  using both methods

$E$   
 $\rightarrow E+E$   
 $\rightarrow E+E*E$   
 $\rightarrow E+E*id$   
 $\rightarrow E+id*id$   
 $\rightarrow id+id*id$

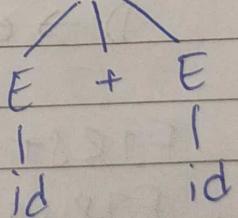
$id + id * id$   
 $\rightarrow E + id * id$   
 $\rightarrow E + E * id$   
 $\rightarrow E * id$   
 $\rightarrow E * E$   
 $\rightarrow E$



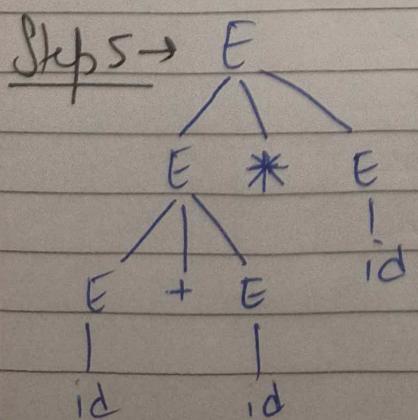
Step 1  $\rightarrow E + id * id$

Step 2  $\rightarrow E + E * id$

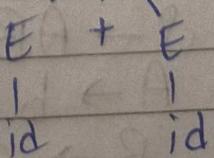
Step 3  $\rightarrow E * id$



Top-Down

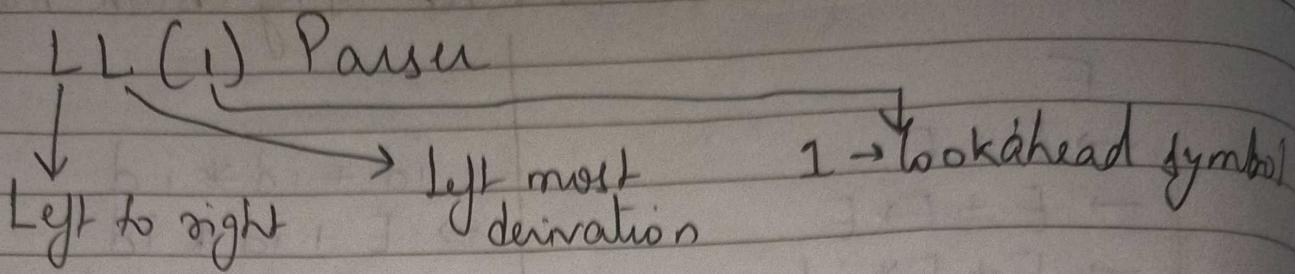


Step 4  $\rightarrow E * E id$



Bottom-up

## Top-Down Parser



\* Computation of FIRST :-

Q.  $A \rightarrow abc \mid def \mid ghi$

$$\Rightarrow \text{First of } A \Rightarrow \text{FIRST}(A) = \{a, d, g\}$$

\* Basically first terminal is called FIRST.

Q.  $\begin{aligned} A &\rightarrow BCD \\ B &\rightarrow C \\ C &\rightarrow d \\ D &\rightarrow e \mid \epsilon \end{aligned}$

$$\Rightarrow \text{FIRST}(A) = \text{FIRST}(B) = \{c\}$$

$$\text{FIRST}(B) = \{c\}$$

$$\text{FIRST}(C) = \{d\}$$

$$\text{FIRST}(D) = \{e, \epsilon\}$$

Q.  $\begin{aligned} S &\rightarrow ABCD \\ A &\rightarrow bDC \\ B &\rightarrow CD \\ C &\rightarrow d \\ D &\rightarrow e \mid ABC \mid \epsilon \end{aligned}$

$$\Rightarrow \text{FIRST}(S) \rightarrow \{b\}$$

$$\text{FIRST}(A) \rightarrow \{b\}$$

$$\text{FIRST}(B) \rightarrow \{d\}$$

$$\text{FIRST}(C) \rightarrow \{d\}$$

$$\text{FIRST}(D) \rightarrow \{e, \epsilon, b\}$$

\* Computation of FOLLOW :-

Q.  $A \rightarrow A \underline{bc}$  Just after A what is there  
(terminal).

$$\text{FOLLOW}(A) = \{b\}.$$

Q.  $A \rightarrow AB \underline{c}$   $\Rightarrow \text{FOLLOW}(A) = \{d\}$   
 $B \rightarrow d$   $\text{FOLLOW}(B) = \{c\}$

