

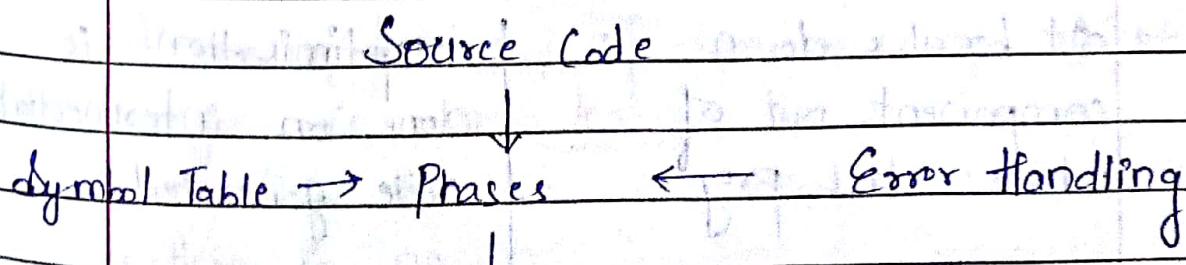
Assembly :-

i/P → compiler → assembly language →  
assembly → target

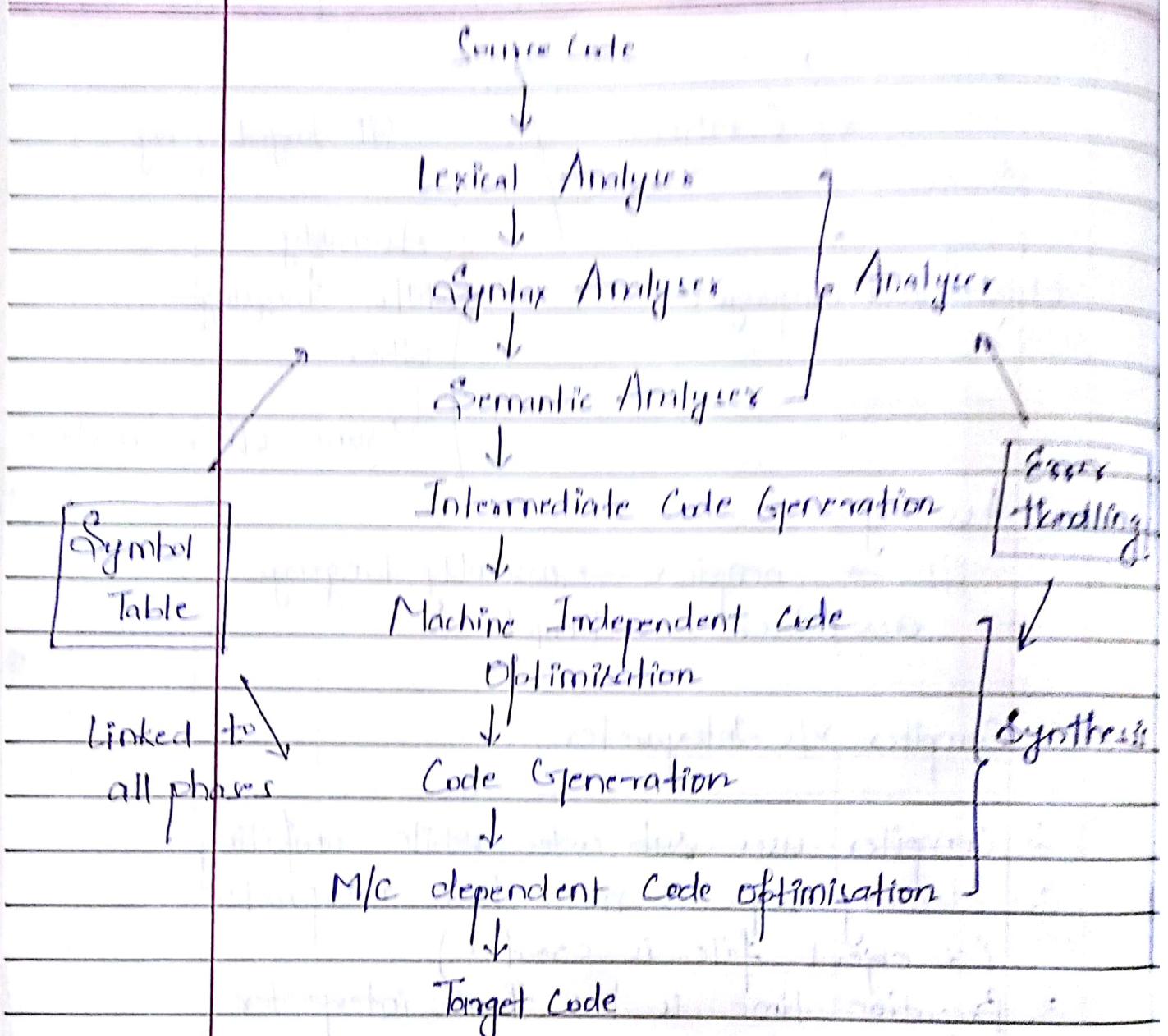
### Compiler v/s Interpreter

1. Compiler uses sub codes while compiling.
2. Memory req. is more than interpreter  
( $\because$  object file is created)
3. Execution time is less than interpreter
4. Compiler has execution as well as compilation time & interpreter has one execution time only

### PHASES OF COMPIRATION



Target Code



Compiler consist of 9 main phases

Compiler  
Analysis Phase      Synthesis Phase

→ It breaks down components out of whole program.

→ Optimization is done on intermediate code generated.

w component are constraints such as int, etc.  
identifier, etc.

- ✓ Then analysis is performed.
- ✓ intermediate code is generated

$$\text{position} := \text{initial} + \text{rate} * 60$$

## # Analysis Phase

### ① Lexical Analysis (Linear analysis / Scanning)

void main ()

{

float initial, rate, position;

position = initial + rate \* 60;

}

• Linear analysis :- bcz it reads each char from left to right i.e. linear scan from left to right.

→ It then converts into some meaningful token. Token are acc. to diff<sup>n</sup> languages.

→ Symbol table contains all keyword from whence token is checked during token generation.

→ A meaningful set of char read by lexical analyser is a lexeme.

Then, this lexeme is matched with existing pattern in present language & then check it in symbol table to confirm if it was a keyword.

E.g. void matches variable name pattern but also present in Symbol Table.  
∴ It is a keyword.

- It only validates lexeme with tokens of source language.
  - Lexeme is current set checked.
  - token is stored constraints.

eg.	void → lexeme	;	→ lexeme
	Keyword → token		semicolon → token

→ (variable name)

- As soon as a new identifier is found, it is entered in symbol table.

- v if, encountered next time, pcf is linked  
to that ptr & stored in ST.

## Symbol Table

Keywords	
position	id1 ← info

## # O/P of Lexical Analysis

<position, id1, pto>  
of ST

<+, plus sign>

w Shows it will not report :-

main()

validate ↗ if not present then it is not a  
problem in this phase.

w OI only checks stored tokens.

