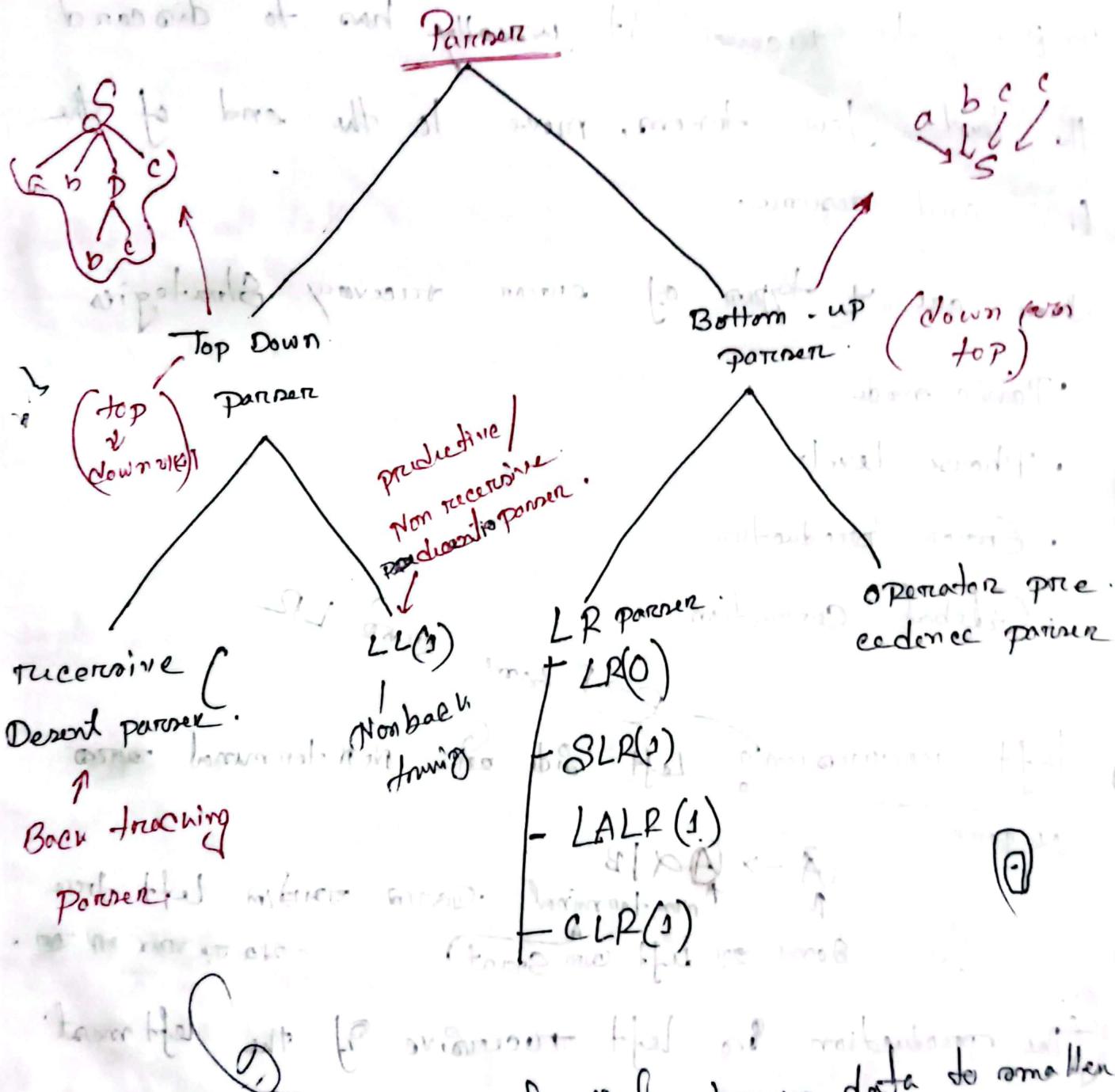


## Segment 4

## Syntax analysis

Complex

5



a parser or a component that breaks data into tokens  
elements for easy translation into other language

## # Error Recovery:

when a syntax error occurs, in order for the compiler to recover, it usually has to discard the last few tokens, move to the end of the line and resume.

there are 4 types of error recovery strategies

- Panic mode
- Pharse Level
- Error production.
- Global Correction

left recursion

LR parser

some  
1st

(A)  $\rightarrow$  (A)  $\alpha | \beta$   
non-terminal  
some 2nd left side start

some  
(0) 1st

some  
(2) 2nd

LR

some  
1st

some  
2nd

left tree

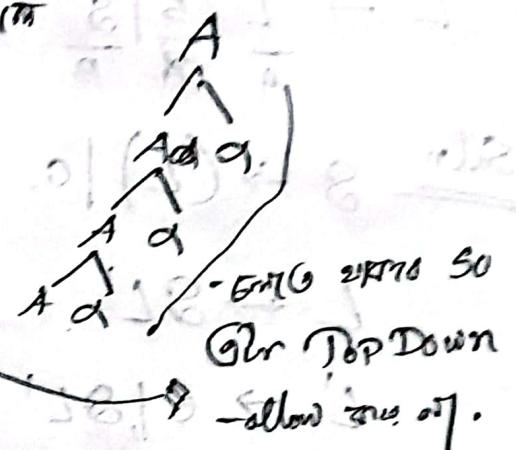
→ to start

If the production is left recursive of the left most symbol on the right side is the same as the non-terminal on the left side (left recursion).

Symbol on the right side is the same as the non-terminal on the left side (left recursion).

# Top Down Parsing LR(0) allows loops because infinite

loop occurs like  $A \rightarrow A\alpha \text{ etc}$



### Elimination of Left Recursion

$A \rightarrow A\alpha \mid \beta$

LR eliminate  $\Rightarrow$

multiple substitution rule

$$\begin{array}{l} \text{Goto } A \rightarrow BA' \\ \text{Goto } A' \rightarrow \epsilon \mid dA' \end{array}$$

converges to

then Goto  $A' \rightarrow \epsilon \mid dA'$   
now  $A' \rightarrow \epsilon$

Example \*  $E \rightarrow E + T \mid T$

multiple substitution rule

Sln:  $E \rightarrow TE'$

$E' \rightarrow E \mid TE'$

\*  $T \rightarrow FT \mid F$

Sln:  $T \rightarrow FT$

$T' \rightarrow E \mid FT$

\*  $A \rightarrow ABd \mid Aa \mid a$

$B \rightarrow Be \mid b$

Sln:  $A \rightarrow AA'$

$A' \rightarrow E \mid BdA' \mid aA'$

\*  $E \rightarrow E + E \mid EXF \mid a$

$E \rightarrow aE'$

$E' \rightarrow E \mid EE' \mid XEE'$

\*  $S \rightarrow (L) | a.$   $\xleftarrow{\text{with } G^{\text{RDL}}}$  grammar must get #

$L \rightarrow \frac{L \cup S}{A} | \frac{S}{B}$   $\xleftarrow{\text{so }} SA \leftarrow A$

Saln  $S \rightarrow (L) | a.$

or make  $L \Rightarrow SL$

resultant  $\Rightarrow$

in  $SL \Rightarrow E, SL'$

converted to  $E$  to reduce

- Advantages of LR

Left Recursion.  
 $A \mid B \leftarrow A$

$\Rightarrow LR$  grammar is more often intuitive than.

The transformed grammar

A left recursive grammar

earlier  $T \mid T \leftarrow T$

$T \mid T \leftarrow E$  \*  $\xleftarrow{\text{expr}}$   
will match expression

$E \mid E : T$

$E \mid E : T$

advantages of the benefits

Bottom-up Parsing takes

of left recursion.

$E \mid E : T$

$A \mid A : E$

$A \mid A : E$

$A \mid A : E$

## Left factoring Top Down parser S.o.

If common prefix factoring out prefix which are common

two or more production rules. If both sides common  
not common

$A \rightarrow a\beta | \gamma$   $\beta \in F$   $\gamma \in F$

Example:  $S \rightarrow iEto | iEto a$   $i$  is common

\*  $S \rightarrow iEto | iEto a$

$E \rightarrow b$

derivation = (bottom) top \*

Am  $S \rightarrow iEto | iEto a$

Am  $E \rightarrow T+E | T$   $\leftarrow A$

$E' \rightarrow E | E$  (optimal ob.)

Am  $E \rightarrow T+E | E$   $\leftarrow A$

$E \rightarrow b$

Am  $E \rightarrow T+E | E$   $\leftarrow A$

where  $b$  is left rigid & none  $x$  is  $x$

LR parser of bottom up are type of bottom up parser that

efficiently handle deterministic Context free language in

bottom up form (A) bottom up parser

Time complexity is  $O(n^2)$  where  $n$  is size of input string

Format and follow [function](#)

Ruler 1955-1961 N.Y.D.

\* final (Terminal) = terminal

$A \rightarrow a b F f G$     (a) Terminal    switch over (overwriting) front

\* If  $x \rightarrow E$  Then  $E$  <sup>and</sup>  $\in$  to  $\text{front}(x)$  | ?  $\leftarrow$  '8

$A \times B \rightarrow E$  അംഗീകാരം ഇല്ല ആവശ്യമാണ് അംഗീകാരം എന്ന് പറയാം

~~ear~~ X သာမ် X နှေ့ကြမ် + Sing. Put -ခံ only.

\*  $A \rightarrow AF$ ,  $F \rightarrow a/F$  or  $F \rightarrow b/F$  so (A) find {a, b, F}

→ एक non-terminal ( $A$ ) का इन सभी वर्तमान रूपों का संग्रह उत्तम रूप से बदलने के लिए आवश्यक है।

यह तभी Non-terminal वाले पद को  $G_{T(A)}$  कहते हैं।

\* यदि एक व्यवस्था जिसके अन्तर्गत  $S \rightarrow a, S \rightarrow b, S \rightarrow c$  होता है। तो इसका फ्रंटेन्ट यह होता है।

Example:  $S \rightarrow abc | def | ghi$  जिसका फ्रंटेन्टेन्ट क्या होगा?

हमें  $S \rightarrow \{a\} \cup S \rightarrow \{d\} \cup S \rightarrow \{g\}$

जिसका फ्रंटेन्टेन्ट  $\{a, d, g\}$  होगा।

\*  $S \rightarrow ABC$

$A \rightarrow a | b | \epsilon$

$B \rightarrow c | d | \epsilon$

$C \rightarrow e | f | \epsilon$

Ans

फ्रंटेन्टेन्ट  $(S) \Rightarrow$  फ्रंटेन्टेन्ट  $(ABC)$

$\Rightarrow \{a, b\} \cup \{c, d\} \cup \{e, f\} \cup \{\epsilon\}$

$f(A) \rightarrow \{a, b, \epsilon\}$

$\Rightarrow \{a, b, c, d, e, f, \epsilon\}$

$f(B) \rightarrow \{c, d, \epsilon\}$

$\Rightarrow \{a, b, c, d, e, f, \epsilon\}$

$f(C) \rightarrow \{e, f, \epsilon\}$

$\Rightarrow \{a, b, c, d, e, f, \epsilon\}$

$E \rightarrow TE'$   
 $E' \rightarrow *TE'E'd$   
 $T \rightarrow FT'$

$T' \rightarrow E \mid FT'$   
 $F \rightarrow id \mid CE$

$\{a \in \Sigma \cup \{b\} \cup \{c\}$   
 $\{d \in \Sigma \cup \{b\} \cup \{c\}$

Joint  $E \rightarrow TE'$  {our Non terminal so posse  
generate our language}

$\Rightarrow \{a, b, c\}$

Joint  $E' \rightarrow *TE'E'd$

$\Rightarrow \{* \}$

Joint  $T \rightarrow FT'$   
 $\Rightarrow \{a, b, c\}$

$DFA \leftarrow A$   
 $\Rightarrow \{d\} \leftarrow A$   
 $\Rightarrow \{b\} \leftarrow B$   
 $\Rightarrow \{a\} \leftarrow C$

Joint  $T \rightarrow E \mid FT'$   
 $\Rightarrow \{a, b, c\}$

Joint  $F \rightarrow id \mid CE$

$\Rightarrow \{id, \{ \} \}$

+ other terminal  
 parenthesis.

## Follow

-rule:

- Extent symbol  $\text{G}(\text{A})$ , କ୍ଷେତ୍ର ପରିମା ହିନ୍ଦି.
  - ଅଳ୍ପାଣି  $\text{G} \rightarrow \text{FT}$  ରୁମ୍ ଏବଂ  $\text{G}$  କେବଳ  $\text{FT}$  ରୁମ୍  
 $\text{F} \rightarrow \text{ST}$

$F \rightarrow ST$   
 ଏହା କାମ କରିବାରେ ଏହାକିମ୍ବା ଏହାକିମ୍ବା ଏହାକିମ୍ବା  
 ଏହା first କିମ୍ବା Line Example ଏହା  $T \Rightarrow$  first  $T \leftarrow A$



১৯৮৫

- ফুল ছানা terminal পেরি<sup>m</sup> direct রসা ৫ → FTA

then follow writing side and (ink) in road

- Non-terminal symbols + products  $\xrightarrow{\text{rule}} \text{A} \rightarrow A^m$   $\rightarrow$   $A^n \rightarrow \dots$   $\rightarrow$   $A^1 \rightarrow a$   
 start follow  $\rightarrow$   $a \in L(G)$

## Example

\*  $S \rightarrow AaAb|BbBa$ .

$A \rightarrow E$

$B \rightarrow E$

$$(A\text{no.}) \Rightarrow f_0(A) = \{a, b\}$$

$$f_0(B) = \{b, a\}$$

$S \rightarrow AB|C$

$A \rightarrow DEF$

$B \rightarrow E$

$C \rightarrow E$

$$\Rightarrow f_0(A) = \{D\} \text{ initial } G \text{ no. } (S) \text{ no. } (E)$$

Follow adding follow first to T

Production	First	Follow
$E \rightarrow TE'$	$\{(), id\}$	$\{(), \$\}$ set by default
$E \rightarrow +TE'   \epsilon$	$\{+, \epsilon\}$	$\{\$\}$ $E'$ no. for $\epsilon$ $\epsilon$ is $E'$ no. for $\epsilon$
$T \rightarrow FT'$	$\{id\}$	$\{+, \$, id\}$
$T' \rightarrow *FT'  \epsilon$	$\{*id\}$	$\{+, \$, *\}$
$F \rightarrow (E)   id$	$\{(id)\}$	$\{*, +, \$, id\}$

## Recursive Descent parser

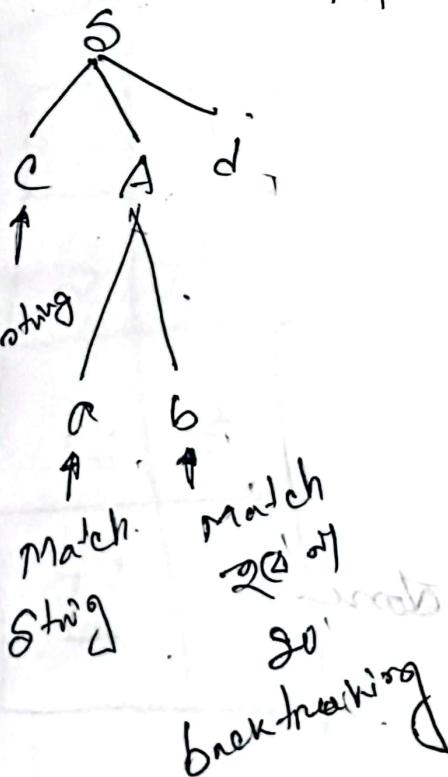
It is a set of recursive procedures to scan input. It involve back-tracking so repeated scan of a to input. [For production we move top down or go right first]

Example  $S \rightarrow CAd$   
 $A \rightarrow ab|a$

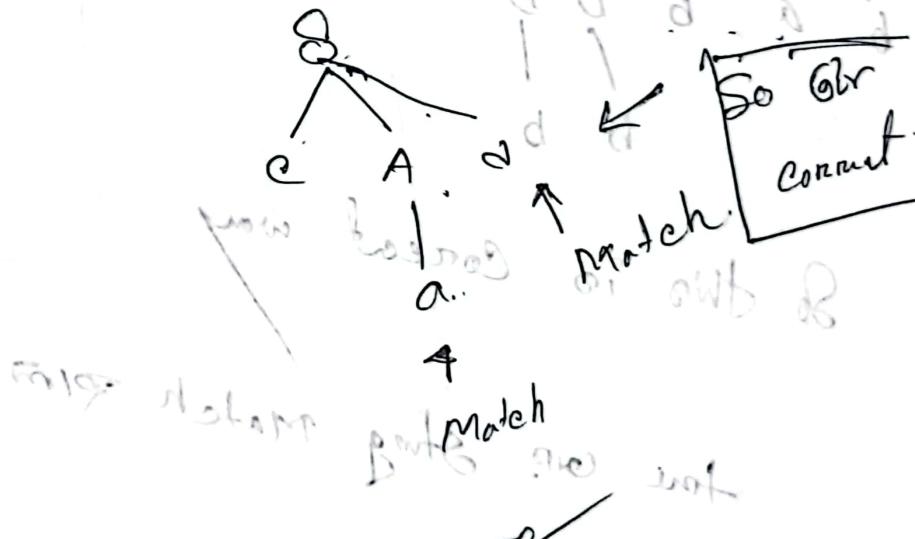
here 2 pointers String start point

Step 1.