Q.  $S \rightarrow ABCD$   $\Rightarrow \text{FOLLOW}(A) = \{c\}$   
 $A \rightarrow b$   $\text{FOLLOW}(B) = \{d\}$   
 $C \rightarrow d/e$   $\text{FOLLOW}(D) = \{\$\}$   
 $D \rightarrow e$   $\text{FOLLOW}(S) = \{d, e\}$   
 $B \rightarrow c$   $\text{FOLLOW}(S) = \{\$\}.$

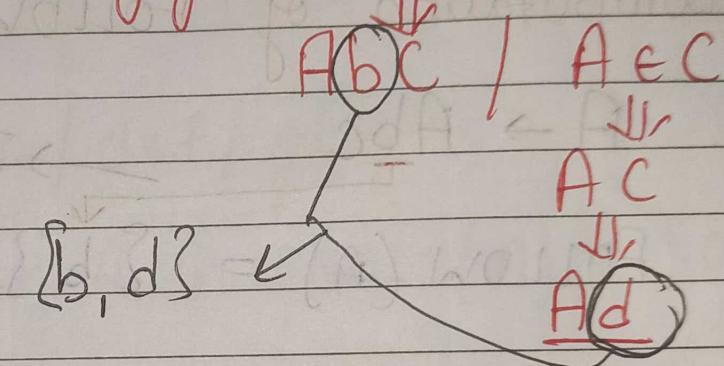
\* Note  $\rightarrow$  Start symbol always have a \$

$S \rightarrow A \underline{BCD}$   
 $D \uparrow$  end me h  $\Rightarrow S$  ka follow copy hogा

$$\text{Q. } S \rightarrow ABC \\ A \rightarrow a \\ B \rightarrow b/e \\ C \rightarrow d$$

$$\Rightarrow \begin{matrix} \sum \$ \\ \sum b, e \\ \sum d \\ \sum \$ \end{matrix} = \{b, d\}$$

Now, follow m e nahi hoga  
so, e ka value put kare ke bad  
kya ayega wo check karenge.



### Practice :-

$$\text{Q. } S \rightarrow ABCD \\ A \rightarrow b/e \\ C \rightarrow d/e \\ D \rightarrow e \\ B \rightarrow c/c$$

$$\Rightarrow \begin{matrix} \sum \$ \\ \sum c, d, e \\ \sum e \\ \sum \$ \\ \sum d, e \end{matrix}$$

$$\text{Q. } S \rightarrow Bb/cd \\ B \rightarrow aB/e \\ C \rightarrow cC/e$$

## LL(1) Parsing Table :-

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

$\Rightarrow$  First of all we make sure there is no left recursion & by L factoring in production rule

We already done this question is left recursion

$$\begin{aligned} \Rightarrow E &\xrightarrow{*} TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *ET' \mid \epsilon \\ F &\rightarrow (E) \mid id \end{aligned}$$

Now, Calculate first and follow.

	FIRST		FOLLOW	
$E \rightarrow TE'$	$\Rightarrow \{ (, id \}$		$\{ \$, ) \}$	
$E' \rightarrow +TE' \mid \epsilon$	$\Rightarrow \{ +, \epsilon \}$		$\{ \$, ) \}$	
$T \rightarrow FT'$	$\Rightarrow \{ (, id \}$		$\{ \$, ), + \}$	
$T' \rightarrow *ET' \mid \epsilon$	$\Rightarrow \{ *, \epsilon \}$		$\{ \$, ), + \}$	
$F \rightarrow (E) \mid id$	$\Rightarrow \{ (, id \}$		$\{ +, *, ), \$ \}$	

LL1 Valid Grammar,  
only one value in one  
box

Now Create Parsing Table						\$
	+	*	(	)	id	\$
$E$			$E \rightarrow TE'$		$E \rightarrow TE'$	
$E'$	$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$		$E \rightarrow \epsilon$
$T$			$T \rightarrow FT'$		$T \rightarrow FT'$	
$T'$	$T' \rightarrow \epsilon$	$T' \rightarrow *ET'$		$T' \rightarrow \epsilon$		$T' \rightarrow \epsilon$
$F$			$F \rightarrow (E)$		$F \rightarrow id$	

Q.  $S \rightarrow A b A a / B a B b$

$$A \rightarrow \epsilon$$

$$B \rightarrow \epsilon$$

$\Rightarrow S \rightarrow A b A a / B a B b$

$$A \rightarrow \epsilon$$

$$B \rightarrow \epsilon$$

FIRST

$$\{b, a\}$$

$$\{\epsilon\}$$

$$\{\epsilon\}$$

FOLLOW

$$\{\$\}$$

$$\{b, a\}$$

$$\{a, b\}$$

	a	b	\$
S	$S \rightarrow A b A a$	$S \rightarrow B a B b$	
A	$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	
B	$B \rightarrow \epsilon$	$B \rightarrow \epsilon$	



Note :- If there is 2 production in any block of the base tree then  $LL(1)$  is not obtained. (We can't generate True).