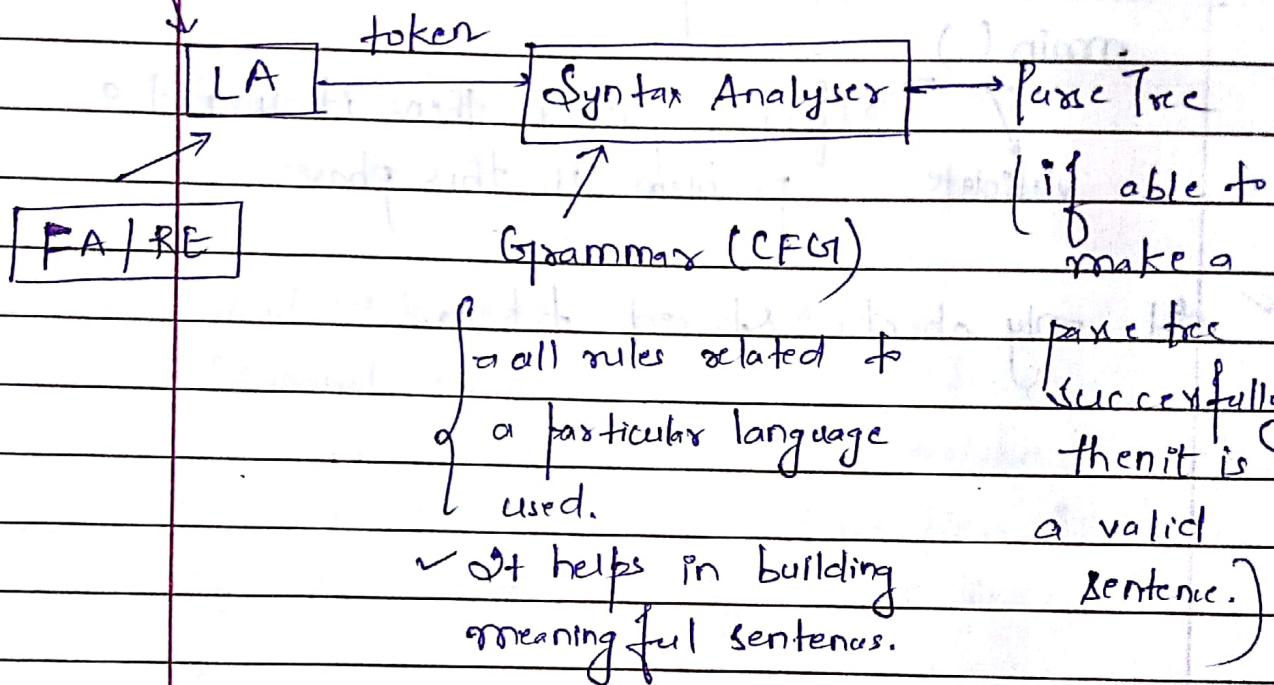
Q.

## SYNTAX ANALYZER

also, Hierarchical Analysis  
Parsing

so tree is traversed.

Src Code



## # Expression

1. Any identifier is an expression
2. Any number "
3. If  $\exp_1$  &  $\exp_2$  are expressions  
then,

$\exp_1 + \exp_2$  } are exp.  
 $\exp_1 * \exp_2$  }  
( $\exp_1$ ) }

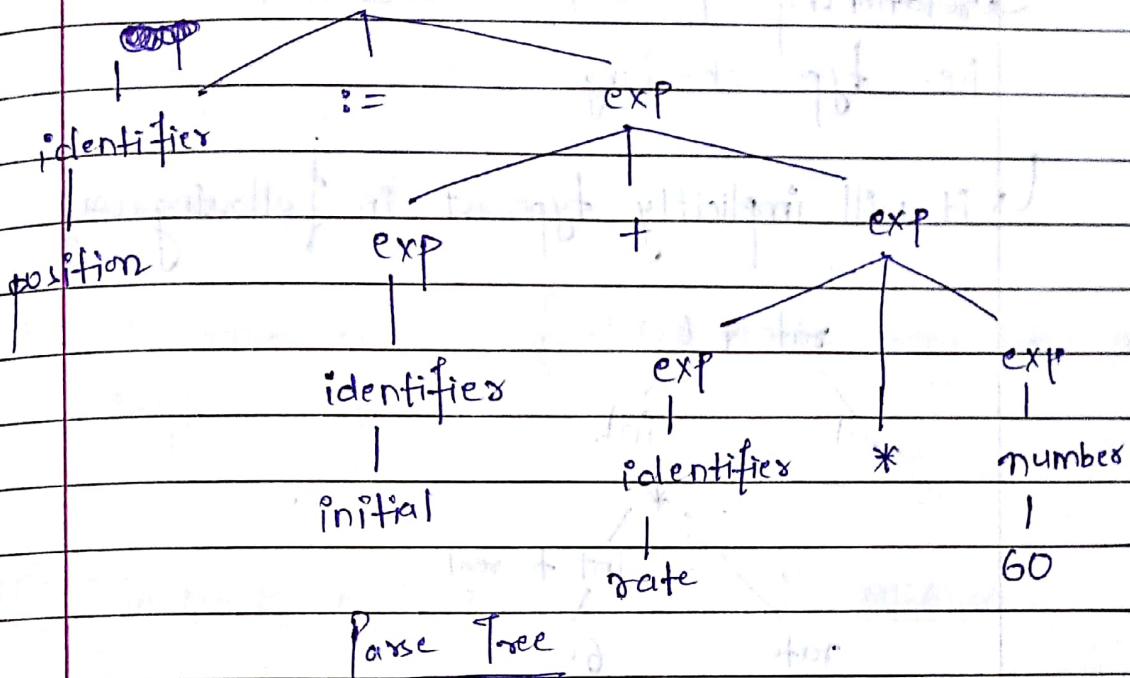
## # Statement

1.  $\text{identifier} = \exp_2$
2.  $\text{while } (\exp) \text{ do } \text{stmt}_2$

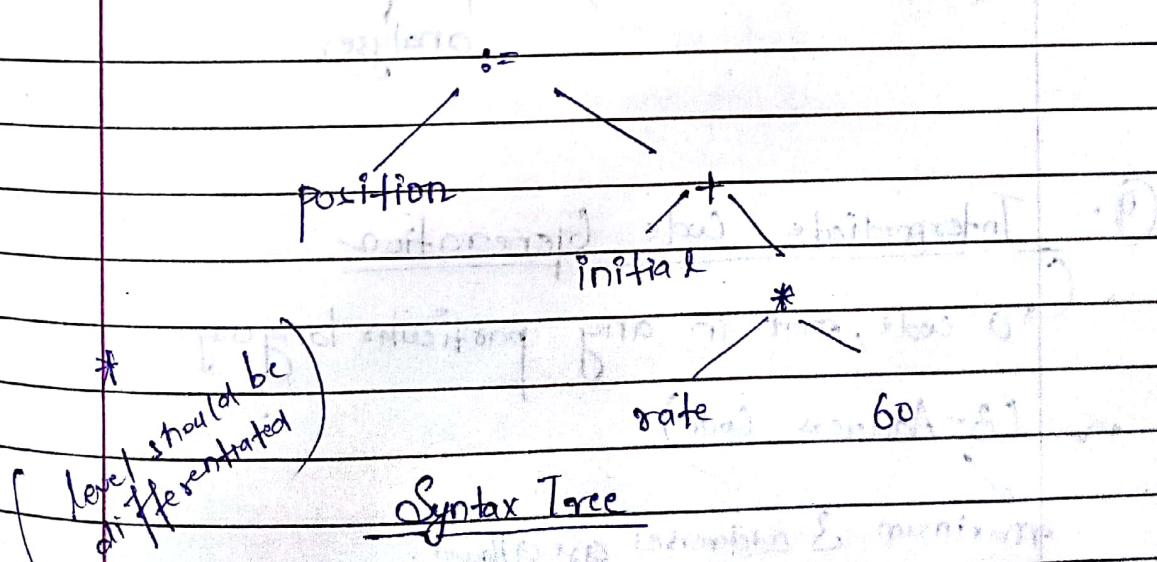
I/P string

e.g. position := initial + rate \* 60

statement  
is obtained.  
→ a valid statement



# Compressed representation of Parse Tree is Syntax Tree.



### ③. Semantic Analysis

↳ relationship b/w variables is checked  
i.e. type checking.

↳ it will implicitly typecast in following way

rate \* 60

real                    int

\*  
int to real

rate                    60

or

+  
rate

60.0

(Annotated Parse Tree) → o/p of Semantic analyser.

### ④. Intermediate Code Generation

↳ a code, not in any particular language

(3-Address Code)

↳ maximum 3 addresses are allowed.

Eg.  $LHS := = + =$

✓ We use temporary variables to ensure 3-address code

$t_1 := \text{int\_to\_real}(60)$

$t_2 := \text{rate} * t_1$

$t_3 := \text{initial} + t_2$

$\text{position} := t_3$

✓ use temp variables freely to make computation as easy as possible.

## ⑤. MACHINE INDEPENDENT CODE OPTIMISATION

→ This optimisation works on all machines.

$t_1 := \text{rate} * 60.0$

$\text{position} := \text{initial} + t_1$

## → LEXICAL ANALYSER

It also keeps new line character which helps in returning the line no. containing error.