Input string = cad.



Step 2 → backtracking



Ques How could a top down parser decide which production for S leads me to derive babb?

$S \rightarrow AB|CD$

$A \rightarrow BC|CA$

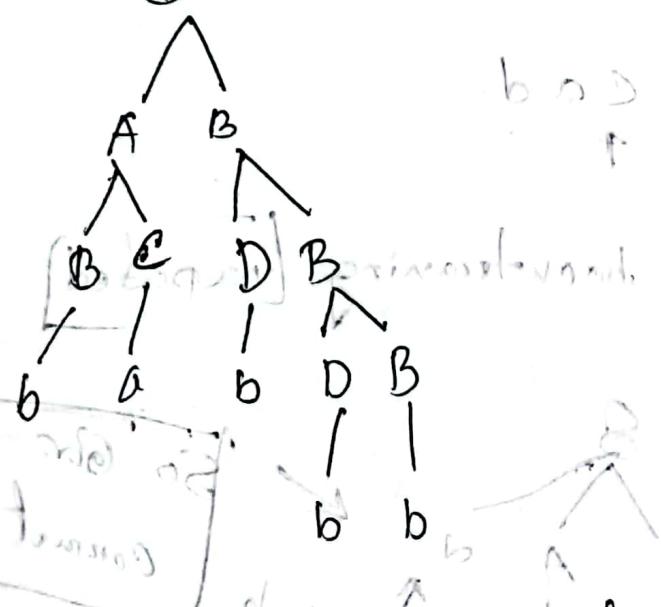
$B \rightarrow CA|DB$

$C \rightarrow BA|AD$

$D \rightarrow AC|BD$

Input string babb.

→ S



So this is correct way

free corp. string

match  $\Rightarrow$  done

front end

## LL Parsing / Predictive parser

An LL parser parses the input from left to right (L) and uses a left-most derivation (L).

→ Find follow sets first then Left Variable (Scope domain)

### Example

$$S \rightarrow CL \quad | \quad a \rightarrow \{ C, a \} \quad \text{first} \quad \text{follow: } \{ ., , ) \}$$

$$L \rightarrow SL' \quad \rightarrow \{ C, a \} \quad \rightarrow \{ ) \}$$

$$L' \rightarrow \epsilon \quad | \quad SL' \quad \rightarrow \{ \epsilon, \} \quad \rightarrow \{ ) \}$$

Total S's production → parse table

	(	)	a	,	)
S	1		2		
L	3		3		
L'		4		5	

$L' \rightarrow \epsilon$   
order L' follow  
order

①. production.

Given  $S \rightarrow (L)$

1

Given find

$\{ C \}$  so search  
in  $\{ C \}$

So go to LL parsing

$$S \rightarrow \overbrace{a S b}^1 \mid \overbrace{b S a}^2 \mid \overbrace{\epsilon}^3$$

first  $\rightarrow \{a, b\}, \epsilon\}$

follow  $\rightarrow \{b, a, \epsilon\}$

	a	b	$\epsilon$
S	1 3	2 3	3

$$S \rightarrow \epsilon \text{ (in)}$$

S fa follow

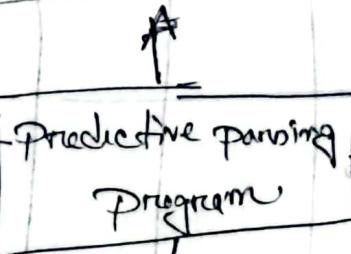
So it is not LL grammar after step

Show the LL parser model (non recursive products)

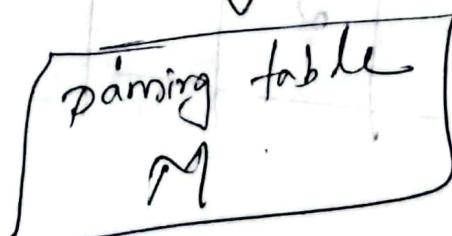
Input	a	+	b	\$

X
Y
Z
\$

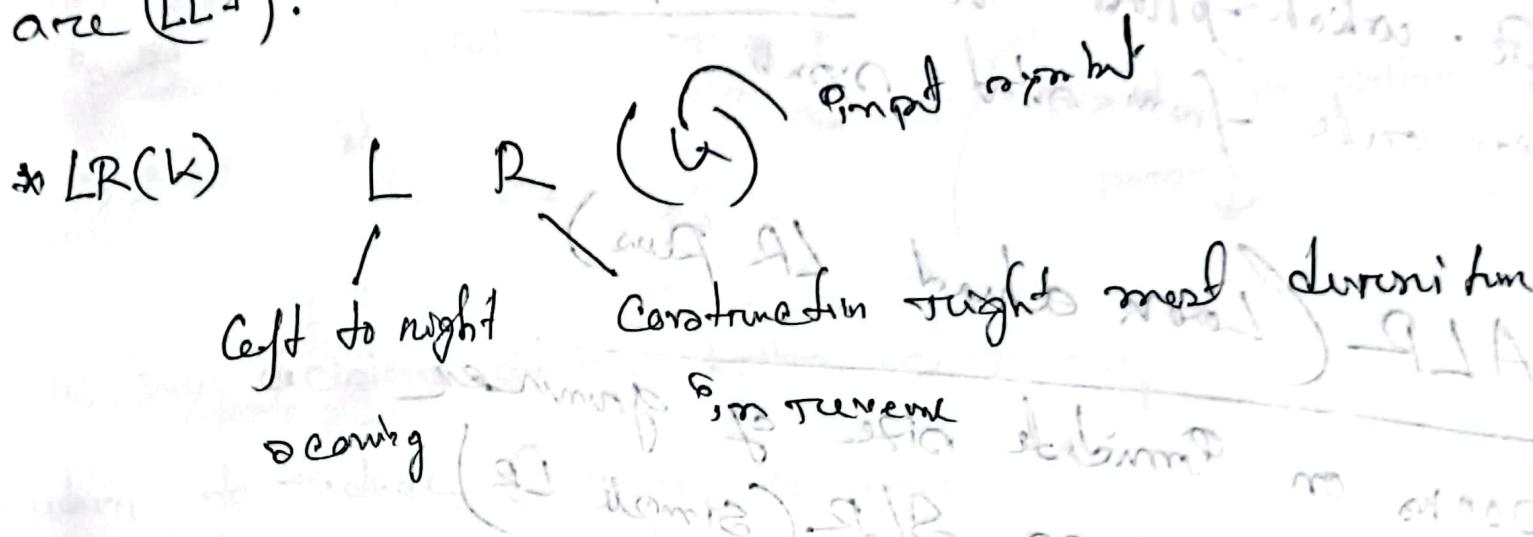
a	+	b	\$



Output



L(4): It's called Non-recursive parser. The basic idea of a non-recursive parser is to use the current input symbol to decide which alternative to choose. Grammar which have the property are  $LL^1$ .



Top down: It is the parse tree is generated in the top down (ie from root to leaves). (Left Most)

Bottom up: It's a strategy for parsing sentence of CFG that attempts to construct parse tree leaf to root.

~~20-3~~ is a set of demands that have  
been drawn from a ①

Answer: The act of binding of general  
agent after the non demand is a mistake.  
General agent can't bind his agent.

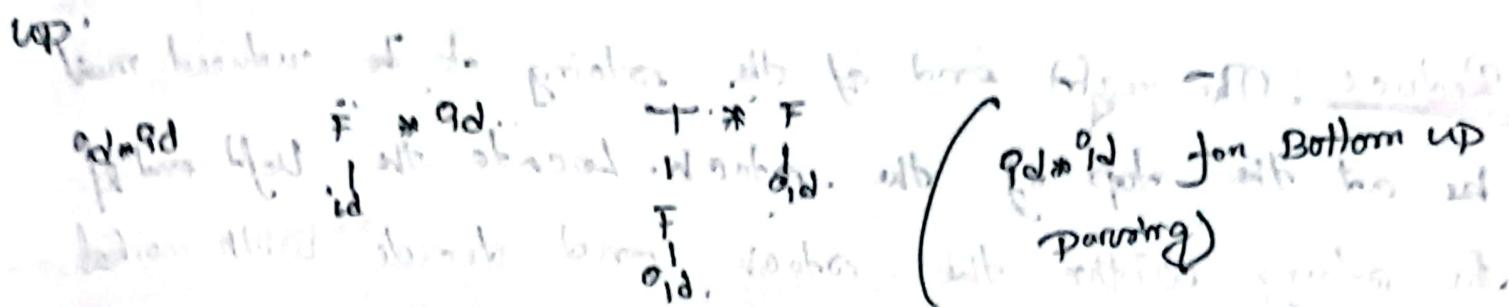
LAMP (Lamp shade lamp)

ALL small on the ground  
water in the SLT (small LL)  
water small on the SLT

## Segment 3 Bottom up Parsing

### Bottom up Parsing

Bottom up parser traverses the parse tree from bottom up.



# The key decisions during bottom up parsing.

- when to reduce (Producing string)
- what production to apply.

# Shift-reduce Parsing is a form of bottom up parsing.

- a stack holds grammar symbols.
- a input buffer holds the rest of the string to be parsed.
- parser reads (nothing until) detected error on stack contains start symbol.

# Shift reduce parser

Shift: Shift the next input symbol onto the top of the stack.

Reduce: The right end of the string to be reduced must be at the top of the stack. Locate the left end of the string within the stack and decide which non-terminal to replace the string.

Accept: Announce successful completion of Parsing.

Error: Discover a syntax error and call an error function.

Terminology + outline

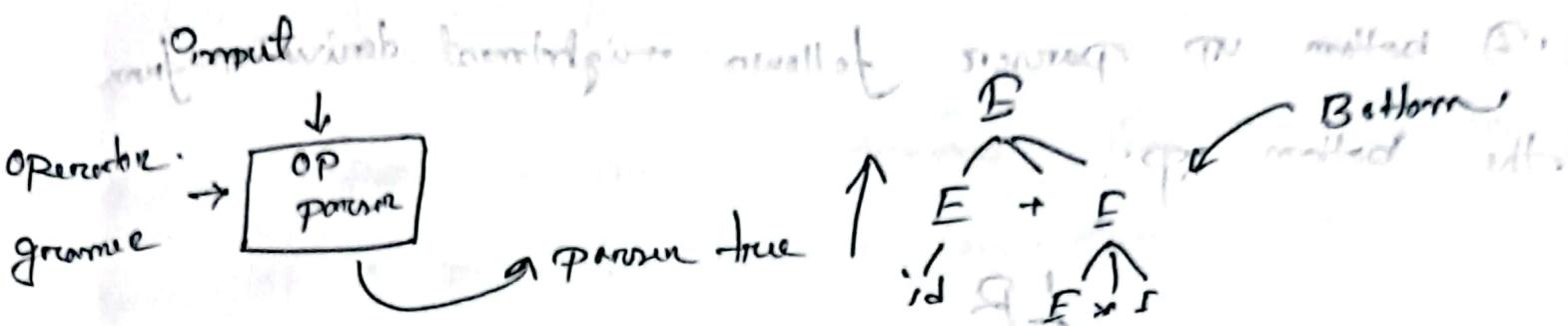
parser on method to make an parser object that

# Shift reduce parser is a process of breaking down a string into tokens according to grammar.

[RHS (Right hand side) of a production rule is replaced by the symbol on LHS (left side of production) along with some tokens.

Example - LR Parsing

## Operation precedence grammar



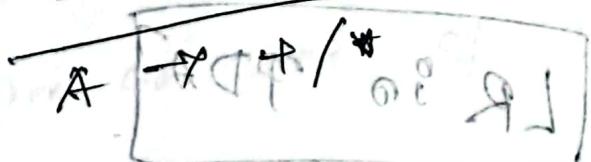
operator grammar  $E \rightarrow E+E | E \cdot E | id$

(i) No epsilon in right side.

(ii) no adjacent variable.

$E \rightarrow id | E+E | id$  ← operator precedence grammar

$E \rightarrow EAE$  ← Non-terminal



So convert

$E \rightarrow E+E | E \cdot E | id$

## LR Parser

- ① bottom up parser follows rightmost derivation from the bottom up.



proc tokens left      rightmost derivation on tokens  
to right      order

b{ E+E } I A B  
tokens are

- LR parsing → The production is applied only after the pattern has been matched.

- In LL(Preactive) Parsing, the production was selected and other matching tokens were matched to it.

- ② In LR parser

- A parse table

- An input buffer

- A stack of LR(0) states.

LR  $\Rightarrow$  PDA

LR parser performs three types of operations:

- Shift: A token from the input buffer to the stack.
- Reduce: Pop sequence of tokens that matches the body of a production and replace them with the head of the production.
- GoTo: a new state.

\* LR(0): o look ahead reduces as soon as ~~as soon as~~ <sup>as soon as</sup> the body of a production is matched by next tokens.

SLR (Simple LR): Depending on the next token

LR(1) parser: 1 (look ahead): incorporates the next token as part of the current "state" of the process.

LALR (Look ahead LR) is an LR(1) parser with its states much consolidated.

## LR(0) 0-term

It is a production with a special marker (•) that marks a position within the body of the production.