So, Question can be asked to check whether it is a  $LL(1)$  or not.

Q.  $S \rightarrow A/a$

$$A \rightarrow a$$

$\Rightarrow S \rightarrow A/a$

$$A \rightarrow a$$

$$\{a\}$$

$$\{a\}$$

$$\$$$

$$\$$$

	a	\$
S	$S \rightarrow A/S \rightarrow a$	
A	$A \rightarrow a$	

→ 2 Production in one block.  
⇒ Not  $LL(1)$ .

For Practise :-

Q. Check if following grammar is LL(1) or not.

$$1) S \rightarrow aSbS / bSaS / \epsilon$$

$$2) S \rightarrow aABb \\ A \rightarrow c / \epsilon \\ B \rightarrow d / \epsilon$$

$$3) S \rightarrow aB / \epsilon \\ B \rightarrow bC / \epsilon \\ C \rightarrow cS / \epsilon$$

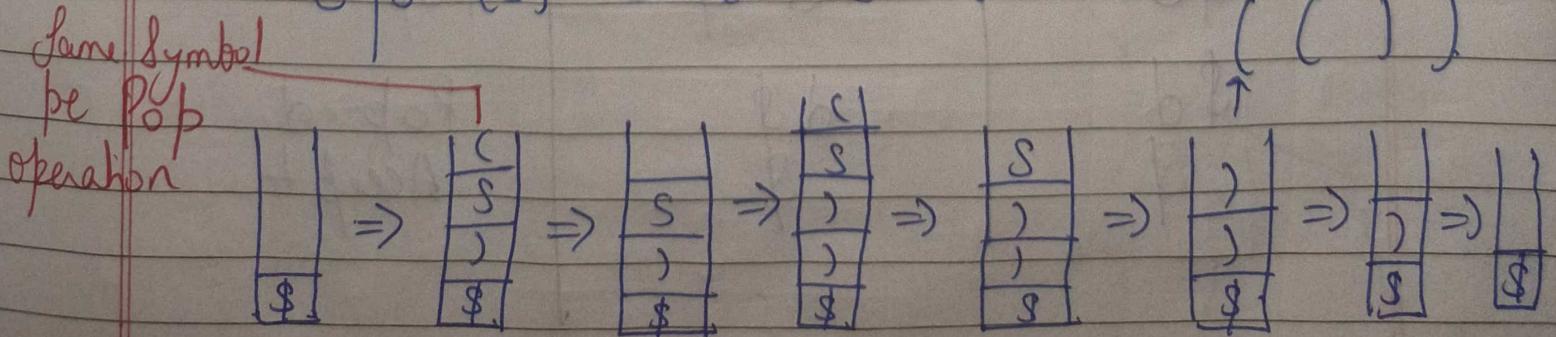
$$4) S \rightarrow aAa / \epsilon \\ A \rightarrow abS / \epsilon$$

\* Making LL(1) Parsing tree from LR(0)  
Parsing Table.

Q.  $S \rightarrow (S) / \epsilon$  Generate (( ))

$$\text{So.} \Rightarrow S \rightarrow (S) / \epsilon \Rightarrow \{c, \epsilon\} \quad \{\$, )\}$$

-	S		C	)	)	\$	\$
-	S		S	→	S		S



How to do in exam?

$$\text{Q} . \quad S \rightarrow cBCd \\ B \rightarrow e / \epsilon \\ C \rightarrow f / \epsilon$$

Generate L(G) for cefd.

$$\text{Sol} \rightarrow \quad S \rightarrow cBCd \\ B \rightarrow e / \epsilon \\ C \rightarrow f / \epsilon$$