Eg.  $f_i(a = 6)$

- $f_i$  → identifier
- $($  → open bkt
- $a$  → identifier
- $=$  → equal
- $6$  → no
- $)$  → closing bkt

error handling      not matching to any pattern

Eg. abc@#cdf

- $a$  → valid
- $b$  → longest match
- $c$  → not allowed in identifier
- $@$  → panic mode
- $#$  → panic mode
- $c$  → identifiers will be identified
- $d$  → if will throw them.
- $f$  → identifiers will be identified

(abc) lexeme

~~Recovery mode~~

1. ~~w~~ panic mode recovery

2. ~~w~~ deleting an extra char

3. ~~w~~ inserting a missing char

4. ~~w~~ replacing incorrect char by a correct char

5. ~~w~~ transposing adjacent char

Ex. 123.

matching no. but not a complete no.

- # ∵ to continue it takes inserting a missing char recovery mode.

Ex. 123, 48

locally it will replace it with . to validate it.

(4)<sup>th</sup> recovery mode.

Ex. E 9 -4

exponent locally handled & replaced by 9 E -4

(5) this recovery mode.

## # Terms related to Automata

1. Symbol → any char 0, 1, a, b, # - etc.
2. ( $\Sigma$ ) Alphabet → set of symbols
3. String → finite seq of symbol
4. Language → set of strings generated by a regular expression.
5. (abc) prefix (n+) if n symbols including .a, e, ab, abc

6. suffix (n+1) c, bc, abc, etc.

7. proper prefix  $\Rightarrow$  abc i.e. w if string not included (here, w = abc)

8. proper suffix  $\Rightarrow$  abc.

$\wedge \ast$

precedence.

$\cdot$  concatenation

$+$  union

$\emptyset \rightarrow$  null set containing  $\emptyset$   $\sqcup$   
 $\epsilon \rightarrow$  empty string.

## # Regular definition

regular expressions are assigned a name which is used, while writing R.E. containing those exp.

Eg.

$/ \rightarrow$  same as +

letter  $\rightarrow$  only R.E. =  $(a/b/c/-/z/A/B/-/z)(a/b/c/-/z/A/B/-/z)$   
 digit  $\rightarrow$  only  $0/1/2/3/4/5/6/7/8/9$   
 with  $\rightarrow$  id =  $(id)^*$

Converting to  
definition.

letter  $\rightarrow A/B/C - /z/a/b/-/z$   
 digit  $\rightarrow 0/1/2/3/4/5/6/7/8/9$

Notes,

Identifier  $\rightarrow$  letter (letter | digit)\*

Eg. (number)

exponent, fraction,

digit  $\rightarrow$  0/1/2/3/4/5/6/7/8/9

sign  $\rightarrow$  +/-

digits  $\rightarrow$  digit digit\* or (digit+)

op fraction  $\rightarrow$  e | . digits

op exp  $\rightarrow$  e | E (+/-/e) digit

num  $\rightarrow$  (digits | op fraction | op exp)  
 $\quad\quad\quad (+/-/e)$

num  $\rightarrow$  (+/-/e) (digits | op fraction | op exp)

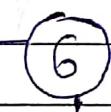
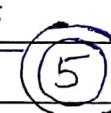
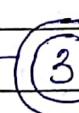
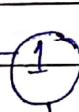
## # Transitional diagram

(1). relational  
operator allowed  
in PASCAL  
 $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $=$ ,  $\neq$

not equal to

relational  
operator

start



return (reloop, LE)

return (reloop, NE)

return (reloop, LT)  
if means wrong, now retract  
to prev. state

otherwise

return (relop, EQ)

return (relop, GTE)

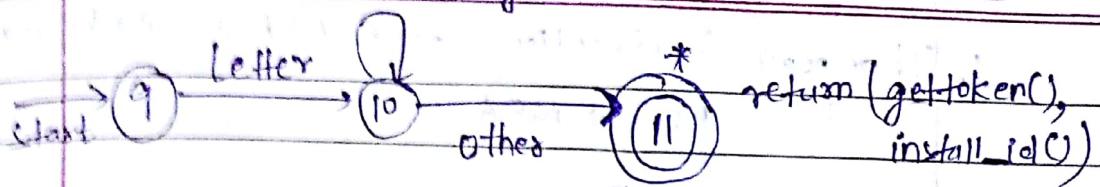
return (relop, GT)

Every time a  
terme matches  
starting at state

by (6)

- (2). Identifier  $\rightarrow$  starting with alphabet only,  
followed by a numeral or  
alphabet.

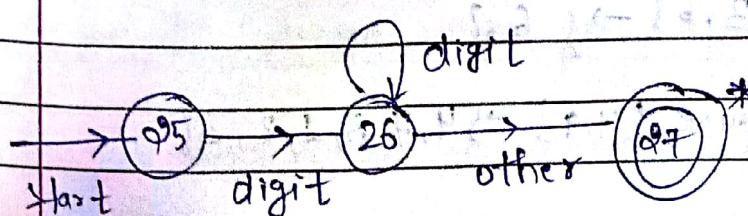
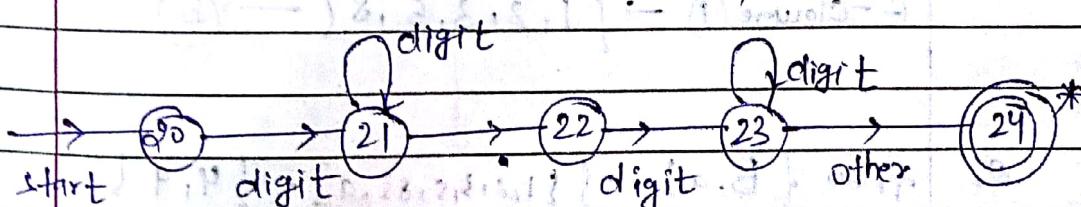
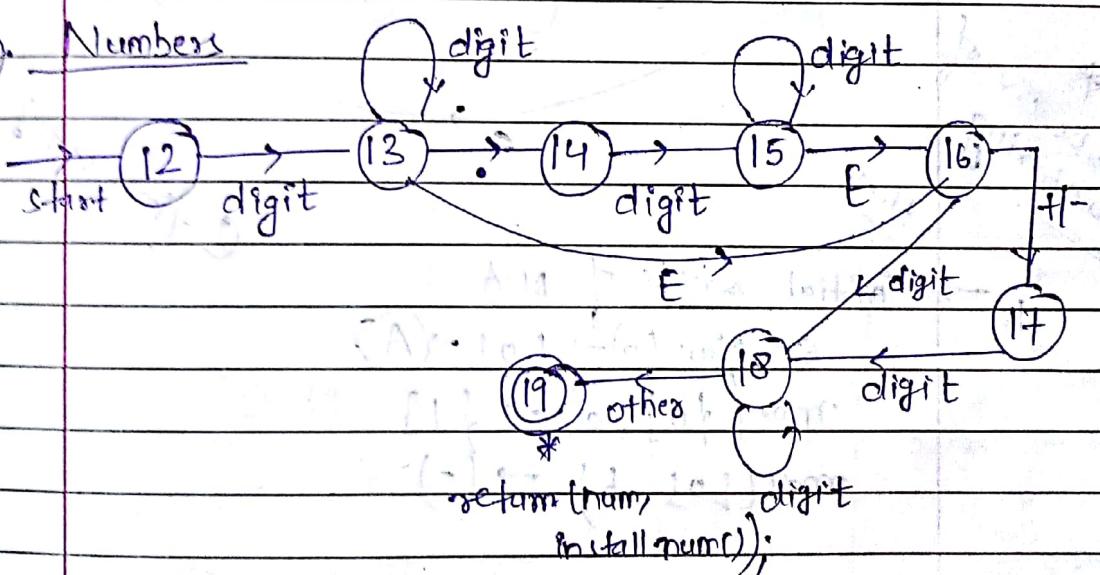
letter/digit



{ if, lenme is  
to token  
keyword then,  
it will return

} install\_id() as to  
else, it will make  
an entry and will  
return ptr to that  
entry!