- Build LR parse table, and find  $\text{LR}(0)$  0-term

E<sub>0</sub>

$$E \rightarrow E + T$$

The variable  $\text{LR}(0)$  0-term.

$$E \rightarrow \cdot E + T, E \xrightarrow{\cdot} E \cdot + T, E \rightarrow E \cdot T$$

#  $[A \rightarrow \alpha \cdot B]$  so we have processed  $\alpha$  and we might process  $B$  next

we will categorize : (backed) | : error | : accept

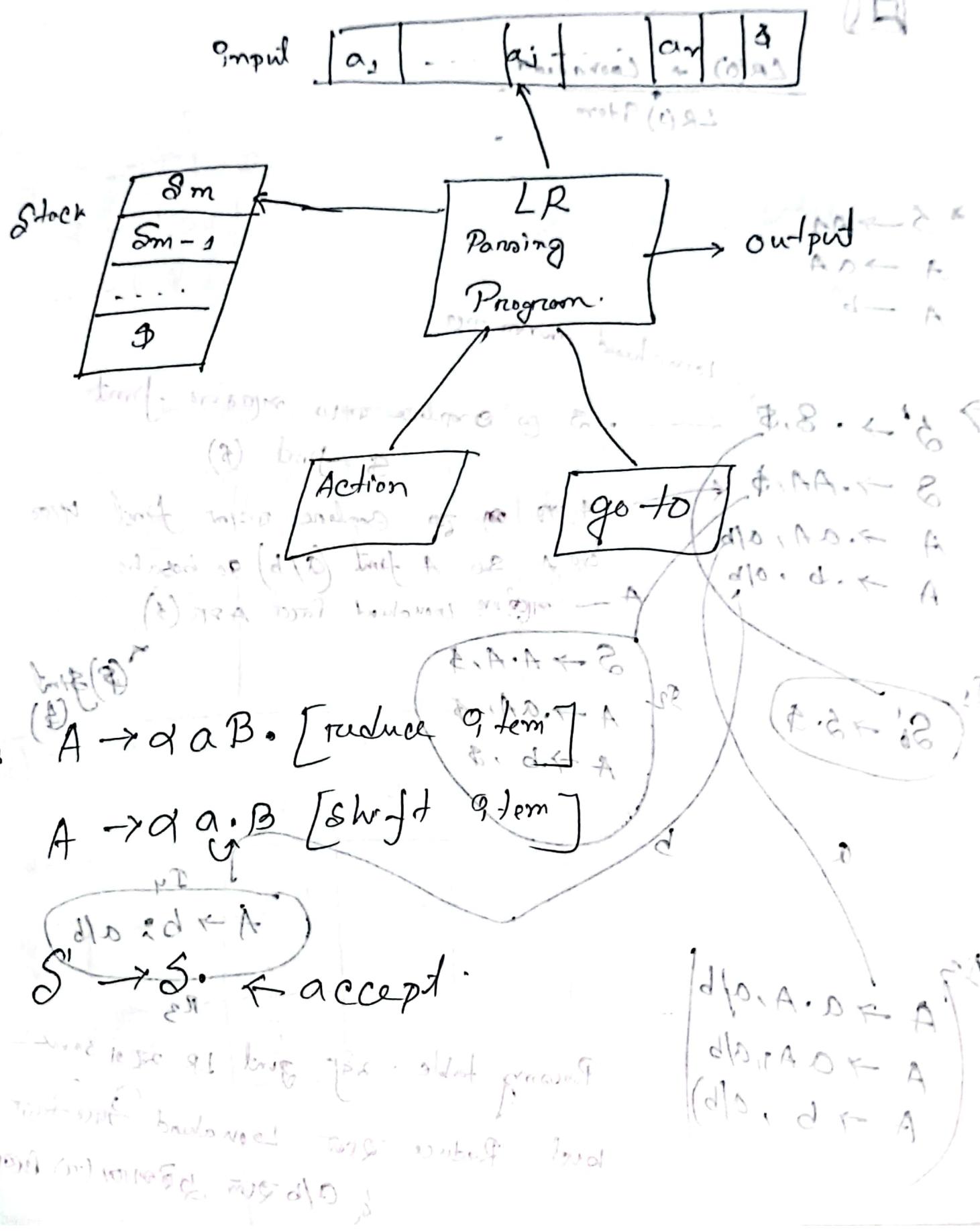
We can build a PDA whose states are sets of LR(0) 0-term.

After running (0.9) we get (all backs removed) all back errors

## LR Parsing

- Step 0
- First we will change the grammar w/ the new added symbol.
- $[S' \rightarrow \cdot \alpha]$
- Initial state  $\varnothing_0$  called  $\varnothing_0$  (0-term 0)
- State  $\varnothing_0$  is the closure of the set of  $S' \rightarrow \cdot \beta$ .
- for each item  $[A \rightarrow \cdot \beta]$  set each product  $B \rightarrow \cdot \gamma$  in grammar add the item  $[B \rightarrow \cdot \gamma]$ .
  - We call  $[B \rightarrow \cdot \gamma]$  an initial  $B$ -item.
- The left side will never appear  $AB \cdot A$  because there is no  $B$  after  $A$ .  
 Before appearing  $AB \cdot A$  either  $A$  is terminal or  $A$  is non-terminal.

## LR Parsing Model



## CLR(0) Parsing

E

$LR(0)$  + Lookahead  
 $\xrightarrow{LR(1) \text{ item}}$

$S \rightarrow AA$   
 $A \rightarrow aA$   
 $A \rightarrow b$

$\Rightarrow S' \rightarrow .S, \$$

$S \rightarrow .AA, \$$

$A \rightarrow .aA, a/b$

$A \rightarrow .b, a/b$

$\xrightarrow{\text{Lookahead } \text{incom. tree}}$

$S \rightarrow .S$  go explore tree  $\xrightarrow{\text{first}} \$ \text{ find } (\$)$

$A \rightarrow .aA$  so  $A \text{ first } (a/b) \text{ go back to } A \Delta R(\$)$

$A \rightarrow .b$   $\xrightarrow{\text{incom. tree}} \text{lookahead } \text{for } A \Delta R(\$)$

I<sub>1</sub>

$S'_0 \rightarrow S \cdot \$$

a

S<sub>2</sub>

$S \rightarrow A \cdot A, \$$   
 $A \rightarrow .aA, \$$   
 $A \rightarrow .b, \$$

b

(\\$) find  
 (a)  
 (b)

I<sub>4</sub>

$A \rightarrow b; a/b$

$\begin{cases} A \rightarrow a \cdot A, a/b \\ A \rightarrow a A, a/b \\ A \rightarrow b, a/b \end{cases}$

Parsing table wif first and LR same

level Reduce  $\xrightarrow{\text{out}}$  Lookahead  $\xrightarrow{\text{out}}$   
 { a/b  $\xrightarrow{\text{out}}$  reduction ( $\pi$ ) level }

shift-reduce parsing

LL(1)arsing

#

$$A \rightarrow Q$$

$$Q \rightarrow S+R$$

$$S \rightarrow S_0 + E$$

$$R \rightarrow Q R + E$$

String  $\Rightarrow "Q Q Q Q R"$

①	$A \rightarrow Q$	$\rightarrow \{ S, E, +, R \}$	$\rightarrow (\$)$
②	$Q \rightarrow S + R$	$\rightarrow \{ S, E, +, R \}$	$\rightarrow (\$)$
③	$S \rightarrow S_0 + E$	$\rightarrow S, E$	$\rightarrow (+)$
④	$R \rightarrow Q R + E$	$\rightarrow Q, E$	$\rightarrow (R)$

(LR) parsing

S

+

Q

R

\$

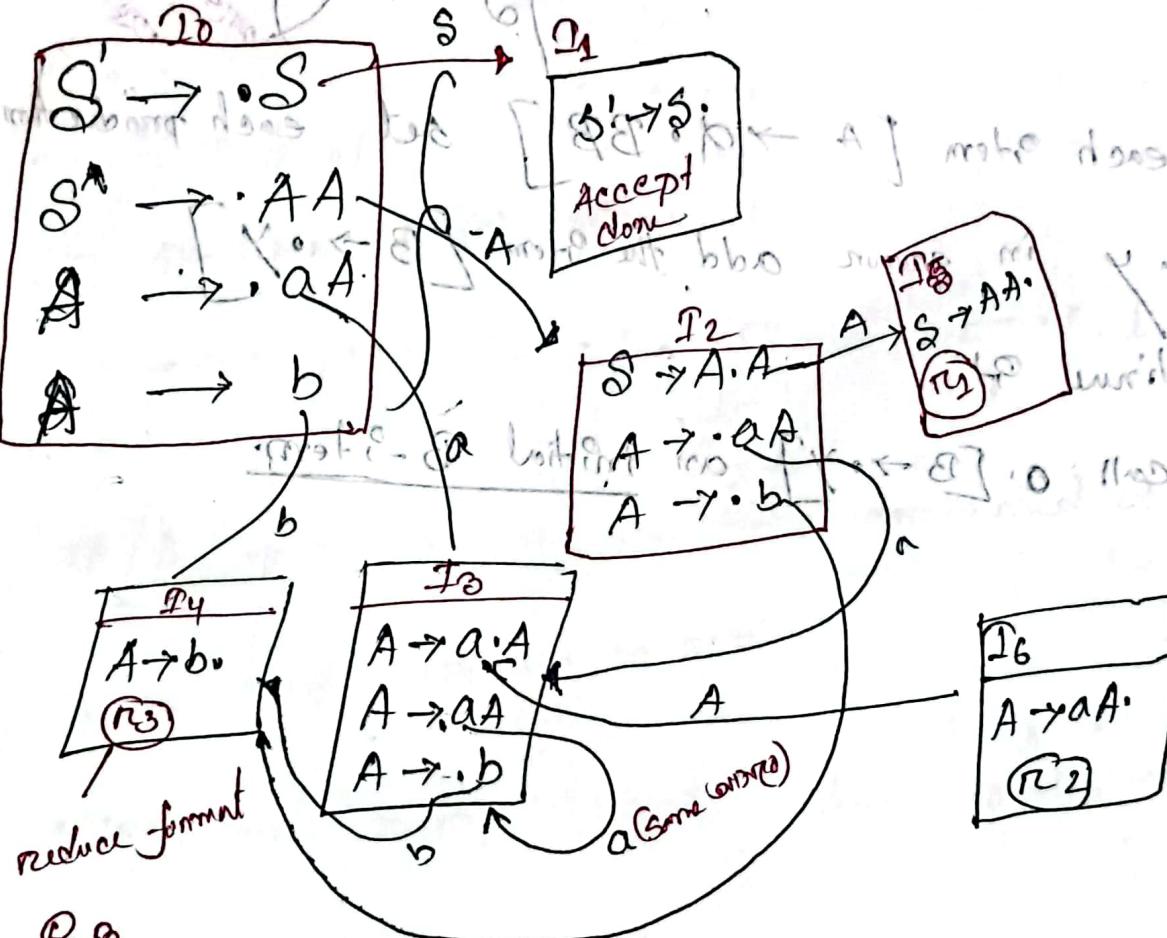
A	I	I	I	I		
Q	I	I	I	I		
S	I	I				
R			ii	ii	ii	

# Check if the given grammar is LR(0) or not

$$\begin{aligned} S &\rightarrow AA \leftarrow \text{Production 1} \\ A &\rightarrow aA \leftarrow 2 \\ A &\rightarrow b \leftarrow 3 \end{aligned}$$

[Shift and reduce  
Same table of LR(0)  
[LR(0)]]

- Start symbol  $S \rightarrow \cdot S$  (initial)
- New start symbol  $S' \rightarrow S$
- Same set of symbols



R3

↑  
Producing  
① ②

P3

Note:  $\cdot A A$  can be derived from  $A \cdot A$  or  $A \cdot A \cdot$   
 since  $\cdot A$  can be derived from  $A$  by Production ①  
 (i.e.) same process used  
 $\cdot A A \cdot, b \cdot$  reduce format of (i.e.)

## LR(0) Parsing table

States	Action	Go to
$I_0$	a $\rightarrow \delta_3$ , b $\rightarrow \delta_4$ , $\epsilon \rightarrow \epsilon$	$I_1$ , $I_2$ , $A$
$I_1$	<del>b <math>\rightarrow \delta_1</math></del> , $\epsilon \rightarrow \epsilon$	accept
$I_2$	$\delta_3$ , $\delta_4$	$I_3$
$I_3$	$\delta_3$ , $\delta_4$	$I_4$
$I_4$	$R_3$ , $R_3$ , $R_3$ Action reduce current to box $\rightarrow$	$I_5$
$I_5$	$R_1$ , $R_1$ , $R_1$	$I_6$
$I_6$	$R_2$ , $R_2$ , $R_2$	

So here Don't arrange Shift reduce on same line

So do LR(0) Parsing

## Example 2 LR(0) on mat

$$E' \rightarrow E$$

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

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

$$F \rightarrow (E) \mid id$$

=

$$\boxed{F \rightarrow (E.) \\ E \rightarrow E \cdot T}$$

$$\begin{array}{l}
 I_0 \\
 \hline
 F \rightarrow (\cdot E) \\
 E \rightarrow \cdot E + T \\
 E \rightarrow \cdot T \\
 T \rightarrow \cdot T * F \\
 T \rightarrow \cdot F \\
 F \rightarrow \cdot (E) \\
 F \rightarrow id
 \end{array}$$

$$\begin{array}{l}
 I_0 \\
 \hline
 E' \rightarrow \cdot E \\
 E \rightarrow \cdot E + T \\
 E \rightarrow \cdot T \\
 T \rightarrow \cdot T * F \\
 T \rightarrow \cdot F \\
 F \rightarrow \cdot (E) \\
 F \rightarrow id
 \end{array}$$

$$\begin{array}{l}
 I_1 \\
 \hline
 E \rightarrow E + T. \\
 T \rightarrow T \cdot * F
 \end{array}$$

$$\begin{array}{l}
 I_{10} \\
 \hline
 T \rightarrow T \cdot F
 \end{array}$$

$$\begin{array}{l}
 I_2 \\
 \hline
 E \rightarrow E! \\
 E \rightarrow E \cdot T
 \end{array}$$

$$\begin{array}{l}
 I_6 \\
 \hline
 E \rightarrow E + \cdot T \\
 T \rightarrow \cdot T * F \\
 T \rightarrow \cdot F \\
 F \rightarrow \cdot (E) \\
 F \rightarrow \cdot id
 \end{array}$$

$$\begin{array}{l}
 I_2 \\
 \hline
 E \rightarrow T. \\
 T \rightarrow T \cdot * F
 \end{array}$$

$$\begin{array}{l}
 I_7 \\
 \hline
 T \rightarrow T \cdot * F \\
 F \rightarrow \cdot (E) \\
 F \rightarrow \cdot id
 \end{array}$$

$$\begin{array}{l}
 I_3 \\
 \hline
 T \rightarrow F.
 \end{array}$$

$$\begin{array}{l}
 I_5 \\
 \hline
 F \rightarrow id.
 \end{array}$$

Stack	Input	Action
\$	<u>4444 R\$</u>	
\$ A - 1st production over.	4441 R\$	A → Q (cred)
<u>\$ @Rn</u>	<u>4444 R\$</u>	Q → Rn
\$ 4Rn	"	R → <u>Qn</u>
<del>4444 R\$</del>	R\$	
Rn	( \$ )	
	( \$ )	
Gn	( \$ )	
\$	\$	Accept
	S	
	F	
	L	
	A	
	D	
	B	
	C	

Sept 6.

## Syntax Directed Translation

### Syntax Directed Definition (SDD):

It is a context-free grammar with attributes added to the grammar symbols.

$SDD \rightarrow CFG + Semantic\ rules.$

SDD each node has:

- a set of synthesized attributes
- a set of inherited attributes

child to parent

synthesized attributes.