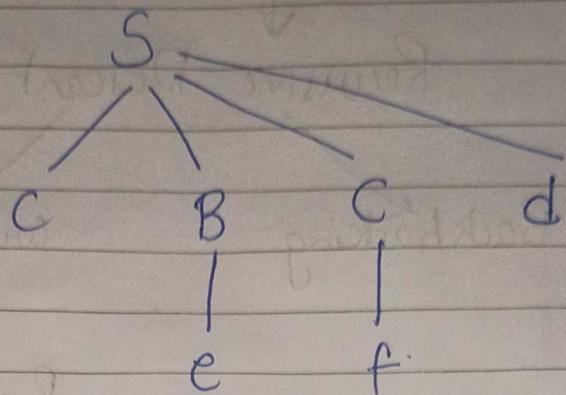
$$\{c\} \\ \{e, \epsilon\} \\ \{f, \epsilon\}$$

$$\{\$\} \\ \{f, d\} \\ \{d\}$$

	c	e	f	d	\$
S	$S \rightarrow cBCd$				
B		$B \rightarrow e$	$B \rightarrow \epsilon$	$B \rightarrow f$	
C			$C \rightarrow f$	$C \rightarrow \epsilon$	

Stack	Input	Action
\$	cefd \$	
\$S	cefd \$	$S \rightarrow cBCd$
\$dCBc	cefd \$.	Pop c
\$dCB	efd \$	$B \rightarrow e$
\$dCe	efd \$	Pop e
\$dC	fd \$	$C \rightarrow f$
\$df	fd \$	Pop f
\$d	d \$	Pop d
\$	\$	Accept

Now draw tree :-

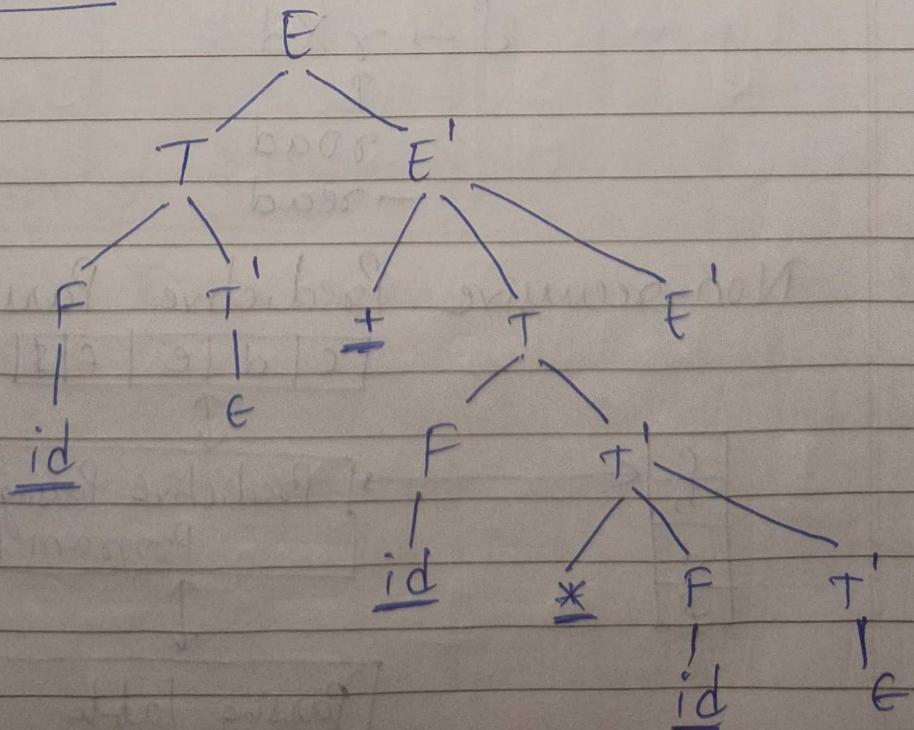


## Question for Practise :-

$$\begin{array}{l}
 \text{Q. } E \rightarrow TE' \\
 E' \rightarrow +TE' / E \\
 T \rightarrow FT' \\
 T' \rightarrow *FT' / \\
 F \rightarrow (E) / \text{id}
 \end{array}$$

Generate LR(0) tree for  $id + id * id$

Ans to Match :-



## Top-Down Parser

### Recursive Descent Parsing

Backtracking

without Backtracking

### Predictive Parser

### LL(1) Parser

If generate errors then don't backtrack.

Backtracking :-

$$S \rightarrow \gamma Xd / \gamma Zd$$

$$X \rightarrow 0a / ea$$

$$Z \rightarrow aid$$

Generate string read

$$S \rightarrow \gamma Xd$$

↑  
road

→ read

Non-recursive Predictive Parsing :-

c | d | e | f | \$

c
b
d
f

Predictive Parsing Program

OUTPUT

Parsing table

Overview :-

TOP DOWN PARSER

Remove left Recursion & Left factoring

Predictive Parser

Remove Backtracking.