### ③. Numbers

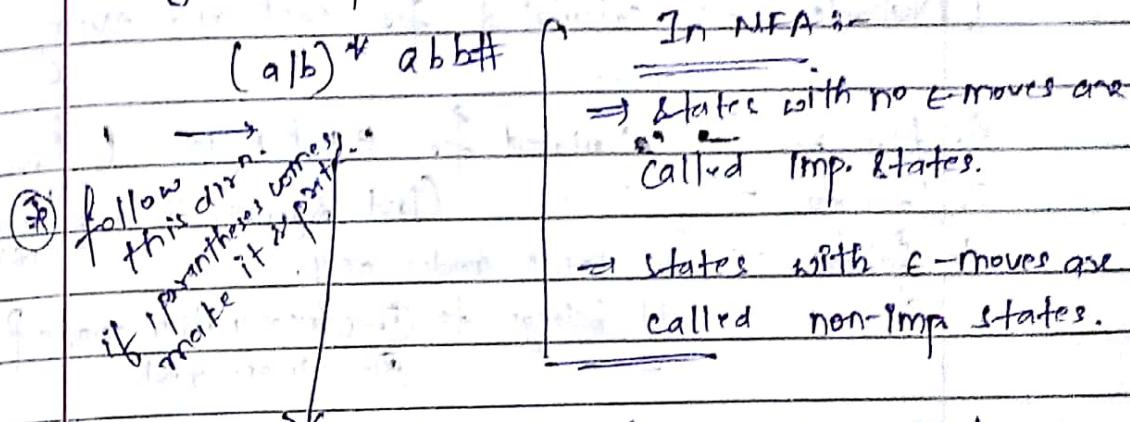


\* If directly mentioned then draw NFA w/o E.

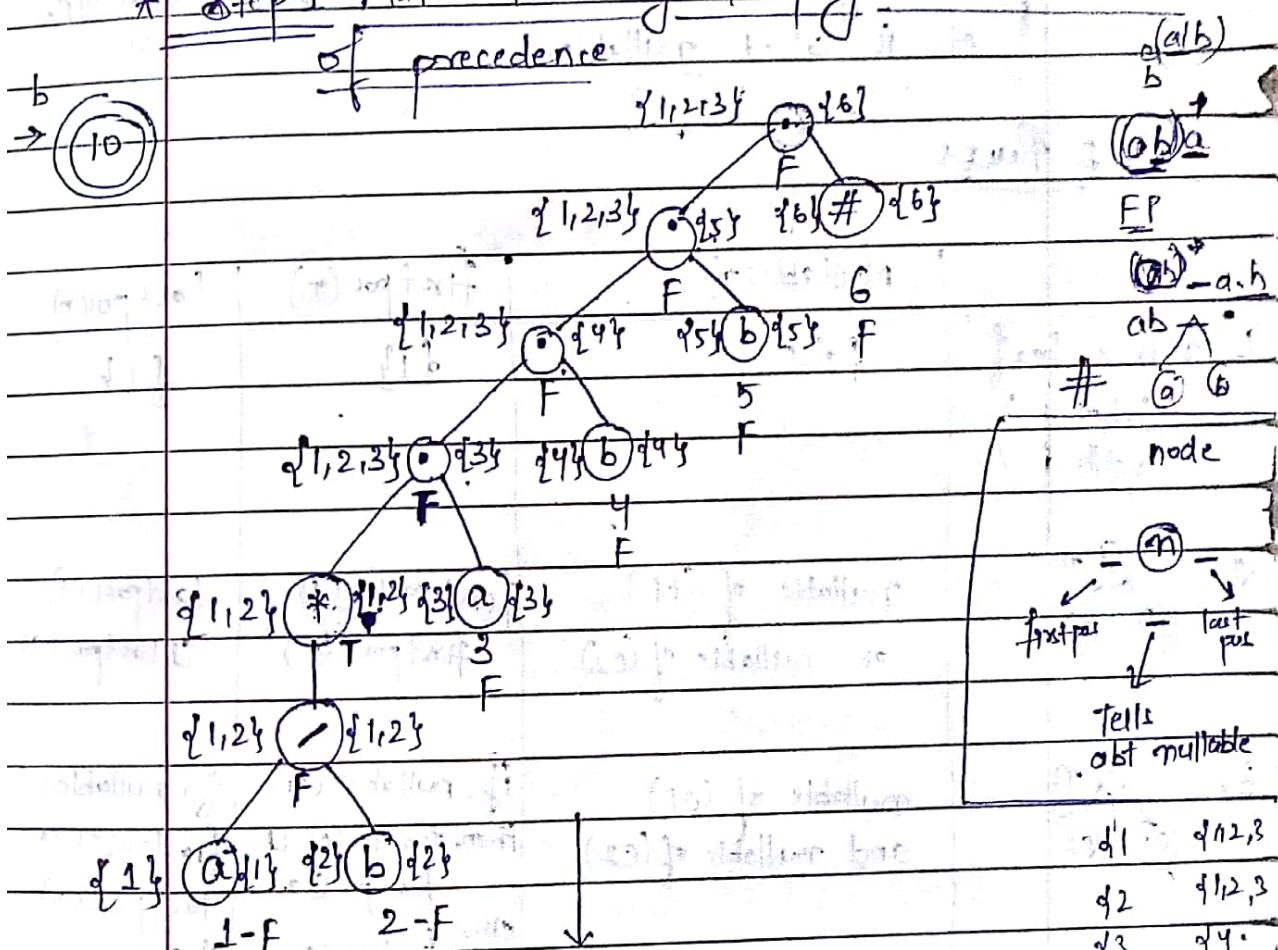
Page No.

Date: / /

## 2. Regular Expression to DFA



\* Step I: Make Tree by keeping in mind order of precedence



\* Step II: Assign unique pcf. to leaf nodes.

Every node (interior or exterior) is an expression.

Step 3: Calculate following fn for the tree

1. Nullable → calculated for each and every node

- a. firstpos → calculated for each node (first symbol in generated string)
- b. lastpos → " (last symbol in a generated string)
- c. follow → for some nodes only.  
it belongs to concatenation node  
(\*) → clear node

→ if a language generates 'ε' along with any o/p, it is called nullable.

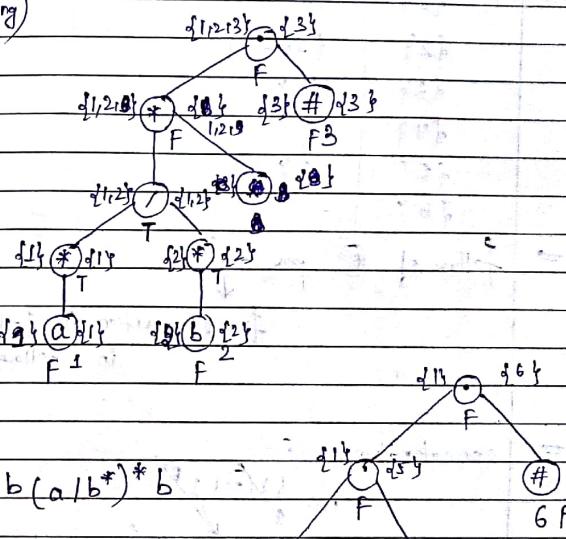
#### Rules

	Nullable( $\eta$ )	firstpos( $\eta$ )	lastpos( $\eta$ )
1. min. leaf (labeled with $\eta$ )	false	$\{1\}$	$\{1\}$
2.	nullable of ( $C_1$ ) or, nullable of ( $C_2$ )	$firstpos(C_1) \cup firstpos(C_2)$	$lastpos(C_1) \cup lastpos(C_2)$
3.	nullable of ( $C_1$ ) and nullable of ( $C_2$ )	$if, nullable(C_1)$ then, $firstpos(C_1) \cup firstpos(C_2)$ $else, firstpos(C_1)$	$if, nullable(C_2)$ $lastpos(C_1) \cup$ $else,$ $lastpos(C_2)$
4. *	true	$firstpos(C)$	$lastpos(C)$

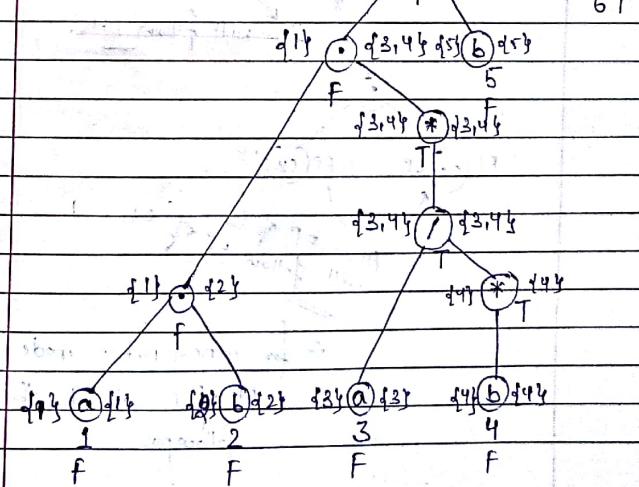
Q. e.g.  $(a^* / b^*)^*$

symbol in generated string

(H.W.)