E.g., value at a

parent note can be determined.

from of its children]

$A \rightarrow B C$

Syn -  $A \Rightarrow B.b$   
 $B \Rightarrow C.c$

b.a.b →

$\Sigma \rightarrow ABC$

A value of its parent

to its sibling (value

of its parent)

B.C. sibling

[It determined from the value of attributes at the sibling and Parent, and it will replace those that have that much]

# Production (20) Semantic rules

## Syntax Directed definition

### Production

$$L \rightarrow E_n$$

$$E \rightarrow E_1 + T$$

$$E \rightarrow + \text{ (digit)} \quad \text{E.val = E1.val + T.val}$$

$$T \rightarrow T_1 * F$$

$$T \rightarrow F \quad \text{T.val = F.val above does not change}$$

$$F \rightarrow (E)$$

$$F \rightarrow \text{digit}$$

### Semantic rules

$$L.\text{val} = E.\text{val} \text{ of } L$$

$$E.\text{val} = E_1.\text{val} + T.\text{val} \text{ of bubbles}$$

$$E.\text{val} = T_1.\text{val} * F.\text{val}$$

$$T.\text{val} = T_1.\text{val} * F.\text{val}$$

$$T.\text{val} = F.\text{val} \text{ above does not change}$$

$$F.\text{val} = E.\text{val} \text{ below does not change}$$

$$F.\text{val} = \text{digit}.1 \text{ (below does not change)}$$

# Syntax directed definition zero annotated.

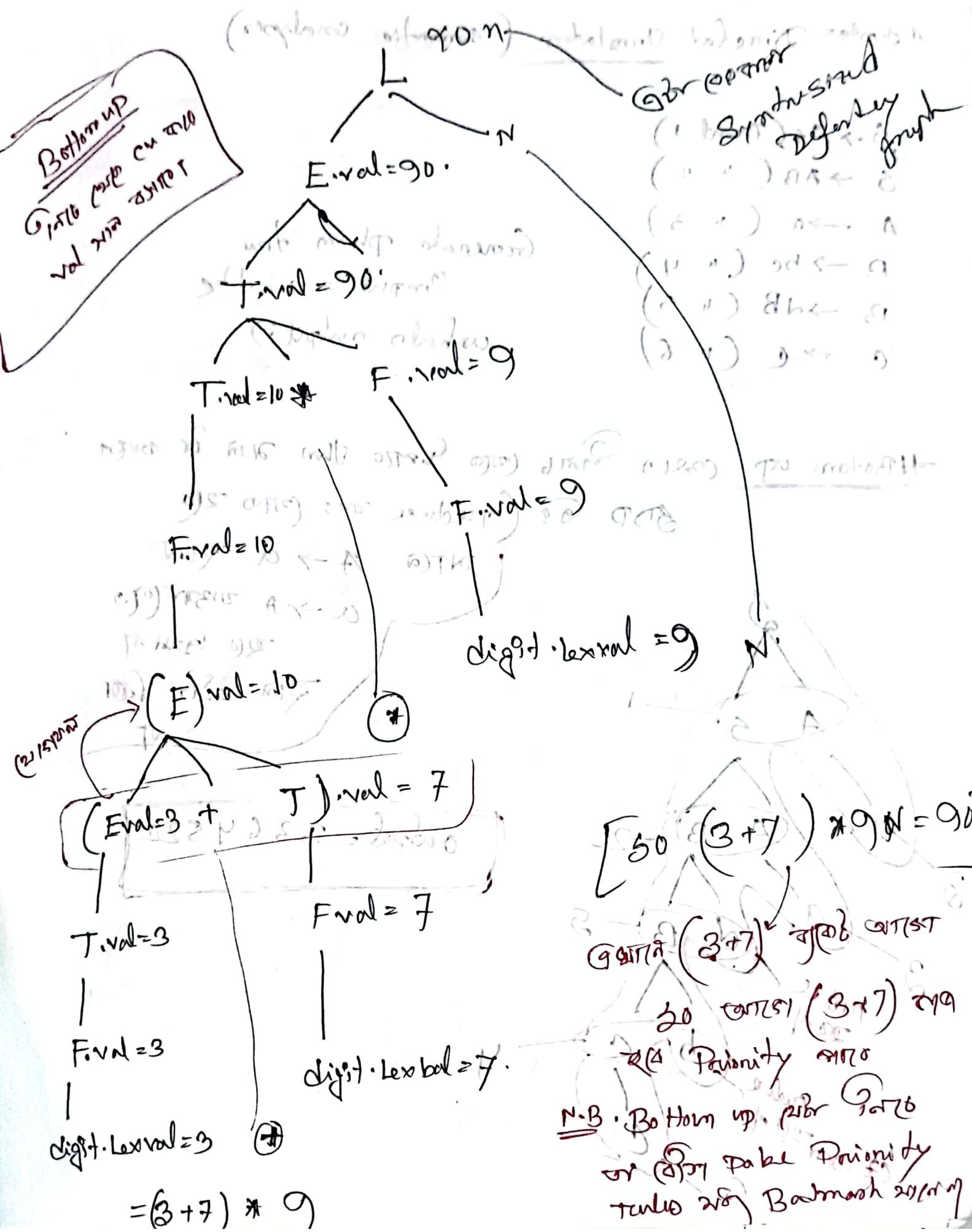
### Parentheses handling

produces previous example ③ 2DD 200 → to above step (P)

at most bracketed (E) with 9n by annotated due to nesting

NB: Strength nesting before factor design

nesting (G) step to next



# Syntax Directed Translation (Semantic analysis)

$S \rightarrow As$  (Print 1)

$S \rightarrow AB$  (" 2 )

$A \rightarrow a$  (" 3 )

$B \rightarrow bc$  (" 4 )

$B \rightarrow dB$  (" 5 )

$C \rightarrow c$  (" 6 )

Generate plan from tree

Input =  $a \text{ and } bc$

What's output?

Bottom up

Output =  $a \text{ and } bc$  from 2nd or 3rd

SDP (reduce zero) (2nd - 2nd)

ATC (A  $\rightarrow a$ ) (2nd - 2nd)

$a \rightarrow A$  (2nd - 2nd)

Q (Goto)

2nd = (Goto) factor

$F = \text{last. T}$

Output = 3 3 6 4 5 2

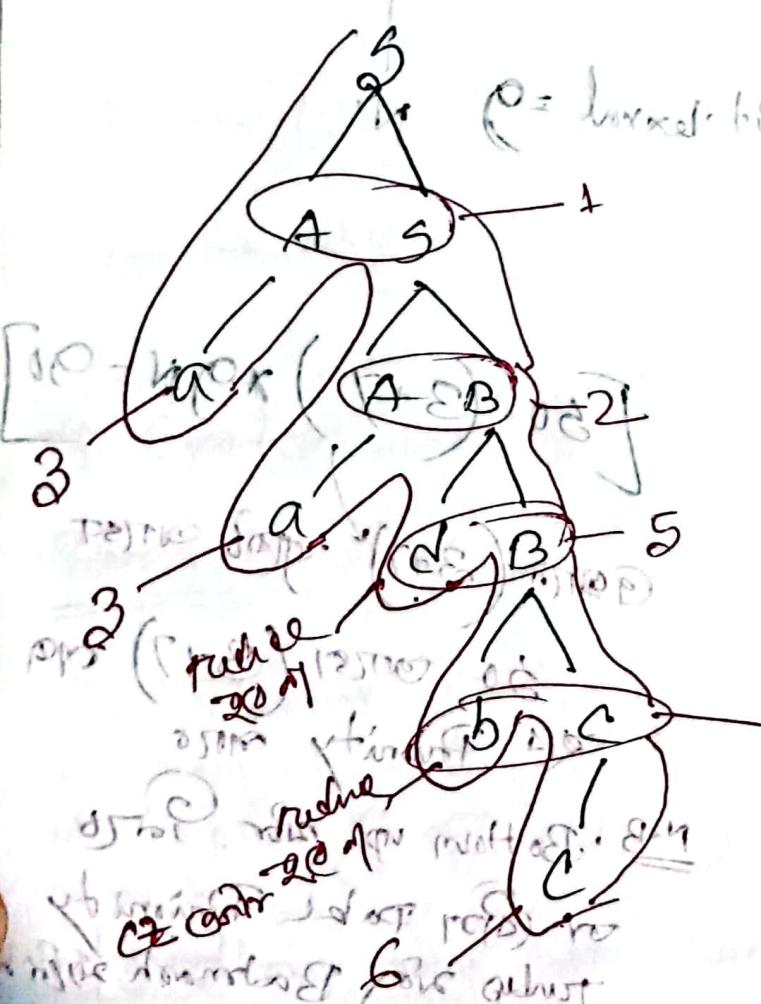
$F = \text{last. T}$

$E = \text{last. T}$

$G = \text{last. T}$

$E = \text{last. T}$

$P * (F + G) =$



$\# E \rightarrow E \& T \{ E.\text{val} = \{ E.\text{val} \rightarrow T.\text{val} \} \}$

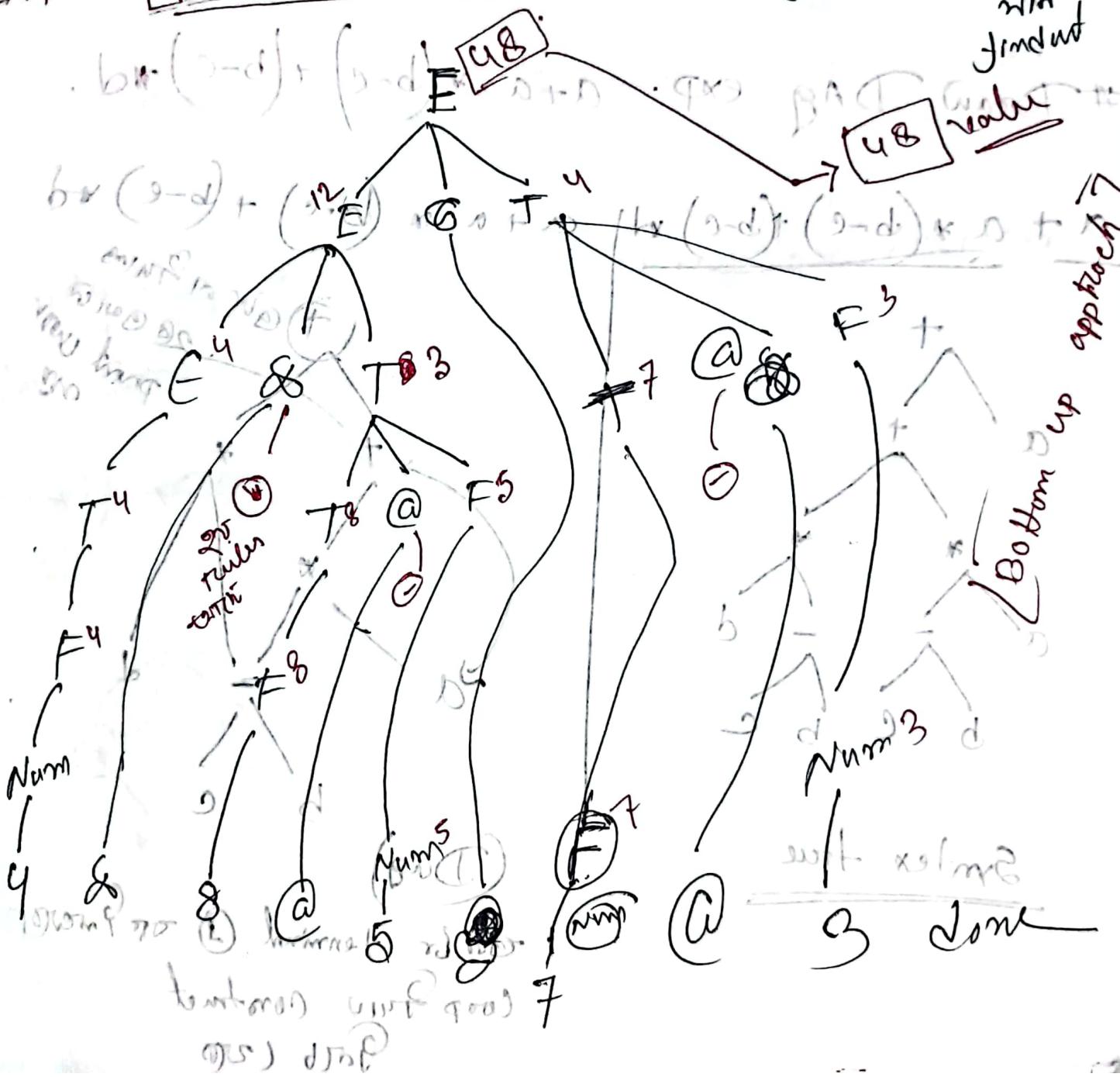
$E \rightarrow T$  of  $\text{Eval} = T.\text{val}$

$$T \rightarrow T_{\text{real}} \quad F_{\text{real}} = T_{\text{real}} - T_{\text{ideal}}$$

$$T \rightarrow F \quad \left\{ \begin{array}{l} T \text{ val} = F \cdot \text{val} \\ \end{array} \right\}$$

$F \rightarrow$  Num of F. rat = number of 1D components of output?

Input  $\rightarrow$  4 @ 28 @ 5 & 7 @ 3.



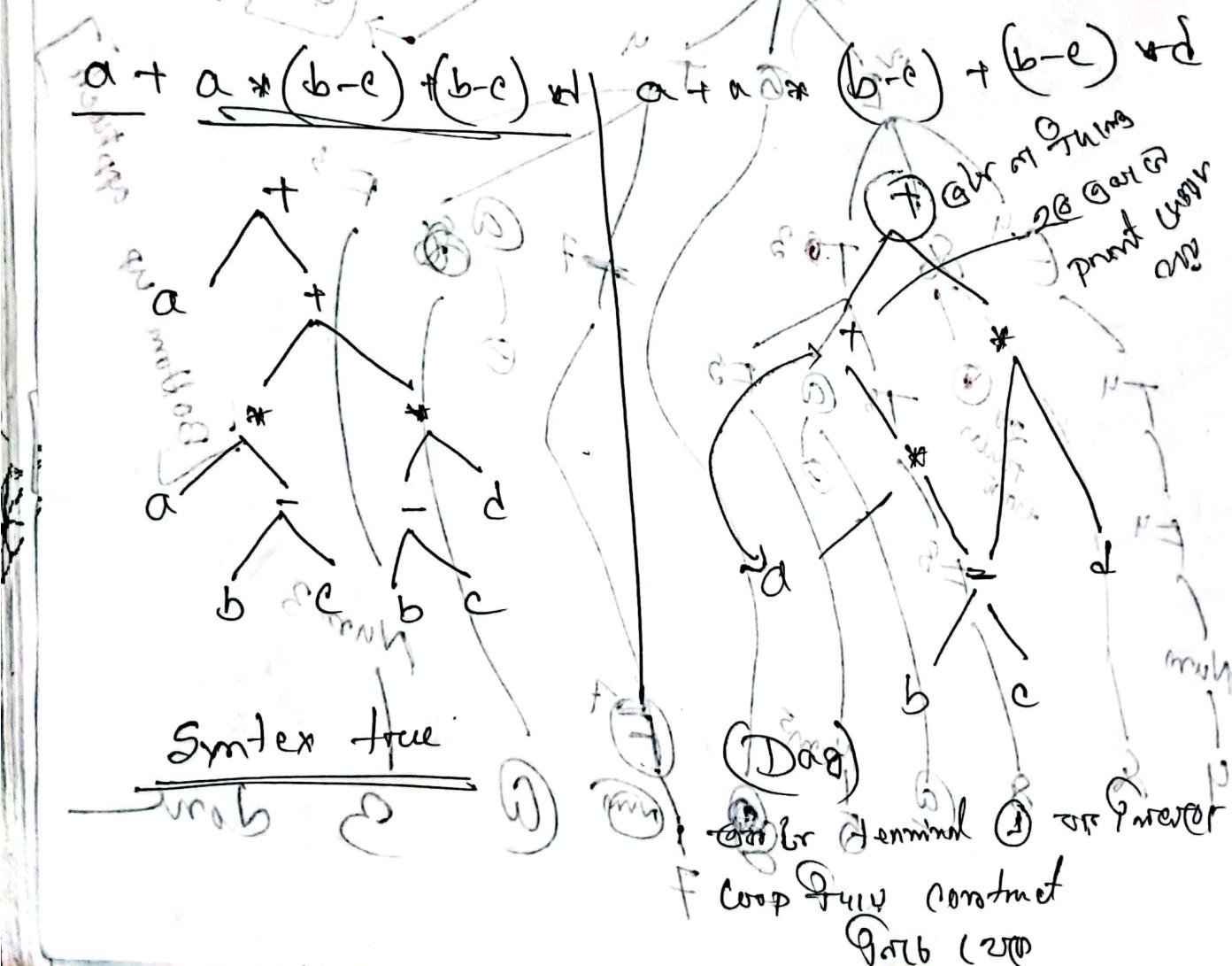
## #DAG (Directed Acyclic Graph)

#A DAG for an expression identifies the common sub expression in the expression. (Ans (250-723))

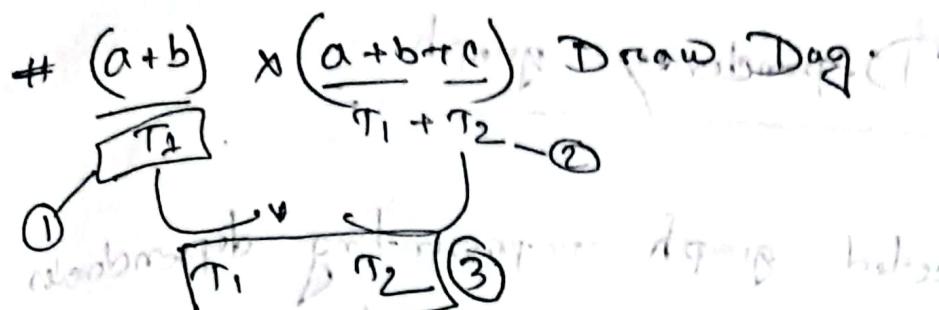
Syntax tree: It is constructed from a parse tree

useful for representing language constructs.

