Final

Bottom up parsing

- shift reduce parsing
- SLR / LR (0)
- CLR / LR(1)
- LALR

[1] shift Reduce Parsing:

Grammar:

$S \rightarrow aTRe$

$T \rightarrow Tbc \mid b$

$R \rightarrow d$

Input : abbcde

Step1

$\begin{array}{|c|} a \\ \hline \end{array}$ shift → a

Remaining : bbcde

step2

$\begin{array}{|c|} b \\ a \\ \hline \end{array}$ shift b

Remaining Input : bcde

Step3

$\begin{array}{|c|} T \\ a \\ \hline \end{array}$ Reduced by T → b

R.I : bcde

Step4

$\begin{array}{|c|} b \\ a \\ \hline \end{array}$ shift → b

R.I : cde

Step 5

$\begin{array}{|c|} c \\ b \\ T \\ a \\ \hline \end{array}$ shift → c

R.I : de

Step 6

$\begin{array}{|c|} T \\ a \\ \hline \end{array}$ ~~shift~~ reduce by T → Tbc

R.I : de

step7

$\begin{array}{|c|} d \\ T \\ a \\ \hline \end{array}$ shift → d

R.I : e

Step8

$\begin{array}{|c|} R \\ T \\ a \\ \hline \end{array}$ Reduce by R → d

Step9

$\begin{array}{|c|} e \\ R \\ T \\ a \\ \hline \end{array}$ shift → e

R.I :

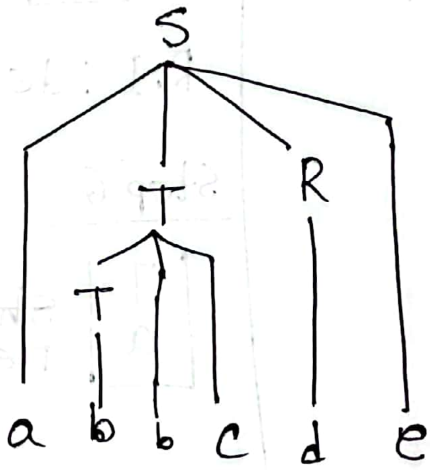
Step 10

$\boxed{S}$ Reduced by $S \rightarrow aTRe$

This parsing is successful

* Shift Reduce করে starting symbol না পেলে সেটা ambiguous grammar.

Parsing Tree:



* Grammar

$E \longrightarrow E+T$

$E \longrightarrow T$

$T \longrightarrow T*F$

$T \longrightarrow F$

$F \longrightarrow id$

$F \longrightarrow (E)$

Input : id+id

stack	input	action
$	id + id $	shift id
$id	+ id $	Reduce by F → id
$F	+ id $	" " T → F
$T	+ id $	" " E → T
$E	+ id $	shift +
$E+	id $	" id
$E+id	$	Reduced by F → id
$E+F	$	" " T → F
$E+T	$	" " E → E+T
$E	$	Accept