$ab(a/b^*)^*b$



find out

positions

→ follow is calculated for all nodes but they are only calculated for (0) and (1) nodes.

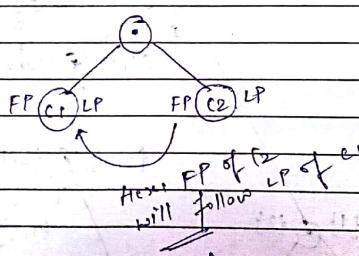
• (abb)\*abb#

pos	follow
{1}	{1, 2, 3, 4}
{2}	{1, 2, 3}
{3}	{4}
{4}	{5}
{5}	{6}
{6}	-

→ follow of (0) pos →

first position is followed up in follow pos.

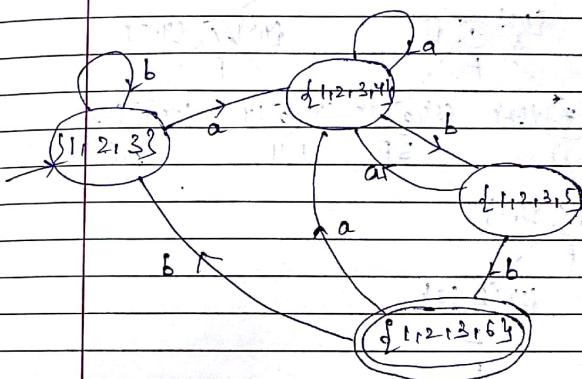
⇒ concatenation, (0)



∴ for concatenation mode,  
LP of c2 will be the  
follow pos.

Rules :-

1. Start state of DFA, consists of all (FP) of root node.
2. To find out transition, we find that transition variable or pos included in current mode or state. Then, union of all the followpos is taken.



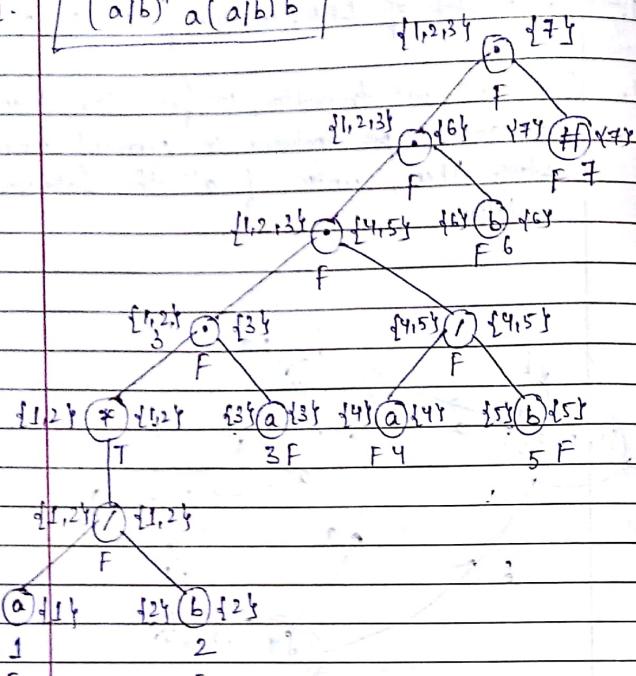
3. Accepting state = # pos.

(Here, # pos = 6)

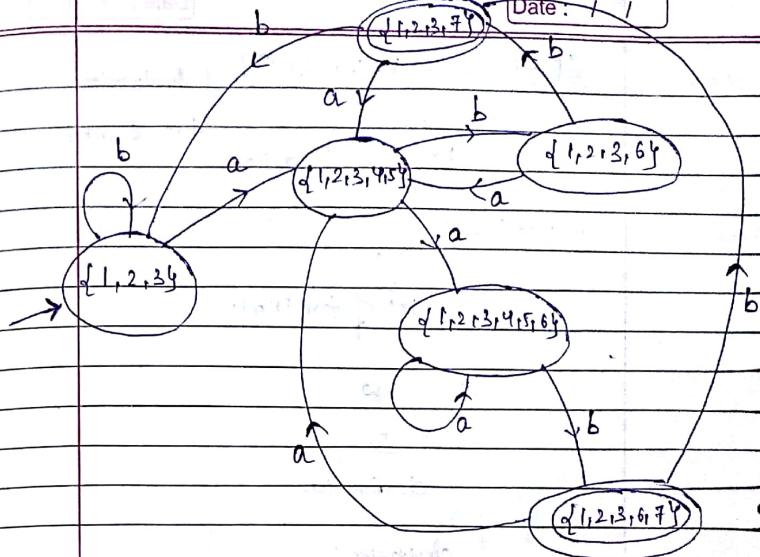
Now, all states including # pos will have final state.

Page No.  
Date: 1/1

$$Q. \quad (a|b)^* a(a|b)b$$



Page No.  
Date: 1/1



follow up

pos	follow
{1}	{1,2,3}
{2}	{1,2,3}
{3}	{4,5}
{4}	{6}
{5}	{6}
{6}	{7}
{7}	-

## SYNTAX ANALYSIS

Page No.

Date: / /

$CFL, G_1 = \{ V, T, S, P \}$  production  
 Non-Terminals      Terminals      Start Symbol  
 terminals

$L(G_1) = \{ \dots \}$   
 set of terminals

Q    W

$S \xrightarrow{*} \omega$   
 derivation

derivation

Leftmost      Rightmost

- ✓ Graphically derivation & Parse tree.
- ✓ Ambiguity  $\Rightarrow$  leftmost & rightmost 2 or more parse trees for same string.

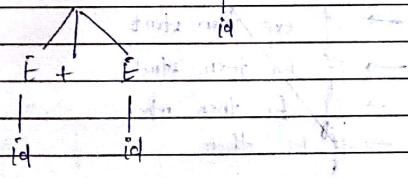
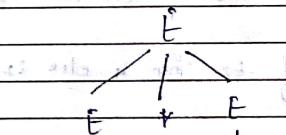
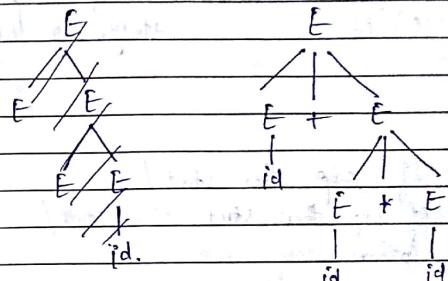
$E \rightarrow E * E / E + E / (F) / -E / id$

string  $\rightarrow (id + id * id)$

Two right most

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

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



$\Rightarrow$  Some parser can take ambiguous trees but, they need to have some disambiguation rule to handle it.  $\rightarrow$  difficult task.

To handle this we try to convert ambiguous grammar into unambiguous grammar.

Unambiguous grammar for last eg:-

$$\begin{aligned} E &\rightarrow E+E \mid T & \rightarrow \text{only one left most} \\ T &\rightarrow T+F \mid F & \text{derivation for a particular} \\ F &\rightarrow (E) \mid id & \text{string, e.g. Rightmost} \end{aligned}$$

stmt  $\rightarrow$  if exp then stmt /  
if exp then stmt else stmt /  
other

~~not-ambiguous~~ (1). if  $E_1$  then ( $S_1$  else if  $E_2$  then  $S_2$  else  $S_3$ )

~~ambiguous~~ (2). if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$ .

stmt  $\rightarrow$  if exp then stmt  
 $\rightarrow$  if  $E_1$  then stmt  
 $\rightarrow$  if  $E_1$  then other  
 $\rightarrow$  if  $E_1$  then