# Draw DAg exp:  $a+a * (b-c) + (b-c) * d$



# Badminton 27/2  
① 6/181 Pranavneel

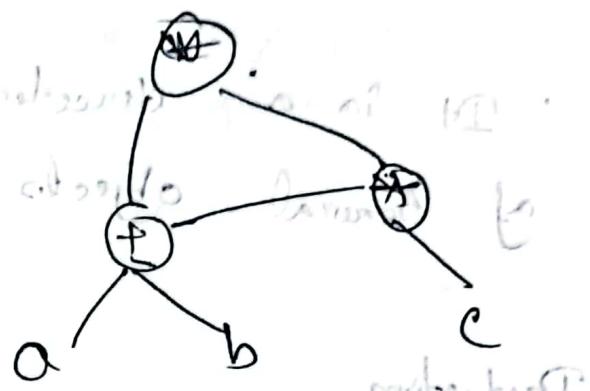


প্রাপ্তি এবং

$$T_1 = a+b$$

$$T_2 = b+c$$

$$\text{Now } T_3 = T_1 \times T_2$$

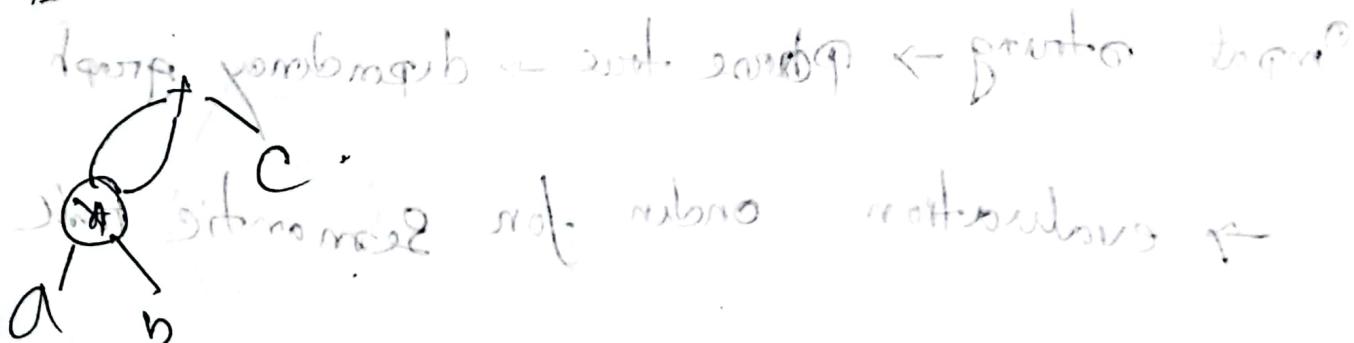


# DAG গুরুত্বের অন্তর্ভুক্ত করে দিনের স্পেস ফুরু

DAG গুরুত্বের অন্তর্ভুক্ত করে দিনের স্পেস ফুরু

জন্ম সমে পার্লে গ্রাফ রুল!

$\frac{(a+b)}{T_1} \times ((a+b) + c)$  হ'ল এই কারণে



## Dependency (graph + O) x (d+n)

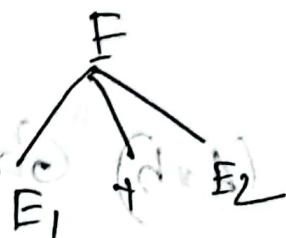
- It is a directed graph representing dependencies of general objects towards each other

Production

$$E \rightarrow E_1 + E_2$$

Semantic rule

$$E \cdot \text{val} = E_1 \cdot \text{val} + E_2 \cdot \text{val}$$



$A \rightarrow BCD \leftarrow \text{Synthesized}$   
BCD are A components

$A \rightarrow B(CD) \wedge A(A(C))$   
B,D are parts of A

- Conceptual view of STD(SDT)

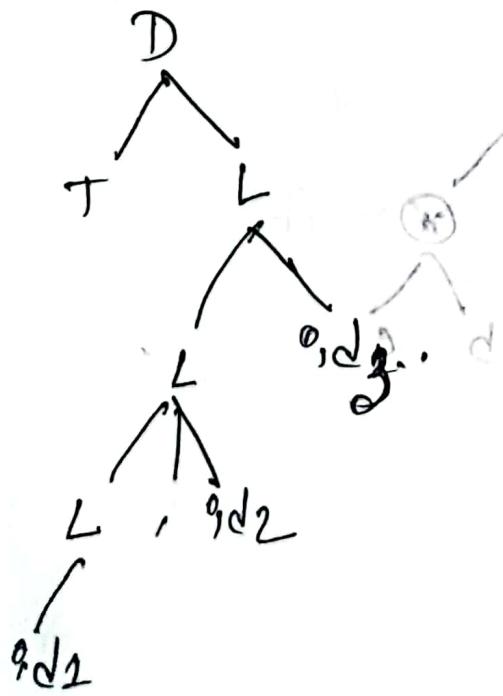
Input string  $\rightarrow$  parse tree  $\rightarrow$  dependency graph

$\rightarrow$  evaluation order for semantic rule

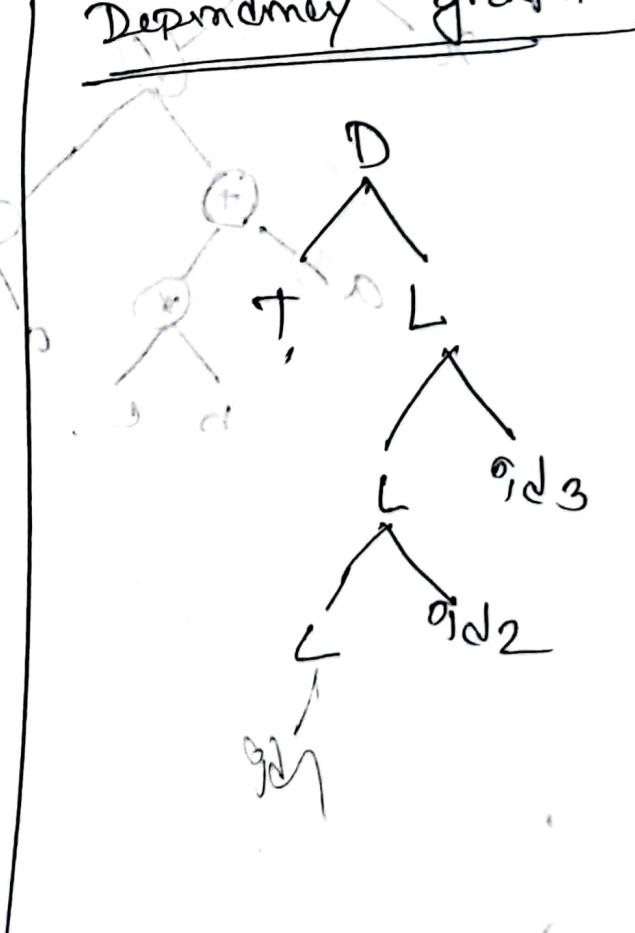
~~Ques.~~  
 Ques.  $D \rightarrow TL$   
 $T \rightarrow \text{int} / \text{real}$   
 $L \rightarrow L_1, \text{id}_1 | \text{id}_2$   
 Find semantic rule.  
 Draw noted parse tree with inherited attributes and dependency graph for real  $\text{id}_1, \text{id}_2, \text{id}_3$  by grammar.

Semantic Rules:  
 $D \rightarrow TL \quad \left\{ \begin{array}{l} D_{\text{real}} \rightarrow T_{\text{real}} + L_{\text{real}} \\ T_{\text{real}} = \text{int\_val}, \quad T_{\text{real}} = \text{real\_val} \end{array} \right\}$   
 $T \rightarrow \text{int} | \text{real} \quad \left\{ \begin{array}{l} T_{\text{real}} = \text{int\_val}, \quad T_{\text{real}} = \text{real\_val} \\ T_{\text{int}} = \text{int\_val} \end{array} \right\}$   
 $L \rightarrow L_1, \text{id}_1 | \text{id}_2 \quad \left\{ \begin{array}{l} L_{\text{real}} = L_1_{\text{real}} + [\text{id}_1], \quad \text{id}_1 = \text{real\_val} \\ L_{\text{int}} = L_1_{\text{int}} + [\text{id}_1], \quad \text{id}_1 = \text{int\_val} \end{array} \right\}$   
 Parse trees for the sentence  $\text{real } \text{id}_1, \text{id}_2, \text{id}_3$  using

Parse Inherited tree :



Dependency graph



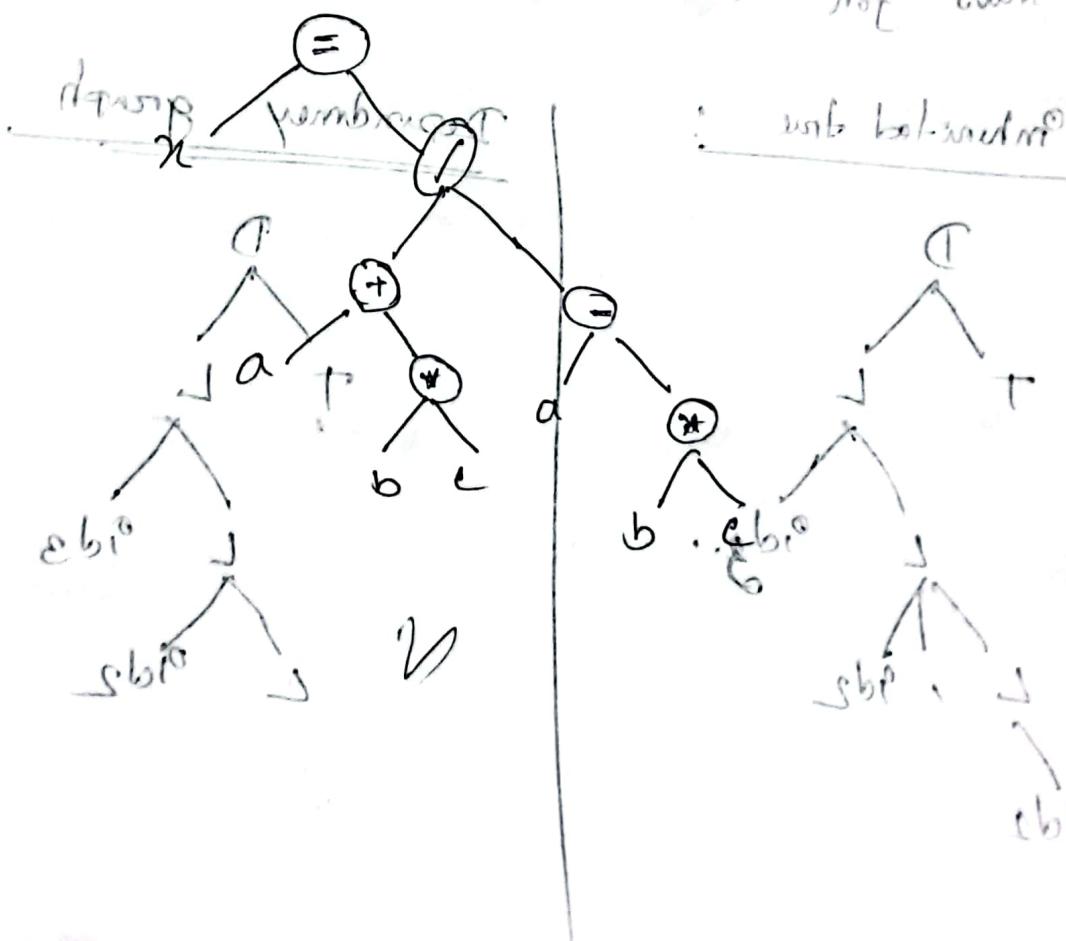
# Intermediate Code generation

not strong verbosen  
 Source code  $\rightarrow$  Intermediate representation  $\rightarrow$  Target Code  
 memory  $\times$  codegen, chip, loop generation

Commonly used Intermediate Code for representation

- ① Syntax tree
- ② Postfix
- ③ Directed Acyclic Graph
- ④  $\beta$  Address Code

Syntax tree:  $= \{ a + (b * c) \} / (a - (b * c))$



## Postfix Notation

$$a+b \rightarrow ab+ \text{ (Postfix)}$$

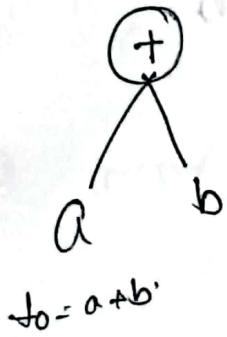
$$\# (a+b)*c \Rightarrow \boxed{\text{PN}} \rightarrow ab+c*$$

$$\# (a-b)* (c+d) + (a-b) \Rightarrow ab-cd+ab-$$

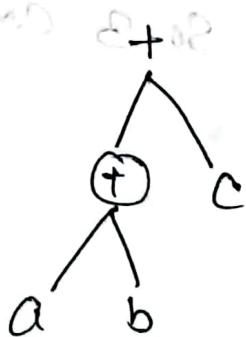
$$\# ((a+b)*c)*(c-d) \Rightarrow ab*c \cancel{+ da} \cancel{+ da} \cancel{-}$$

DAG:  $a+b+c$

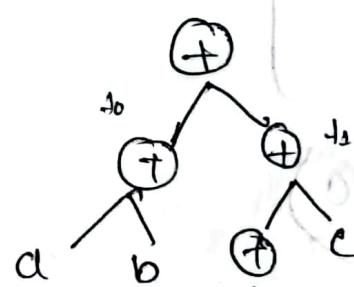
$$J_0 = a+b, J_1 = J_0 + c \quad d = J_0 + J_1$$



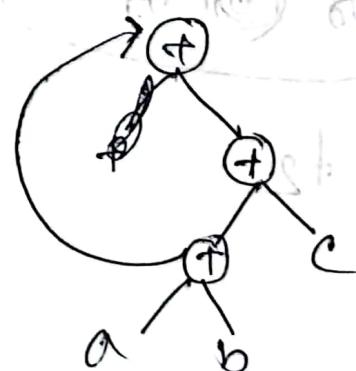
$$J_0 = a+b$$



$$J_1 = J_0 + c$$



↓  
should move



### 3 Address code

\*\*

$$A = b + c * d.$$

let  
 $t_1 = c * d.$

$$t_2 = b + t_1.$$

$$A = t_2.$$

of 3 br general address generation

Example

$$n = \frac{(a+b)}{2} + \frac{(c+d)}{4} \quad [ \text{3 br with 3 br expression} ]$$

or 3 br alphabets

let

$$t_1 = c * d.$$

$$t_2 = a * b$$

$$t_3 = t_2 + t_1$$

$$* = t_3.$$

$t_1 = b + d = b, t_2 = d, t_3 = d$

3 br with alphabets

with 3 br address



(ii)  $a * -b (b+c)$

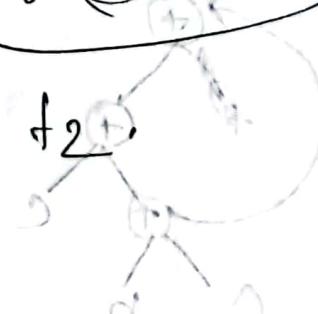


$$t_1 = b + c.$$

$$t_2 = \text{uminus } t_1$$

$$t_3 = a * t_2.$$

② 3 br general



$$(ü) a = b + c * d.$$

$d_1 = c * d$

$d_2 = b + d_1$

$a = d_2$

③ address statement

① assignment

$n = y$ , op code 2 →  $n = y + z$   
value  $\times 2^{n_0}$

② jump conditional if  $n > y$  go to L.  
un n ~~go to l.~~

③ array:  $n = y[i]$

④ pointer address assign

translate expression to quadruple, triple, or indirect.

$$\text{triple} \quad a = b * -c + b * -c$$

$\Rightarrow$

$$t_1 = \text{uminus } c$$

$$t_2 = b * t_1$$

$$t_3 = b * \text{uminus } b * c$$

$$t_4 = b * t_3$$

$$t_5 = t_2 + t_4$$

$$a = t_5$$

OP code

+,-,\*,uminus.