L(i) Parser

<u>Q.</u>	$S \rightarrow A \cup S / e / \epsilon$	$\Rightarrow \{e, \epsilon, a, c\}$	$\Sigma \$ \}$
	$A \rightarrow a / cAd'$	$\{a, c\}$	$\{b, d\} \}$
	String aab\$.		

	a	b	c	d.	e	\$
S	$S \rightarrow A \cup S$		$S \rightarrow A \cup S$		$A \rightarrow e$	
A	$A \rightarrow a$		$A \rightarrow cAd'$			$A \rightarrow \epsilon$

Input String	Stack	Action
aab\$	\$S	$S \rightarrow A \cup S$
aab\$	\$SbA	$A \rightarrow a$
aab\$	\$Sba	Pop a
ab\$	\$Sb	Error

Now, If Error occur. Then how to deal.

1) Panic Mode :-

- a. If top of stack does not match then pop top element
- b. If synch. then pop top element
- c. If  $A \rightarrow a$  is empty then skip input symbol.

2) Phrase Mode :-

insert/ replace or delete I/P Symbol.

Let understand how to see it :-

1. a:

Ari error aaya tha usi ko handle  
karte h.

Stack	Input String	Action
\$ \$ b	ab \$	Yha aaya tha error. Now, solve
\$ S	ab \$	Remove top of stack $S \rightarrow A B S$
\$ \$ b A	ab \$	$A \rightarrow a$
\$ \$ b a	ab \$	Pop a
\$ \$ b	b \$	Pop b
\$ \$	\$	$S \rightarrow \epsilon$
\$	\$	Accept String

1. b: Hmlog karte ye h ki Jis v follow first ka symbol me  $\epsilon$  nhi h. Like me we put Synch for follow.

Eg. →	a	b	c	d	e	\$
S	$S \rightarrow A B S$		$S \rightarrow A B S$		$S \rightarrow c$	$S \rightarrow \epsilon$
A	$A \rightarrow a$	Synch	$A \rightarrow c$	Synch.		

Ye Jo last ari solve kar rhe the uska A ka follow wala me synch. dalo.

1. c: For  $A \rightarrow e$ , we just skip input symbol. (i.e agar koi nhi ho 1a ya 1b me se to ye karof).

Question  $\rightarrow S \rightarrow A \cup S^*$

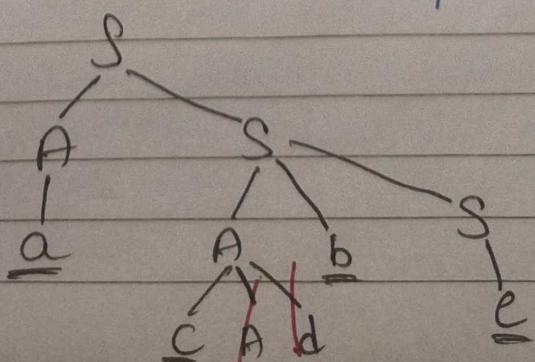
$$A \rightarrow a \cup cAd$$

Generate parse tree for input acbe.

$$\Rightarrow S \rightarrow A \cup S^* \Rightarrow \{a, c, e\} \cup \{\$ \} \quad \{a, c\} \cup \{b, d\}$$

	a	b	c	d	e	\$
S	$S \rightarrow A \cup S^*$		$S \rightarrow A \cup S^*$		$S \rightarrow e$	$S \rightarrow e$
A	$A \rightarrow a$	Synch.	$A \rightarrow cAd$	Synch.		

Stack	Input String	Action
\$S	acbe \$	$S \rightarrow A \cup S^*$
\$SBA	acbe \$	$A \rightarrow a$
\$Sba	acbe \$	Pop a
\$Sb	cbe \$	Pop b (remove top of stack)
\$S	cbe \$	$S \rightarrow A \cup S^*$
\$SbA	cbe \$	$A \rightarrow cAd$
\$SbdAc	cbe \$	Pop c
\$SbdA	be \$	Pop A
\$Sbd	bc \$	Pop d
\$Sb	be \$	Pop \$ & b
\$S	e \$	Pop \$ $S \rightarrow e$
\$e	e \$	Pop e
\$	\$	Accept



For Practice :-

Q.  $E \rightarrow \bullet TE'$   
 $E' \rightarrow +TE' / e$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' / e$   
 $F \rightarrow (E) / id$

Generate Parse tree for )id\*+id