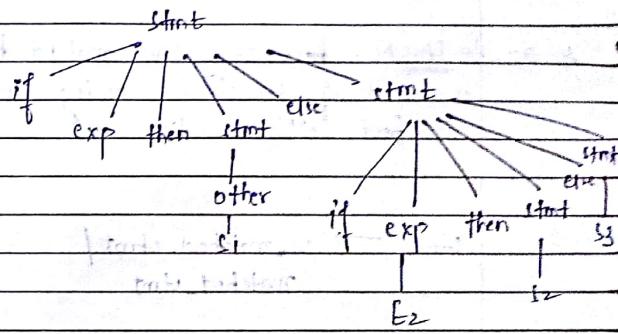
stmt  $\rightarrow$  if exp then stmt else stmt  
 $\rightarrow$  if  $E_1$  then ~~stmt~~ else (stmt)  
 $\rightarrow$  if  $E_1$  then other else stmt  
 $\rightarrow$  if  $E_1$  then  $S_1$  else stmt  
 $\rightarrow$  if  $E_1$  then  $S_1$  else if exp then

$\rightarrow$  if  $E_1$  then  $S_1$  else if  $E_2$  then other else

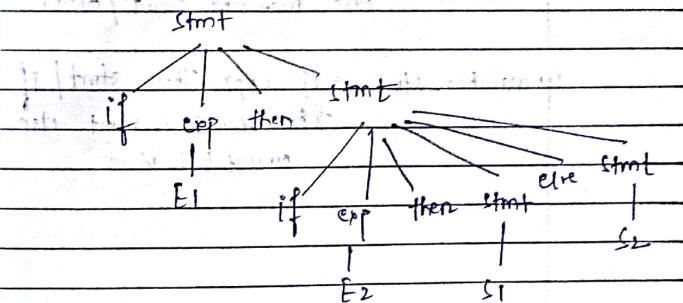
$\rightarrow$  if  $E_1$  then  $S_1$  else if  $E_2$  then other else

$\rightarrow$  if  $E_1$  then  $S_1$  else if  $E_2$  then  $S_2$  else

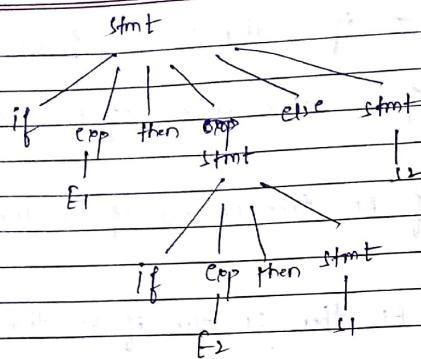
$\rightarrow$  if  $E_1$  then  $S_1$  else if  $E_2$  then  $S_2$  else  $S_3$



(a)



Page No.  
Date: / /



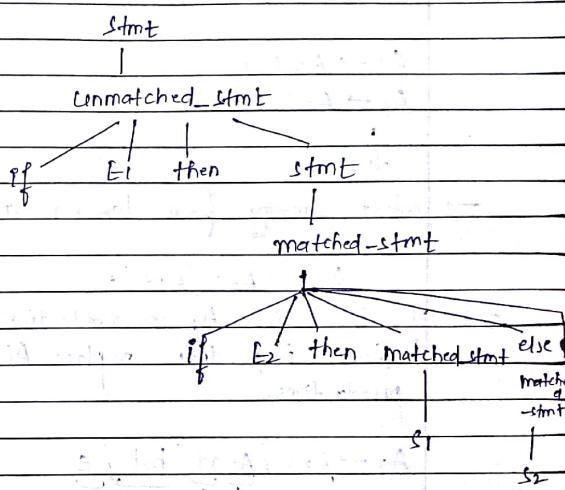
\* acc. to PASCAL, ~~last~~ every elck matches to its new most if, which is coming incorrect in ~~last~~ derivation.

$\text{stmt} \rightarrow \text{unmatched\_stmt} / \text{matched\_stmt}$

$\text{matched\_stmt} \rightarrow \text{if expr then matched\_stmt}$   
else matched\\_stmt / other

$\text{unmatched\_stmt} \rightarrow \text{if expr then stmt} / \text{if expr then matched\_stmt else unmatched\_stmt}$

Eg. if E<sub>1</sub> then if E<sub>2</sub> then S<sub>1</sub> else S<sub>2</sub>



Now, it is unambiguous grammar.

### # LEFT RECURSION ( $A \rightarrow A\alpha / \beta$ )

\* Top-Down parser do not accept left recursion  
Bottom-up parser can accept left recursion.

$$A \rightarrow B A' \\ A' \rightarrow \alpha A' \beta$$

Left-Recursion  
Immediate (Clearly visible)  
Non-Immediate (after substitution, it becomes visible.)

Eg:

$$S \rightarrow A_0 | b$$

$$A \rightarrow A_1 c | S d | E$$

$$A_1 \rightarrow A_2 a | b$$

$$A_2 \rightarrow A_2 c | A_1 d | E$$



→ Start removing from lower most production.

$$A_2 \leftarrow A_2 c | A_1 d | E$$

\* If there is  $A_j$  in production of  $A_i$  such that  $i < j$  then replace  $A_i$  with its productions.

$$A_2 \rightarrow \underbrace{A_2 c}_{A_1 d_1} | \underbrace{A_2 d}_{A_1 d_2} | \underbrace{bd}_{\beta_1 \beta_2} | E$$

$$A_2 \rightarrow bd A_2' | A_2'$$

$$A_2' \rightarrow c A_2' | ad A_2' | e$$

Now,

$$A_1 \rightarrow A_2 a | b$$

$$A_1 \rightarrow bd A_2' a | A_2' a | b$$

$$\left. \begin{array}{l} A_1 \rightarrow bd A_2' a | A_2' a | b \\ A_2 \rightarrow bd A_2' | A_2' \\ A_2' \rightarrow c A_2' | ad A_2' | e \end{array} \right\}$$

(i)  $A' \rightarrow \alpha'$

### LEFT-FACTORING

Start - if exp -then start else start /

if exp -then start

of 2 productions have common part  
of if exp then start:

start  $\rightarrow \alpha \beta_1 | \alpha \beta_2$  {be left factor. this  
is such  
is}

$$A \rightarrow \alpha \beta_1 | \alpha \beta_2 \rightarrow A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 | \beta_2$$

Decision of which production to choose is  
deferred for future.

# Top-down parser

- (i) left recursion is removed
- (ii) left factoring is done

$$(i)$$

$$\begin{aligned} E &\rightarrow F + T | T \\ T &\rightarrow T * F | F \\ F &\rightarrow (E) | id \end{aligned}$$

$$A. \quad T \rightarrow T * F \quad | \quad F \quad (\text{left recursion})$$

$$\alpha_1 \quad | \quad \beta_1$$

$$T' \rightarrow \alpha F T' | \epsilon$$

$$T \rightarrow F T'$$

EOC  
S

$$E \rightarrow E + T/T$$

$$f \rightarrow TE^1$$

$$F \rightarrow +TE'/\varepsilon$$

$$F \rightarrow (F) / \text{id}$$

(2)  $A \rightarrow Ba | c$

$B \rightarrow AA$

C → B / b

$$\text{Ans. } A_1 \rightarrow A_2 \cap A_3$$

$$A_2 \rightarrow A_1 A_1$$

$$A_3 \rightarrow A_2 \mid b$$

$$\text{Ag} \rightarrow \text{A}_1 \text{A}_1$$

下

$$A_2 \rightarrow A_2 a A_1 | A_3 A_1$$

1

$$A_8 \rightarrow A_3 A_1 A_2'$$

$$A_2' \rightarrow a A_1 A_2' | \varepsilon$$

$$A_3 \rightarrow A_2 / b \rightarrow A_3 \rightarrow A_3 \underbrace{A_1 A_2'}_{\alpha} \underbrace{b}_{\beta}$$

$$A_3 \rightarrow b A'_3$$

$$A_3' \rightarrow A_1 A_3' A_3' | \varepsilon$$

$$A_1 \rightarrow A_2 \alpha | A_3$$

$$A_8 \rightarrow A_3 A_1 | A_2$$

$$A_{\alpha'} \rightarrow a A_1 A_2' / \varepsilon$$

$$A_3 \rightarrow b A'_3$$

$$A_3' \rightarrow A_1 A_2' A_3' |\varepsilon$$





~~1.~~ S → start symbol ; FOLLOW(S) = { } & { }

2.  $A \rightarrow \alpha B \beta$   
FOLLOW(B) → first( $\beta$ ) ∪ FOLLOW(A)

if  $\beta$  contains  $\epsilon$

3.  $A \rightarrow \alpha B$   
then, FOLLOW(B) → FOLLOW(A)

4.  $A \rightarrow \alpha B \beta$   
(if, FOLLOW of B i.e. first( $\beta$ ) contains  $\epsilon$ )  
then FOLLOW(A) also.