## ① Quadruple

#	OP	arg1	arg2	result
①	uminus	c	-	t <sub>1</sub>
②	*	b	t <sub>1</sub>	t <sub>2</sub>
③	uminus	c	-	t <sub>3</sub>
④	*	b	t <sub>3</sub>	t <sub>4</sub>
⑤	+	t <sub>2</sub>	t <sub>4</sub>	t <sub>5</sub>
	=	t <sub>5</sub>	-	a

#	opcode	Arg1	Arg2
(0)	uminus	c	-
(1)	*	-b	(0) — (0) location 25 26kk
(2)	uminus	c	-
(3)	*	b	(2)
(4)	+	(3)	(1)
(5)	=	a	(4).

Indirect triple:

#	Segment	triple	Arg1	Arg2
(0)	n	(0)		
(1)	n	" (1)		
(2)	n	" (2)		
(3)	"	" (3)		
(4)	"	" (4)		
(5)	"	" 5		

④ Describe 3-address code, Define Quadruples

Triples, Indirect triples.

• 3 address code is a type of Intermediate code which is easy to generate and can be easily converted to machine code. Ex:  $X := Y OP Z$

(i) Quadruples: It is a record structure with 4 fields, which we call operation, arg1, arg2 & result. op code is internal code. Thus, stated statement  $X = Y OP Z$  where  $Y$  is ~~temp~~ Arg1,  $Z$  Arg2  $OP$  is operation ( $*, +, -$ ) and  $X$  is result.

(ii) Triples: It doesn't use of extra temporary variable to represent a operation. It can represent by 3 fields, OP, Arg1 and Arg2.

(iii) Indirect triple: It has been considered in that of listing pointer to triples. The purpose less, complex. It has only operation & segment

④ write 8 address code for symbol tree & DAG

$$a = b^* - c + b^* - c.$$

of program have combine  $a = (b^*(-c)) + (b^*(-c))$   
Code for expression will be added address

$$t_1 = -c$$

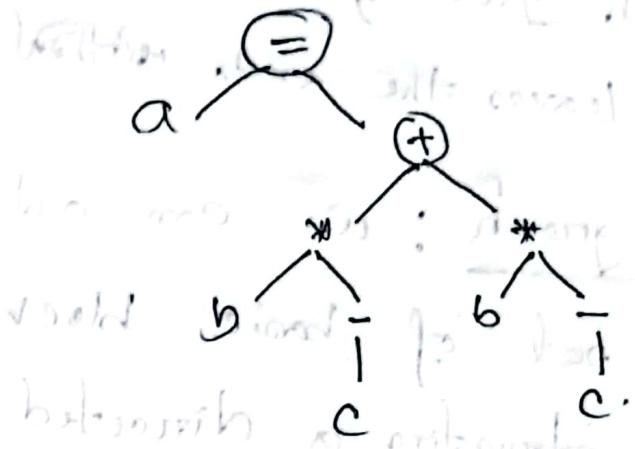
$$t_2 = b^* t_1$$

$$t_3 = -c$$

$$t_4 = b^* t_3$$

$$t_5 = t_2 + t_4$$

$$a = t_5.$$



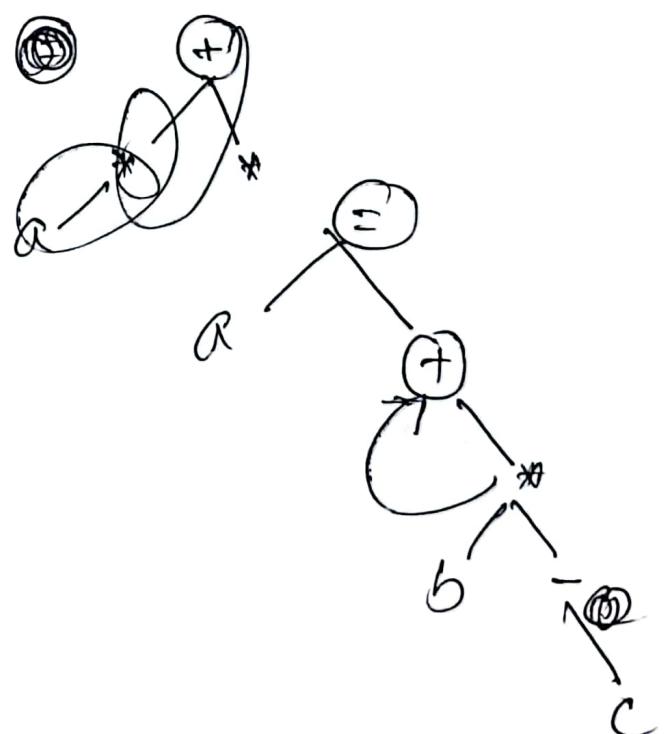
code for DAG

$$t_1 = -c$$

$$t_2 = b^* t_1$$

$$t_3 = t_2 + t_2.$$

$$t_4 = t_5$$



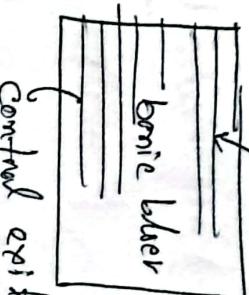
## #Basic Block & flow graph.

Basic block: It is a sequence of statements in which flow of controls enter at the beginning and leaves the end. ~~and~~

Flow graph: We can add flow of control info to the set of basic blocks making up a program by constructing a directed graph or flow graph

## Example of Bonic Block & flow graph

ent-on leader



\* Find starting leader.

(\*) Add address of conditional, unconditional go to one leaders. If  $i \leq s$  go to ③

go to 2<sup>nd</sup> unconditional leader

\* Next line of conditional and unconditional  
go to one leaders (go to 2<sup>nd</sup> unconditional leader)

\* Example.

1.  $i = 1$  Leader B<sub>1</sub>

2.  $i = 1$  Leader B<sub>2</sub>

3.  $t_1 = 5$  \* i Leader B<sub>3</sub>

$$7. a [t_1] = -1$$

$$8. j = j + 1$$

$$9. j < 5 \text{ go to } 3$$

$$10. i = t_1 + 1 \text{ Leader B}_4$$

$$11. \{ i \leq s \text{ go to } 2 \}$$

$$6. t_1 = 12$$

Find the nodes & edge of flow graph



\* 3 address code representation for C code

# int a[10], b[10], i, dp = 0;

```

for (i=0; i<10; i++) {
    dp += a[i] * b[i];
}

```

### 3 address code

1.  $dp = 0$
  2.  $i = 0$
  3. If  $i >= 10$  goto (16)
  4.  $t_1 = \text{addr}(a)$
  5.  $t_2 = i * 4$
  6.  $t_3 = t_1[t_2]$
  7.  $t_4 = \text{addr}(b)$
  8.  $t_5 = i * 4$
  9.  $t_6 = t_4[t_5]$
  10.  $t_7 = t_3 + t_6$
  11.  $t_8 = dp + t_7$
  12.  $dp = t_8$
- Annotations:
- Line 3: A bracket groups lines 4-12, with an arrow pointing to (16) labeled "end loop". To the right, "if false" is written above "2nd".
  - Line 4: An arrow points from  $t_1$  to  $a[i]$ , with "become" written above and "i" below.
  - Line 5: An arrow points from  $t_2$  to  $i * 4$ , with "4b" written below.
  - Line 6: An arrow points from  $t_3$  to  $t_1[t_2]$ .
  - Line 7: An arrow points from  $t_4$  to  $b[i]$ , with "become" written above and "i" below.
  - Line 8: An arrow points from  $t_5$  to  $i * 4$ .
  - Line 9: An arrow points from  $t_6$  to  $t_4[t_5]$ .
  - Line 10: An arrow points from  $t_7$  to  $t_3 + t_6$ .
  - Line 11: An arrow points from  $t_8$  to  $dp + t_7$ .
  - Line 12: An arrow points from  $dp$  to  $t_8$ .
- Final notes:
- $dp = t_8$  is enclosed in a box with arrows pointing to "Goto Goto" and "first mean val".
  - A note at the bottom right says "mean val".

13.  $i \leftarrow i + 1$

14.  $i = 10$

15.  $\text{goto } (d)$

16. end.

(1) C code  $\Rightarrow$  Address code.

$C = 0$

do

{ if ( $a < b$ ) then

$x++;$

else

$x--;$

$C++;$

while ( $C < 5$ )

8 address code

1.  $C = 0$

2.  $i, j (a < b)$  go to (a) if  $a < b$  then

3. go to

4.  $i_1 = x + 1$   $\leftarrow (x + 1)$  so of (a)

5.  $x_1 = +1$

6. go to (g)

7.  $i_2 = x - 1$

8.  $x = i_2$

9.  $C_2 = C + 1$

10.  $C = +3$

11. if ( $C < 5$ ) go to (e)

12. end

4. C code to 3 address

while ( $A \leq c$  and  $B > D$ ) do

    if  $A = 1$  then  $C = C + 1$

    else while  $D \leq D$

        do  $A = A + B$

Ans. 3 address code

- ① if  $A \leq c$  then go to (3) — 2<sup>21</sup> 3 (02101  
2110 next compare)
- ② go to (15) — at 2<sup>21</sup> end 0
- ③ if  $B > D$  goto (5) — condition ok.
- ④ goto (15) — at 2<sup>21</sup> end 0
- ⑤ if  $A = 1$  goto (7)
- ⑥ goto (10)
- ⑦  $t_1 = C + 1$
- ⑧  $C = t_1$
- ⑨ goto (1)
- ⑩ if ( $A \leq D$ ) goto (11)
11.  $t_2 = A + B$
12.  $A = t_2$
14. goto (10)

### Example 5

$\text{prod} = 0$

$i = 1$

do {  
     $\text{prod} = \text{prod} + a[i] \times b[i]$

$i = i + 1$

while ( $i \leq 40$ )

### 8 address code

1.  $\text{prod} = 0$

2.  $i = 1$

3.  $T_1 = 4 \times i$

4.  $T_2 = a[T_1]$

5.  $T_3 = 4 \times i$

6.  $T_4 = b[T_3]$

7.  $T_5 = T_2 \times T_4$

8.  $T_6 = \text{prod} + T_5$

9.  $\text{prod} = T_6$

10.  $T_7 = i + 1$

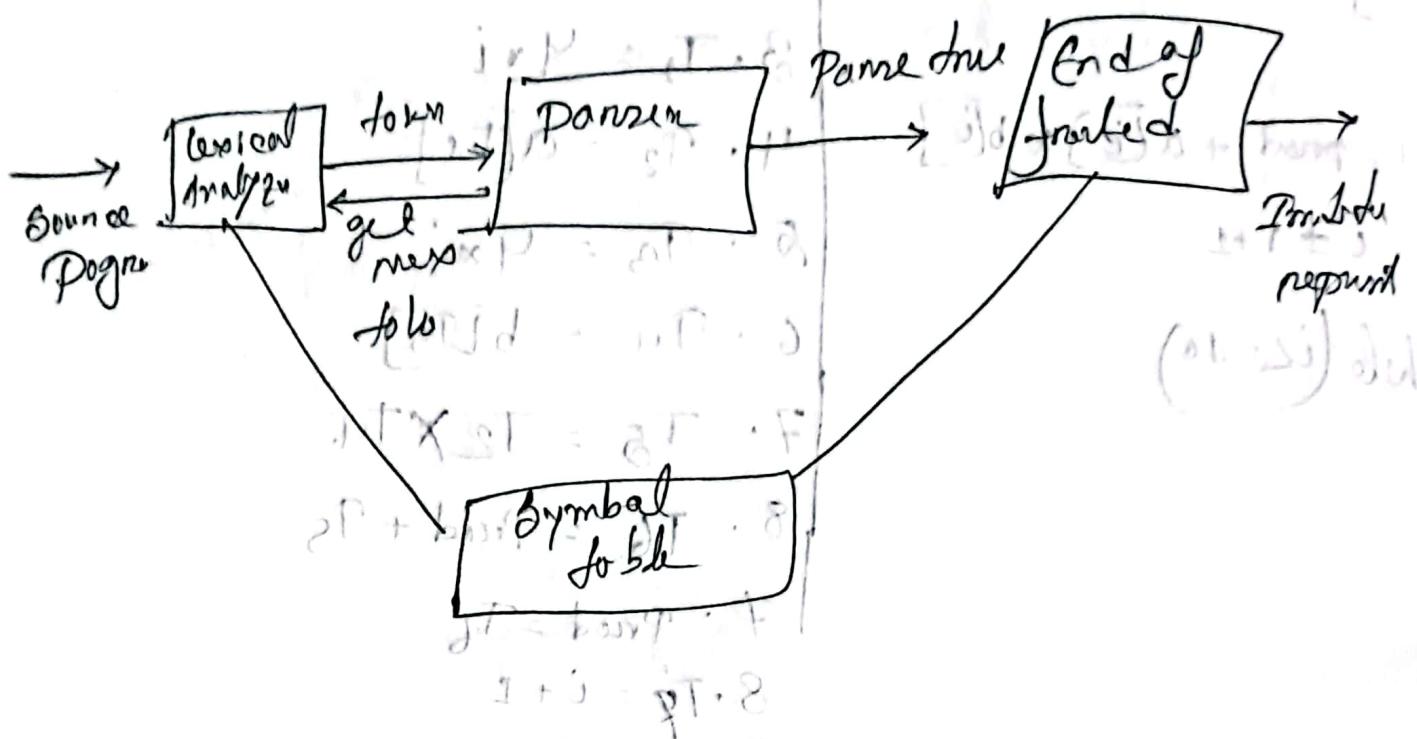
11.  $i = +7$

12. if ( $i \leq 10$ ) goto 3.