PARSING TABLE

Terminals →

id

+

\*

(

)

\$

\$ E'T	id * id \$
\$ E'T'F	id * id f
\$ E'T'i\$	id * id i
\$ E'T'	* id \$
\$ E'T'F*	* id \$
\$ E'T'F	id \$
\$ E'T'f	id \$
\$ E'T'	\$
\$ E'T'	\$
\$ E'T'	\$

→ accepted string

✓ Grammar Generated is LL(1)

↓

left scanning is done      left most derivate      face 

→ Power      → no. of look ahead symbol

$X \rightarrow Yb$ 

Page No.

Date: / /

[ab]

Page No.

Date: / /

$$(a) \quad S \rightarrow [Sx] | a$$

$$X \rightarrow +S.Y | Yb | E$$

$$Y \rightarrow Sx.c | E$$

$$\text{FIRST}(S) = \{ [ , a ] \}$$

$$\text{FOLLOW}(S) = \{ \$, +, [ , a, b, ], c, \}$$

$$\text{FIRST}(X) = \{ +, E, [ , a, b ] \}$$

$$\text{FOLLOW}(X) = \{ ], c, ] \}$$

$$\text{FIRST}(Y) = \{ E, [ , a ] \}$$

$$\text{FOLLOW}(Y) = \{ ], b, ] \}$$

$$\begin{array}{ll} \$ & S \\ \$ & [Sx] \\ \$ & [dx] \\ \$ & X ] \\ \$ & Yb ] \\ \$ & B ] \\ \$ & X \end{array} \quad \begin{array}{ll} [ab] & \$ \\ [ab] & [ab] \\ [ab] & [ab] \\ b ] & \$ \\ b ] & \$ \\ b ] & \$ \\ \$ & \$ \end{array}$$

Parsing Table:

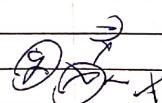
	[ ]	a	+	b	c	]	Y
S	$\rightarrow [Sx]$	$\rightarrow a$					
X	$X \rightarrow Yb$	<del><math>X \rightarrow a</math></del>	$X \rightarrow Yb$	$X \rightarrow +S.Y$	$X \rightarrow +S.Y$	<del><math>X \rightarrow E</math></del>	
Y	$Y \rightarrow Sx.c$	$Y \rightarrow E$	$Y \rightarrow Sx.c$	$Y \rightarrow E$	$Y \rightarrow E$		

Accepted.

\* IMP Rule :-

 $X \rightarrow \alpha \beta \dots$ 

if  $\alpha$  contains  $\epsilon$  then, we'll put  
this production in first of  
further terminals or terminals.  
non-



= 50

Premium

1118

## Questions

①.  $\subseteq \rightarrow A$

$$A \rightarrow aB | Ad$$

$B \rightarrow b\bar{B}C$  If

C → g

String ( $a b \bar{f} a$ )

$$\text{FIRST}(s) = \{a\}$$

Follow(s) = { } }

$$\text{FIRST}(A) = \{a\}$$

$\text{follow}(A) = \{d_1, d_2\}$

FIRST(B)  $\Rightarrow$  b,

Follow(B) = {g, d, \$}

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

$$\text{Follow}(c) = \{q_1, q_2, q_3\}$$

	a	d	b	c	f	g	h
S	$S \rightarrow A$	-	-	-	-	-	-
A	$A \rightarrow aB$	-	-	-	-	-	-
B	-	-	-	-	-	D	-
C	-	-	-	-	-	P2	-
Premium	-	-	-	-	-	P2	-
B	-	-	-	-	-	-	-
C	-	-	-	-	-	-	-

L-parser → left Recursion & left  
remove factoring

Page No.

Date : / /

$S \rightarrow A$

$A \rightarrow aB \mid Ad$

$B \rightarrow bBC \mid f$

$C \rightarrow g$

→ String (abfg)

$A \rightarrow Ad \mid aB$

$A' \rightarrow aBA'$

$A' \rightarrow dA'e$

$S \rightarrow A$

$A \rightarrow aBA'$

$A' \rightarrow dA' \mid E$

$B \rightarrow bBC \mid f$

$C \rightarrow g$

FIRST(S) = {a}

FIRST(A) = {a}

FIRST(A') = {d, e}

FIRST(B) = {b, f}

FIRST(C) = {g}

FOLLOW(S) = { $\$$ }

FOLLOW(A) = { $\$$ }

FOLLOW(A') = { $\$$ }

FOLLOW(B) = {d, g,  $\$$ }

FOLLOW(C) = {a, g, f}

a

b

d

f

g

$\$$

$S \rightarrow A$

$A \rightarrow aBA'$

$A' \rightarrow dA'$

$A' \rightarrow E$

$B \rightarrow bBE$

$B \rightarrow f$

## Spring:

abfg

Stack	
\$ S	abfg \$
\$ A	abfg \$
\$ A'Bd	abfg \$
\$ A'B	bfg \$
\$ A'CBb	bfg \$
\$ A'CB	fg \$
\$ A'CF	fg \$
\$ A'C	g \$
\$ A'g	g \$
\$ A'	\$
\$	\$

Qd.  $E \rightarrow uv | EBE | v | [E]$

$$\underline{V \rightarrow a/b}$$

U → </>

B → ? / ! / @

string : a @ alg

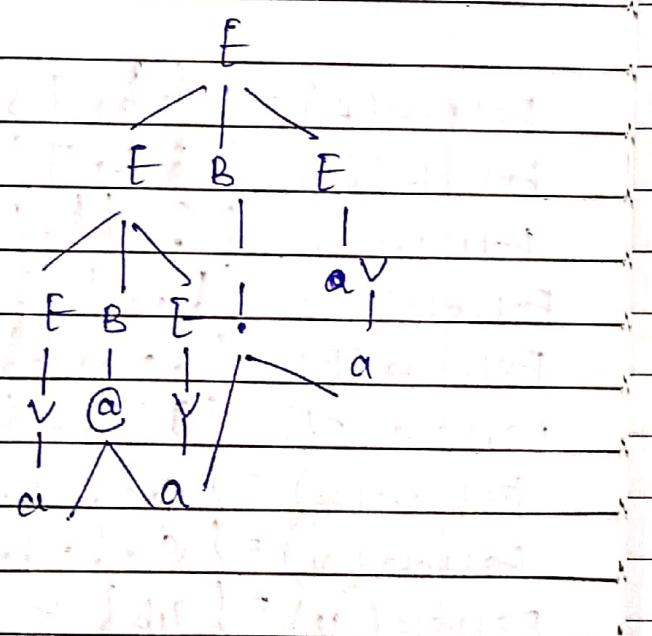
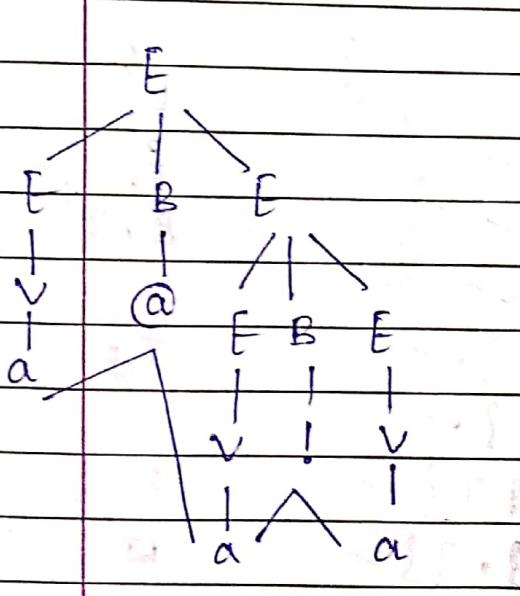
Ambiguous or not.

~~2 left max~~  $a@a|a$

$E \rightarrow EBE \rightarrow VBE \rightarrow aBE \rightarrow a@a$

$\oplus \rightarrow a@aEBE \rightarrow a@aVBE \rightarrow a@a@BE$   
 $\rightarrow a@a!E \rightarrow a@a!V \rightarrow [a@a!]$

$E \rightarrow EBE \rightarrow EBEBE \rightarrow VBEBE \rightarrow aBEBE$   
 $\rightarrow a@aEBE \rightarrow a@aVBE \rightarrow a@a@BE$   
 $\rightarrow a@a!E \rightarrow a@a!V \rightarrow [a@a!]$



Q3.  $E \rightarrow TE'$

$E' \rightarrow ?E|E$

$T \rightarrow FT'$

$T' \rightarrow !T|e$

$F \rightarrow WF'$

$F' \rightarrow @F|e$

$W \rightarrow UV|V| [F]$

$V \rightarrow ab$  or

$U \rightarrow </>$

$$\text{Follow}(E) = \{ \$ \} \quad \checkmark$$

E'

$$\text{Follow}(F') = \{ \$ \} \quad \checkmark$$

E'

$$\text{Follow}(T) = \{ 2, \$ \} \quad \checkmark$$

E, T

$$\text{Follow}(T') = \{ 1, \$ \} \quad \checkmark$$

T, P

$$\text{Follow}(F) = \{ !, 2, \$ \} \quad \checkmark$$

T, P

$$\text{Follow}(F') = \{ !, 2, \$ \} \quad \checkmark$$

T, P

$$\text{Follow}(W) = \{ @, !, 2, \$ \} \quad \checkmark$$

F

$$\text{Follow}(V) = \{ @, !, 2, \$ \} \quad \checkmark$$

W

$$\text{Follow}(U) = \{ a, b \} \quad \checkmark$$

$$\text{FIRST}(E) = \{ [, <, a, b, ?] \} \checkmark$$

$$\text{FIRST}(E') = \{ ?, !, @ \} \checkmark$$

$$\text{FIRST}(T) = \{ [, <, a, b, ?] \} \checkmark$$

$$\text{FIRST}(T') = \{ !, @ \} \checkmark$$

$$\text{FIRST}(F) = \{ [, <, a, b, ?] \} \checkmark$$

$$\text{FIRST}(F') = \{ @, ! \} \checkmark$$

$$\text{FIRST}(W) = \{ [, <, a, b, ?] \} \checkmark$$

$$\text{FIRST}(V) = \{ a, b \} \checkmark$$

$$\text{FIRST}(U) = \{ < \} \checkmark$$

$$\text{FOLLOW}(E) = \{ \$, ] \} \checkmark$$

$$\text{FOLLOW}(E') = \{ \$, ] \} \checkmark$$

$$\text{FOLLOW}(T) = \{ ?, \$, ] \} \checkmark$$

$$\text{FOLLOW}(T') = \{ ?, \$, ] \} \checkmark$$

$$\text{FOLLOW}(F) = \{ !, ?, \$, ] \} \checkmark$$

$$\text{FOLLOW}(F') = \{ !, ?, \$, ] \} \checkmark$$

$$\text{FOLLOW}(W) = \{ @, !, ?, \$, ] \} \checkmark$$

$$\text{FOLLOW}(V) = \{ @, !, ?, \$, ] \} \checkmark$$

$$\text{FOLLOW}(U) = \{ a, b \} \checkmark$$

	?	!	@	[	]	a	b
E							
E'	$E' \rightarrow ?E$					$E \rightarrow TE'$	$E \rightarrow TE'$
T						$E \rightarrow E$	
T'		$T' \rightarrow E$	$T' \rightarrow IT$		$T \rightarrow FT'$	$T \rightarrow FT'$	$T \rightarrow FT'$
F						$T \rightarrow F$	
F'	$F' \rightarrow E$	$F' \rightarrow F$	$F' \rightarrow @F$		$F \rightarrow WF'$	$F \rightarrow bF'$	$F \rightarrow bF'$
W						$F \rightarrow E$	
V					$W \rightarrow [F]$	$W \rightarrow V$	$W \rightarrow V$
U						$V \rightarrow a$	$V \rightarrow b$

a@a!a

Stack	I/P
\$E	a@a!a \$
\$E'T	a@a!a \$
\$E'T'F	a@a!a \$
\$E'T'F'W	a@a!a \$
\$E'T'F'V	a@a!a \$
\$E'T'F'A	a@a!a \$
\$E'T'F'I	@ a!a \$
\$E'T'F@	@ a!a \$
\$E'T'F	a!a \$
\$E'T'F'W	a!a \$
\$E'T'F'V	a!a \$
\$E'T'F'A	a!a \$
\$E'T'F'I	!a \$
\$E'T'	!a \$
\$E'T!	!a \$
\$E'T	a \$
\$E'T'F	a \$
\$E'T'F'W	a \$
\$E'T'F'V	a \$

<	>	\$	\$E'T'F'@	a \$
$E \rightarrow TE'$	$E \rightarrow TE'$		$\$ E'T'F'@$	\$
			$\$ E'T'F'I$	\$
			$\$ E'T'$	\$
$T \rightarrow FT'$	$T \rightarrow FT'$		$\$ E'$	\$
			$\$ T$	\$
$F \rightarrow WF'$	$F \rightarrow WF'$			
			$\$$	\$
$W \rightarrow UV$	$W \rightarrow UV$			
$U \rightarrow C$	$U \rightarrow$			

string  
accepted

Q.4  $S \rightarrow X/b$  ] left recursion.  
 $X \rightarrow S/a$

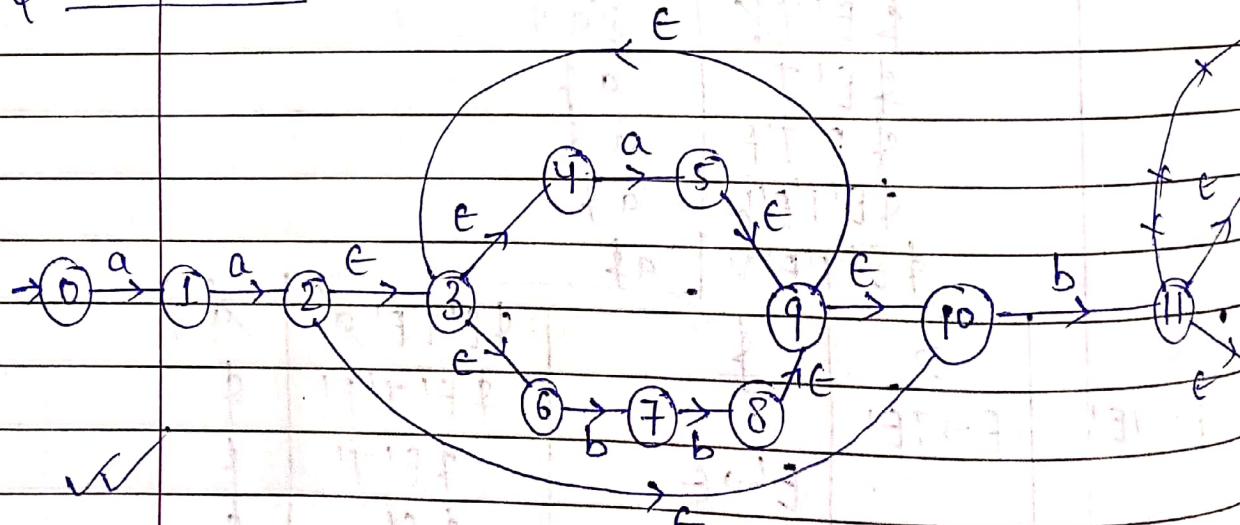
✓ Q.5.  $aa(a|bb)^*b(a|b)$  — Thompson

~~LL-Parse~~ :-

Q.6 // SLR, 1.  $F \rightarrow id(P);$   
 $P \rightarrow PF \mid id \mid id$

2.  $T \rightarrow B \mid dL$   
 $L \rightarrow TL \mid B$   
 $B \rightarrow a \mid b.$

Q.7. THOMPSON :-



✓  $[aa(a|bb)^*b(a|b)]$

DFA  
from NFA  
direct algo (Parse)  
direct from mind

Page No.

Date: / /

Q4 = Left-Recursion

$$S \rightarrow X \mid b$$

$$X \rightarrow S \mid a$$

$$A_1 \rightarrow A_2 \mid b$$

$$A_2 \rightarrow A_1 \mid a$$

$$A_1 \rightarrow A_2 \mid b$$

$$A_2 \rightarrow A_2 \mid b \mid a$$

$$\begin{aligned} A_1 &\rightarrow A_2 \mid b \\ A_2 &\rightarrow b \mid a \end{aligned}$$