### Ex 6

newest move {  
 1.  $\text{prod} = 0$ ,  $i = 1$ ,  $a[1] = 10$ ,  $b[1] = 20$   
 2.  $\text{prod} = \text{prod} + a[1] \times b[1]$   
 3.  $i = i + 1$   
 4.  $a[2] = 15$ ,  $b[2] = 30$   
 5.  $\text{prod} = \text{prod} + a[2] \times b[2]$   
 6.  $i = i + 1$   
 7.  $a[3] = 20$ ,  $b[3] = 40$   
 8.  $\text{prod} = \text{prod} + a[3] \times b[3]$   
 9.  $i = i + 1$   
 10.  $a[4] = 25$ ,  $b[4] = 50$   
 11.  $\text{prod} = \text{prod} + a[4] \times b[4]$   
 12.  $i = i + 1$   
 13.  $a[5] = 30$ ,  $b[5] = 60$   
 14.  $\text{prod} = \text{prod} + a[5] \times b[5]$   
 15.  $i = i + 1$   
 16.  $a[6] = 35$ ,  $b[6] = 70$   
 17.  $\text{prod} = \text{prod} + a[6] \times b[6]$   
 18.  $i = i + 1$   
 19.  $a[7] = 40$ ,  $b[7] = 80$   
 20.  $\text{prod} = \text{prod} + a[7] \times b[7]$   
 21.  $i = i + 1$   
 22.  $a[8] = 45$ ,  $b[8] = 90$   
 23.  $\text{prod} = \text{prod} + a[8] \times b[8]$   
 24.  $i = i + 1$   
 25.  $a[9] = 50$ ,  $b[9] = 100$   
 26.  $\text{prod} = \text{prod} + a[9] \times b[9]$   
 27.  $i = i + 1$   
 28.  $a[10] = 55$ ,  $b[10] = 110$   
 29.  $\text{prod} = \text{prod} + a[10] \times b[10]$   
 30.  $i = i + 1$   
 31.  $a[11] = 60$ ,  $b[11] = 120$   
 32.  $\text{prod} = \text{prod} + a[11] \times b[11]$   
 33.  $i = i + 1$   
 34.  $a[12] = 65$ ,  $b[12] = 130$   
 35.  $\text{prod} = \text{prod} + a[12] \times b[12]$   
 36.  $i = i + 1$   
 37.  $a[13] = 70$ ,  $b[13] = 140$   
 38.  $\text{prod} = \text{prod} + a[13] \times b[13]$   
 39.  $i = i + 1$   
 40.  $a[14] = 75$ ,  $b[14] = 150$   
 41.  $\text{prod} = \text{prod} + a[14] \times b[14]$   
 42.  $i = i + 1$   
 43.  $a[15] = 80$ ,  $b[15] = 160$   
 44.  $\text{prod} = \text{prod} + a[15] \times b[15]$   
 45.  $i = i + 1$   
 46.  $a[16] = 85$ ,  $b[16] = 170$   
 47.  $\text{prod} = \text{prod} + a[16] \times b[16]$   
 48.  $i = i + 1$   
 49.  $a[17] = 90$ ,  $b[17] = 180$   
 50.  $\text{prod} = \text{prod} + a[17] \times b[17]$   
 51.  $i = i + 1$   
 52.  $a[18] = 95$ ,  $b[18] = 190$   
 53.  $\text{prod} = \text{prod} + a[18] \times b[18]$   
 54.  $i = i + 1$   
 55.  $a[19] = 100$ ,  $b[19] = 200$   
 56.  $\text{prod} = \text{prod} + a[19] \times b[19]$   
 57.  $i = i + 1$   
 58.  $a[20] = 105$ ,  $b[20] = 210$   
 59.  $\text{prod} = \text{prod} + a[20] \times b[20]$   
 60.  $i = i + 1$   
 61.  $a[21] = 110$ ,  $b[21] = 220$   
 62.  $\text{prod} = \text{prod} + a[21] \times b[21]$   
 63.  $i = i + 1$   
 64.  $a[22] = 115$ ,  $b[22] = 230$   
 65.  $\text{prod} = \text{prod} + a[22] \times b[22]$   
 66.  $i = i + 1$   
 67.  $a[23] = 120$ ,  $b[23] = 240$   
 68.  $\text{prod} = \text{prod} + a[23] \times b[23]$   
 69.  $i = i + 1$   
 70.  $a[24] = 125$ ,  $b[24] = 250$   
 71.  $\text{prod} = \text{prod} + a[24] \times b[24]$   
 72.  $i = i + 1$   
 73.  $a[25] = 130$ ,  $b[25] = 260$   
 74.  $\text{prod} = \text{prod} + a[25] \times b[25]$   
 75.  $i = i + 1$   
 76.  $a[26] = 135$ ,  $b[26] = 270$   
 77.  $\text{prod} = \text{prod} + a[26] \times b[26]$   
 78.  $i = i + 1$   
 79.  $a[27] = 140$ ,  $b[27] = 280$   
 80.  $\text{prod} = \text{prod} + a[27] \times b[27]$   
 81.  $i = i + 1$   
 82.  $a[28] = 145$ ,  $b[28] = 290$   
 83.  $\text{prod} = \text{prod} + a[28] \times b[28]$   
 84.  $i = i + 1$   
 85.  $a[29] = 150$ ,  $b[29] = 300$   
 86.  $\text{prod} = \text{prod} + a[29] \times b[29]$   
 87.  $i = i + 1$   
 88.  $a[30] = 155$ ,  $b[30] = 310$   
 89.  $\text{prod} = \text{prod} + a[30] \times b[30]$   
 90.  $i = i + 1$   
 91.  $a[31] = 160$ ,  $b[31] = 320$   
 92.  $\text{prod} = \text{prod} + a[31] \times b[31]$   
 93.  $i = i + 1$   
 94.  $a[32] = 165$ ,  $b[32] = 330$   
 95.  $\text{prod} = \text{prod} + a[32] \times b[32]$   
 96.  $i = i + 1$   
 97.  $a[33] = 170$ ,  $b[33] = 340$   
 98.  $\text{prod} = \text{prod} + a[33] \times b[33]$   
 99.  $i = i + 1$   
 100.  $a[34] = 175$ ,  $b[34] = 350$   
 101.  $\text{prod} = \text{prod} + a[34] \times b[34]$   
 102.  $i = i + 1$   
 103.  $a[35] = 180$ ,  $b[35] = 360$   
 104.  $\text{prod} = \text{prod} + a[35] \times b[35]$   
 105.  $i = i + 1$   
 106.  $a[36] = 185$ ,  $b[36] = 370$   
 107.  $\text{prod} = \text{prod} + a[36] \times b[36]$   
 108.  $i = i + 1$   
 109.  $a[37] = 190$ ,  $b[37] = 380$   
 110.  $\text{prod} = \text{prod} + a[37] \times b[37]$   
 111.  $i = i + 1$   
 112.  $a[38] = 195$ ,  $b[38] = 390$   
 113.  $\text{prod} = \text{prod} + a[38] \times b[38]$   
 114.  $i = i + 1$   
 115.  $a[39] = 200$ ,  $b[39] = 400$   
 116.  $\text{prod} = \text{prod} + a[39] \times b[39]$   
 117.  $i = i + 1$   
 118.  $a[40] = 205$ ,  $b[40] = 410$   
 119.  $\text{prod} = \text{prod} + a[40] \times b[40]$   
 120.  $i = i + 1$   
 121.  $a[41] = 210$ ,  $b[41] = 420$   
 122.  $\text{prod} = \text{prod} + a[41] \times b[41]$   
 123.  $i = i + 1$   
 124.  $a[42] = 215$ ,  $b[42] = 430$   
 125.  $\text{prod} = \text{prod} + a[42] \times b[42]$   
 126.  $i = i + 1$   
 127.  $a[43] = 220$ ,  $b[43] = 440$   
 128.  $\text{prod} = \text{prod} + a[43] \times b[43]$   
 129.  $i = i + 1$   
 130.  $a[44] = 225$ ,  $b[44] = 450$   
 131.  $\text{prod} = \text{prod} + a[44] \times b[44]$   
 132.  $i = i + 1$   
 133.  $a[45] = 230$ ,  $b[45] = 460$   
 134.  $\text{prod} = \text{prod} + a[45] \times b[45]$   
 135.  $i = i + 1$   
 136.  $a[46] = 235$ ,  $b[46] = 470$   
 137.  $\text{prod} = \text{prod} + a[46] \times b[46]$   
 138.  $i = i + 1$   
 139.  $a[47] = 240$ ,  $b[47] = 480$   
 140.  $\text{prod} = \text{prod} + a[47] \times b[47]$   
 141.  $i = i + 1$   
 142.  $a[48] = 245$ ,  $b[48] = 490$   
 143.  $\text{prod} = \text{prod} + a[48] \times b[48]$   
 144.  $i = i + 1$   
 145.  $a[49] = 250$ ,  $b[49] = 500$   
 146.  $\text{prod} = \text{prod} + a[49] \times b[49]$   
 147.  $i = i + 1$   
 148.  $a[50] = 255$ ,  $b[50] = 510$   
 149.  $\text{prod} = \text{prod} + a[50] \times b[50]$   
 150.  $i = i + 1$   
 151.  $a[51] = 260$ ,  $b[51] = 520$   
 152.  $\text{prod} = \text{prod} + a[51] \times b[51]$   
 153.  $i = i + 1$   
 154.  $a[52] = 265$ ,  $b[52] = 530$   
 155.  $\text{prod} = \text{prod} + a[52] \times b[52]$   
 156.  $i = i + 1$   
 157.  $a[53] = 270$ ,  $b[53] = 540$   
 158.  $\text{prod} = \text{prod} + a[53] \times b[53]$   
 159.  $i = i + 1$   
 160.  $a[54] = 275$ ,  $b[54] = 550$   
 161.  $\text{prod} = \text{prod} + a[54] \times b[54]$   
 162.  $i = i + 1$   
 163.  $a[55] = 280$ ,  $b[55] = 560$   
 164.  $\text{prod} = \text{prod} + a[55] \times b[55]$   
 165.  $i = i + 1$   
 166.  $a[56] = 285$ ,  $b[56] = 570$   
 167.  $\text{prod} = \text{prod} + a[56] \times b[56]$   
 168.  $i = i + 1$   
 169.  $a[57] = 290$ ,  $b[57] = 580$   
 170.  $\text{prod} = \text{prod} + a[57] \times b[57]$   
 171.  $i = i + 1$   
 172.  $a[58] = 295$ ,  $b[58] = 590$   
 173.  $\text{prod} = \text{prod} + a[58] \times b[58]$   
 174.  $i = i + 1$   
 175.  $a[59] = 300$ ,  $b[59] = 600$   
 176.  $\text{prod} = \text{prod} + a[59] \times b[59]$   
 177.  $i = i + 1$   
 178.  $a[60] = 305$ ,  $b[60] = 610$   
 179.  $\text{prod} = \text{prod} + a[60] \times b[60]$   
 180.  $i = i + 1$   
 181.  $a[61] = 310$ ,  $b[61] = 620$   
 182.  $\text{prod} = \text{prod} + a[61] \times b[61]$   
 183.  $i = i + 1$   
 184.  $a[62] = 315$ ,  $b[62] = 630$   
 185.  $\text{prod} = \text{prod} + a[62] \times b[62]$   
 186.  $i = i + 1$   
 187.  $a[63] = 320$ ,  $b[63] = 640$   
 188.  $\text{prod} = \text{prod} + a[63] \times b[63]$   
 189.  $i = i + 1$   
 190.  $a[64] = 325$ ,  $b[64] = 650$   
 191.  $\text{prod} = \text{prod} + a[64] \times b[64]$   
 192.  $i = i + 1$   
 193.  $a[65] = 330$ ,  $b[65] = 660$   
 194.  $\text{prod} = \text{prod} + a[65] \times b[65]$   
 195.  $i = i + 1$   
 196.  $a[66] = 335$ ,  $b[66] = 670$   
 197.  $\text{prod} = \text{prod} + a[66] \times b[66]$   
 198.  $i = i + 1$   
 199.  $a[67] = 340$ ,  $b[67] = 680$   
 200.  $\text{prod} = \text{prod} + a[67] \times b[67]$   
 201.  $i = i + 1$   
 202.  $a[68] = 345$ ,  $b[68] = 690$   
 203.  $\text{prod} = \text{prod} + a[68] \times b[68]$   
 204.  $i = i + 1$   
 205.  $a[69] = 350$ ,  $b[69] = 700$   
 206.  $\text{prod} = \text{prod} + a[69] \times b[69]$   
 207.  $i = i + 1$   
 208.  $a[70] = 355$ ,  $b[70] = 710$   
 209.  $\text{prod} = \text{prod} + a[70] \times b[70]$   
 210.  $i = i + 1$   
 211.  $a[71] = 360$ ,  $b[71] = 720$   
 212.  $\text{prod} = \text{prod} + a[71] \times b[71]$   
 213.  $i = i + 1$   
 214.  $a[72] = 365$ ,  $b[72] = 730$   
 215.  $\text{prod} = \text{prod} + a[72] \times b[72]$   
 216.  $i = i + 1$   
 217.  $a[73] = 370$ ,  $b[73] = 740$   
 218.  $\text{prod} = \text{prod} + a[73] \times b[73]$   
 219.  $i = i + 1$   
 220.  $a[74] = 375$ ,  $b[74] = 750$   
 221.  $\text{prod} = \text{prod} + a[74] \times b[74]$   
 222.  $i = i + 1$   
 223.  $a[75] = 380$ ,  $b[75] = 760$   
 224.  $\text{prod} = \text{prod} + a[75] \times b[75]$   
 225.  $i = i + 1$   
 226.  $a[76] = 385$ ,  $b[76] = 770$   
 227.  $\text{prod} = \text{prod} + a[76] \times b[76]$   
 228.  $i = i + 1$   
 229.  $a[77] = 390$ ,  $b[77] = 780$   
 230.  $\text{prod} = \text{prod} + a[77] \times b[77]$   
 231.  $i = i + 1$   
 232.  $a[78] = 395$ ,  $b[78] = 790$   
 233.  $\text{prod} = \text{prod} + a[78] \times b[78]$   
 234.  $i = i + 1$   
 235.  $a[79] = 400$ ,  $b[79] = 800$   
 236.  $\text{prod} = \text{prod} + a[79] \times b[79]$   
 237.  $i = i + 1$   
 238.  $a[80] = 405$ ,  $b[80] = 810$   
 239.  $\text{prod} = \text{prod} + a[80] \times b[80]$   
 240.  $i = i + 1$   
 241.  $a[81] = 410$ ,  $b[81] = 820$   
 242.  $\text{prod} = \text{prod} + a[81] \times b[81]$   
 243.  $i = i + 1$   
 244.  $a[82] = 415$ ,  $b[82] = 830$   
 245.  $\text{prod} = \text{prod} + a[82] \times b[82]$   
 246.  $i = i + 1$   
 247.  $a[83] = 420$ ,  $b[83] = 840$   
 248.  $\text{prod} = \text{prod} + a[83] \times b[83]$   
 249.  $i = i + 1$   
 250.  $a[84] = 425$ ,  $b[84] = 850$   
 251.  $\text{prod} = \text{prod} + a[84] \times b[84]$   
 252.  $i = i + 1$   
 253.  $a[85] = 430$ ,  $b[85] = 860$   
 254.  $\text{prod} = \text{prod} + a[85] \times b[85]$   
 255.  $i = i + 1$   
 256.  $a[86] = 435$ ,  $b[86] = 870$   
 257.  $\text{prod} = \text{prod} + a[86] \times b[86]$   
 258.  $i = i + 1$   
 259.  $a[87] = 440$ ,  $b[87] = 880$   
 260.  $\text{prod} = \text{prod} + a[87] \times b[87]$   
 261.  $i = i + 1$   
 262.  $a[88] = 445$ ,  $b[88] = 890$   
 263.  $\text{prod} = \text{prod} + a[88] \times b[88]$   
 264.  $i = i + 1$   
 265.  $a[89] = 450$ ,  $b[89] = 900$   
 266.  $\text{prod} = \text{prod} + a[89] \times b[89]$   
 267.  $i = i + 1$   
 268.  $a[90] = 455$ ,  $b[90] = 910$   
 269.  $\text{prod} = \text{prod} + a[90] \times b[90]$   
 270.  $i = i + 1$   
 271.  $a[91] = 460$ ,  $b[91] = 920$   
 272.  $\text{prod} = \text{prod} + a[91] \times b[91]$   
 273.  $i = i + 1$   
 274.  $a[92] = 465$ ,  $b[92] = 930$   
 275.  $\text{prod} = \text{prod} + a[92] \times b[92]$   
 276.  $i = i + 1$   
 277.  $a[93] = 470$ ,  $b[93] = 940$   
 278.  $\text{prod} = \text{prod} + a[93] \times b[93]$   
 279.  $i = i + 1$   
 280.  $a[94] = 475$ ,  $b[94] = 950$   
 281.  $\text{prod} = \text{prod} + a[94] \times b[94]$   
 282.  $i = i + 1$   
 283.  $a[95] = 480$ ,  $b[95] = 960$   
 284.  $\text{prod} = \text{prod} + a[95] \times b[95]$   
 285.  $i = i + 1$   
 286.  $a[96] = 485$ ,  $b[96] = 970$   
 287.  $\text{prod} = \text{prod} + a[96] \times b[96]$   
 288.  $i = i + 1$   
 289.  $a[97] = 490$ ,  $b[97] = 980$   
 290.  $\text{prod} = \text{prod} + a[97] \times b[97]$   
 291.  $i = i + 1$   
 292.  $a[98] = 495$ ,  $b[98] = 990$   
 293.  $\text{prod} = \text{prod} + a[98] \times b[98]$   
 294.  $i = i + 1$   
 295.  $a[99] = 500$ ,  $b[99] = 1000$   
 296.  $\text{prod} = \text{prod} + a[99] \times b[99]$   
 297.  $i = i + 1$   
 298.  $a[100] = 505$ ,  $b[100] = 1010$   
 299.  $\text{prod} = \text{prod} + a[100] \times b[100]$   
 300.  $i = i + 1$   
 301.  $a[101] = 510$ ,  $b[101] = 1020$   
 302.  $\text{prod} = \text{prod} + a[101] \times b[101]$   
 303.  $i = i + 1$   
 304.  $a[102] = 515$ ,  $b[102] = 1030$   
 305.  $\text{prod} = \text{prod} + a[102] \times b[102]$   
 306.  $i = i + 1$   
 307.  $a[103] = 520$ ,  $b[103] = 1040$   
 308.  $\text{prod} = \text{prod} + a[103] \times b[103]$   
 309.  $i = i + 1$   
 310.  $a[104] = 525$ ,  $b[104] = 1050$   
 311.  $\text{prod} = \text{prod} + a[104] \times b[104]$   
 312.  $i = i + 1$   
 313.  $a[105] = 530$ ,  $b[105] = 1060$   
 314.  $\text{prod} = \text{prod} + a[105] \times b[105]$   
 315.  $i = i + 1$   
 316.  $a[106] = 535$ ,  $b[106] = 1070$   
 317.  $\text{prod} = \text{prod} + a[106] \times b[106]$   
 318.  $i = i + 1$   
 319.  $a[107] = 540$ ,  $b[107] = 1080$   
 320.  $\text{prod} = \text{prod} + a[107] \times b[107]$   
 321.  $i = i + 1$   
 322.  $a[108] = 545$ ,  $b[108] = 1090$   
 323.  $\text{prod} = \text{prod} + a[108] \times b[108]$   
 324.  $i = i + 1$   
 325.  $a[109] = 550$ ,  $b[109] = 1100$   
 326.  $\text{prod} = \text{prod} + a[109] \times b[109]$   
 327.  $i = i + 1$   
 328.  $a[110] = 555$ ,  $b[110] = 1110$   
 329.  $\text{prod} = \text{prod} + a[110] \times b[110]$   
 330.  $i = i + 1$   
 331.  $a[111] = 560$ ,  $b[111] = 1120$   
 332.  $\text{prod} = \text{prod} + a[111] \times b[111]$   
 333

## scale of parser



## ⑥ Error Recovery Strategies.

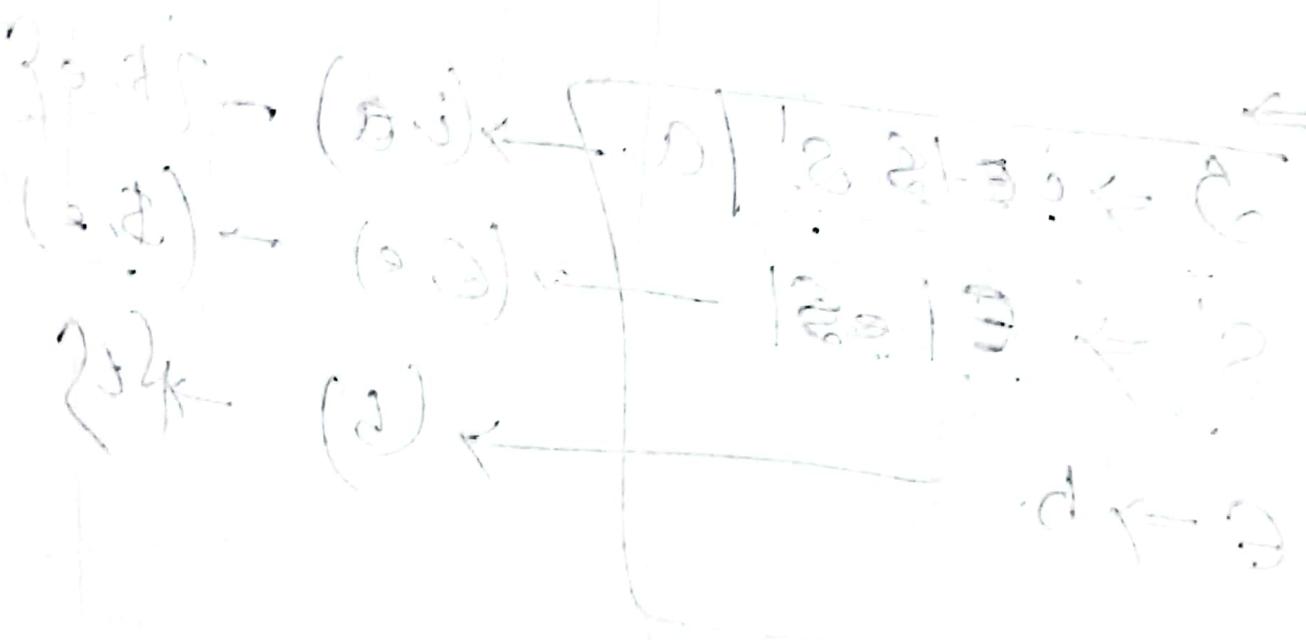
Prime mode: It is the easiest way of error recovery.  
It stops the parser from developing (for parser to Del. the error (in error set))  
If there are less error, then its best

Phrase Level: In this stage If error find then parser perform local correction on this input. It can replace a prefix of the input.

Local local connection will be  
done on input (to prefix will replace to)

Error Production: It occurs in the code. So that the designer can create enhanced grammar for me. So that production create incorrect structure.

Global correction: We want a compiler that makes few changes in producing to give correct output string. If x.input string and grammar or the algorithm itself can find parse tree in one easily and time taken for one step is less.



Find & follow

$$S \rightarrow Bd | c b \rightarrow (a, b, c, d) \rightarrow (\$,)$$

$$B \rightarrow aB | \epsilon \rightarrow (a, \epsilon) \rightarrow (\$)$$

$$C \rightarrow Cc | \epsilon \rightarrow (c, \epsilon) \rightarrow (\$)$$

$$S \rightarrow i \in S | i \in t \cup S | a.$$

$$\epsilon \rightarrow b$$

$$\Rightarrow S \rightarrow i \in t \cup S' | a \rightarrow (i, a) \rightarrow \{\$, e\}$$
$$S' \Rightarrow \epsilon | \in t \rightarrow (\epsilon, \epsilon) \rightarrow (\$, \epsilon)$$
$$\epsilon \rightarrow b \rightarrow (\$) \rightarrow \{t\}$$

## LL(1) parsing table

① and find and follow the LL-table

①	$E \rightarrow TE'$	$\rightarrow (id, C) \rightarrow$	( $id, \$$ )
②	$E' \rightarrow +TE'$	$  C \rightarrow (+, C) \rightarrow$	( $+, \$$ )
③	$T \rightarrow ET'$	$  id, C \rightarrow (+, id, C) \rightarrow$	( $+, id, \$$ )
④	$T' \rightarrow +TT'$	$  \epsilon \rightarrow (+, \epsilon) \rightarrow$	( $+, \$$ )
⑤	$F \rightarrow id   (\epsilon)$	$  id   (\epsilon) \rightarrow$	( $id, +, \$$ )

	$id$	$C$	$*$	$+$	$\$$	)
$E$	①	②				
$E'$			②	②	②	
$T$	④	③				
$T'$			①	④	④	
$F$	⑤	⑥				

2nd  $E' \Rightarrow +TT' | \epsilon$   
 so  
 no  
 $E' \Rightarrow E - 21N$   
 given follow  
 set

⑩  $a(a,a)$ . To re-arrange derivation in short audience.

parson

$S \rightarrow (L) / a$

$L \rightarrow L, S / S$ .

\*  $(a(a,a))$

$\Rightarrow (\delta(a,a))$

$\Rightarrow S(S,a)$

$\Rightarrow S(L,a)$

$\Rightarrow S(L,S)$

$\Rightarrow S(L)$

$\Rightarrow S(L)$

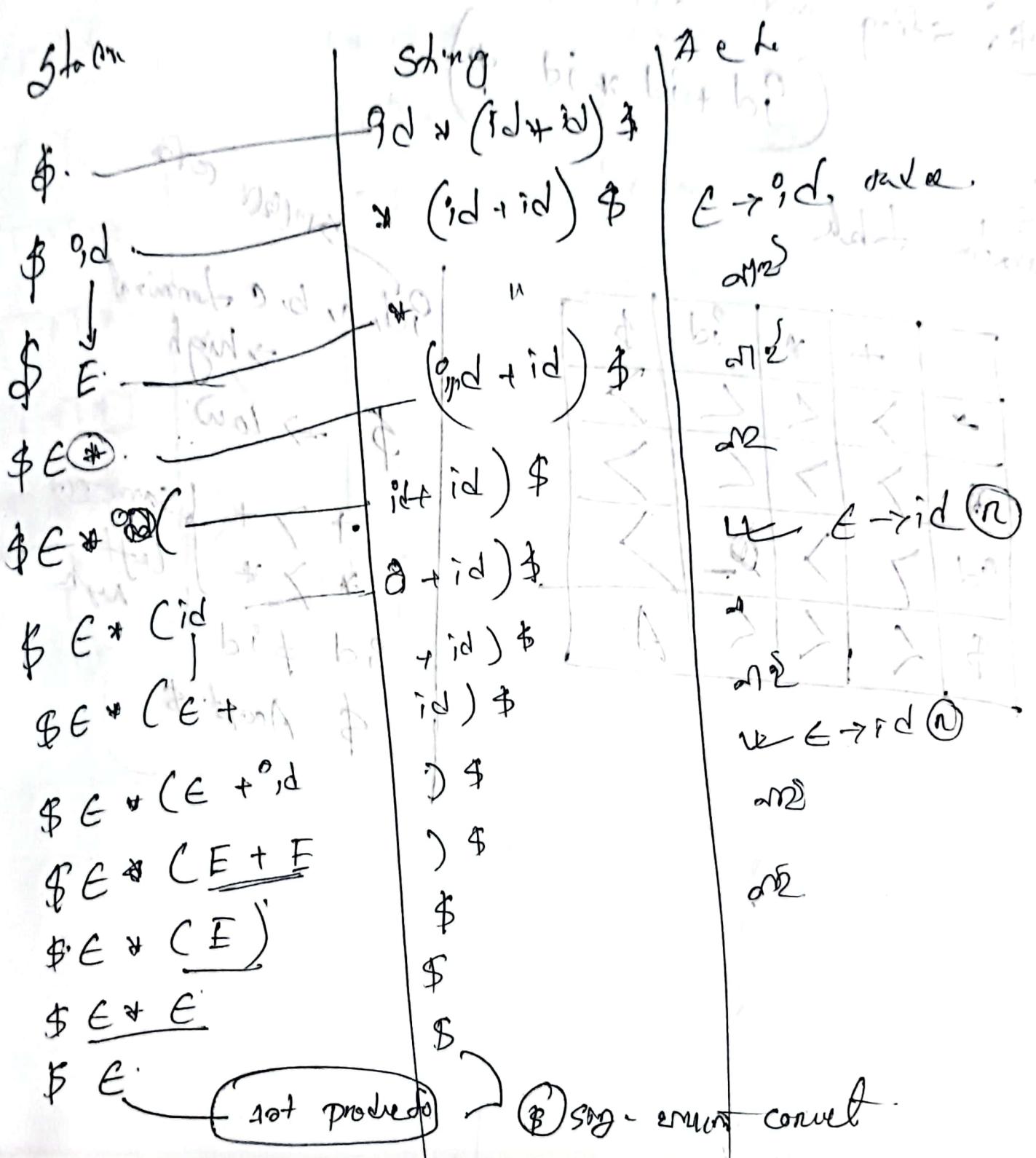
$\Rightarrow (L,S)$

$\Rightarrow (L)$

$\Rightarrow S$

for Q

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E \times E \\ E &\rightarrow (E) \\ E &\rightarrow \underline{id}. \end{aligned}$$



math problem

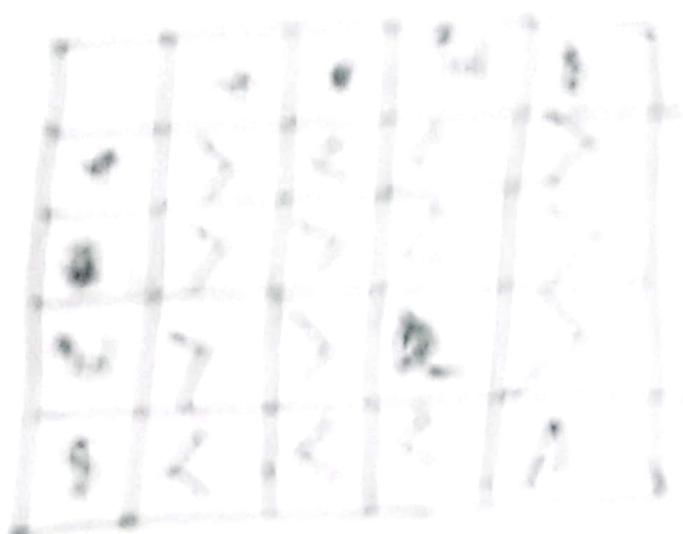
Aug 15, 2014

→ 70 + 20 / 3 \* 7 / 4

⑥ Aug 15, 2014

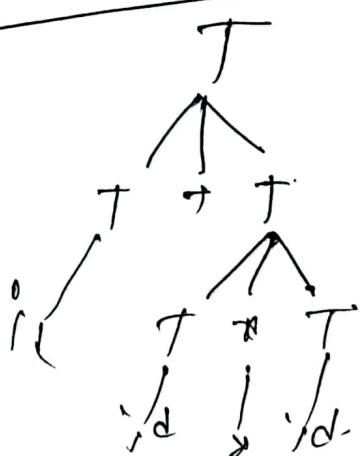
(30 + 20 \* 10 / 8)

Aug 15, 2014



Ans  
→ 70 + 20 / 3 \* 7 / 4  
→ 70 + 20 / 21 / 4  
→ 70 + 20 / 5.25  
→ 70 + 3.9  
→ 73.9

Stack	relation	Input	Action
\$	<	q <sub>d</sub>	shift q <sub>d</sub>
q <sub>d</sub>	>	q <sub>d</sub> *	reduce
q <sub>d</sub> T	<	+ id	shift down level
q <sub>d</sub> T*	>	id *	owl
q <sub>d</sub> T* id	<	*	reduce
q <sub>d</sub> T+T	>	*	shift
q <sub>d</sub> T+T*	<	id	push
q <sub>d</sub> T+T* id	>	\$	reduce
q <sub>d</sub> T+T* T	<	\$	reduce
q <sub>d</sub> T+T	>	\$	reduce
q <sub>d</sub> T	.	\$	reduce
q <sub>d</sub>	.	\$	reduce
	Accept	\$	



## ④ operation precedence

$$A \rightarrow A + A / A * A \quad | \underline{a}, \underline{b}f + \underline{b}f$$

Solving  $a + a * a / a + a * a$

table

	+	*	a	\$	
+	>	x	<	>	$a + b f *$
*	>	>	<	>	$* b f$
a	>	>	x	>	$a * b f$
\$	<	<	<	A	$a + b f$