* Grammar

S → (L) / a

L → L,s / s

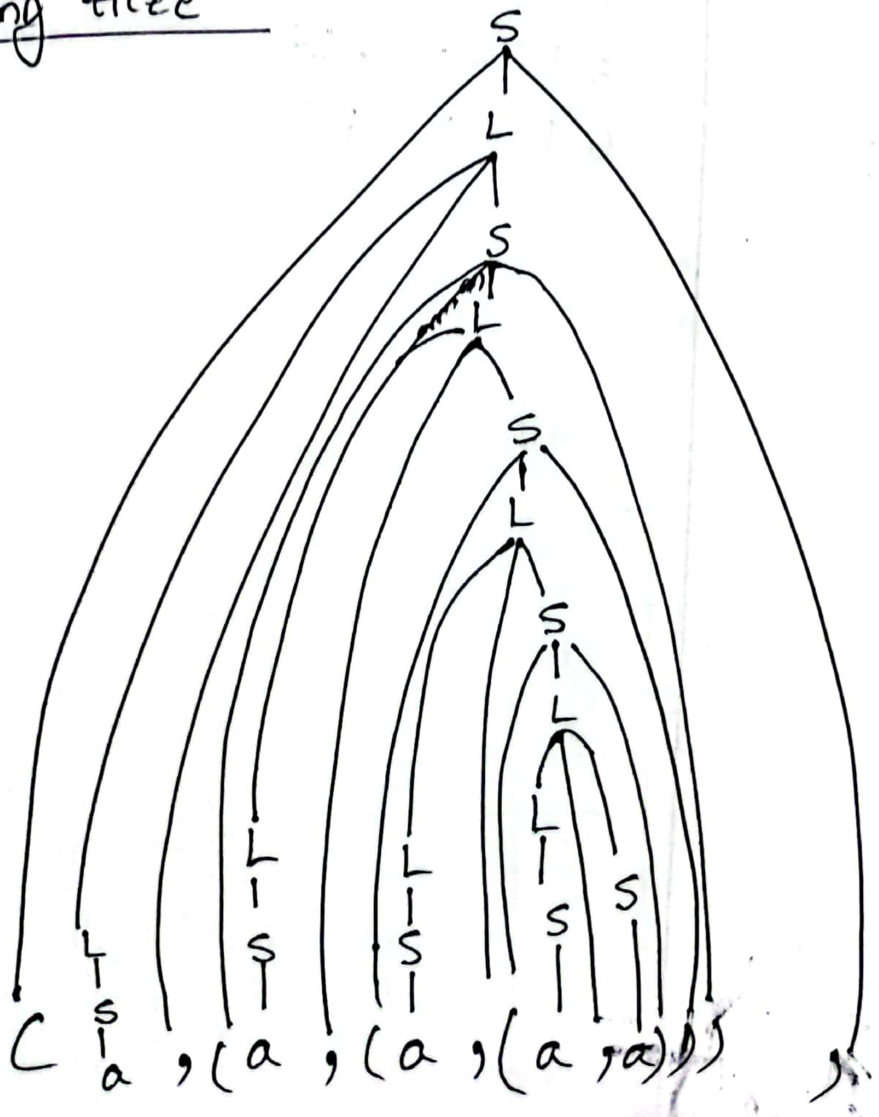
Input : (a, (a, (a, (a,a))))

Stack	Input	Action
$	(a,(a,(a,(a,a))))$	shift (
$(	a,(a,(a,(a,a)))$	" a
$(a	,(a,(a,(a,a))))$	reduced by S → a
$(S	,(a,(a,(a,a))))$	" " L → s
$(L,	(a,(a,(a,a))))$	shift (

Stack	Input	action
$ (L, (	a, (a, (a, a)))) $	shift a ~~shift a~~ ~~reduced by L→S~~
$ (L, (a	,(a, (a, a)))) $	reduced by ~~shift a~~ S→a
$ (L, (S	, (a, (a, a)))) $	~~Reduced by L→S~~ L→S ~~shift (~~
$ (L, (L,	(a, (a, a)))) $	shift ,
$ (L, (L,	(a, (a, a)))) $	shift (
$ (L, (L, (	a, (a, a)))) $	" a
$ (L, (L, (a	,(a, a))) $	reduced by S→a
$ (L, (L, (S	,(a, a)))) $	~~Shift~~ reduced by L→S
$ (L, (L, (L	,(a, a)))) $	" ,
$ (L, (L, (L,	(a, a)))) $	" (
$ (L, (L, (L, (	a, a)))) $	~~Reduced by S→a~~ Shift a
$ (L, (L, (L, (a	, a)))) $	reducedby S→a
$ (L, (L, (L, (S	, a)))) $	~~Shift~~ " " L→S
$ (L, (L, (L, (L	, a)))) $	shift ,
$ (L, (L, (L, (L,	a)))) $	" a ~~Reduced by S→a~~
$ (L, (L, (L, (L, a	)))) $	reducedby " S→a
$ (L, (L, (L, (L, S	)))) $	" " ~~Shift~~ L→ L, S
$(L, (L, (L, (L	)))) $	~~Reduced by~~ shift)
$ (L, (L, (L, (L)	))) $	reduced bx ~~L→L~~ S→(L)
$. (L, (L, (L, S	))) $	" " L→ L, S
$ (L, (L, (L,	))) $	shift)
$ (L, (L, (L)	)) $	reduced by S→(L)

$(L,(L,S	))$	reduced by L → L,S
$(L,(L	))$	shift)
$(L,(L)	)$	reduced by S → (L)
$ (L,S	)$	reduced by L → L,S
$ (L	)$	shift)
$ (L)	$	reduced by S → (L)
$ S	$	Accept

Parsing tree



SLR Parsing →

S → simple

L → left to Right

R → Right most derivation

Grammar

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → id
6. F → (E)

Step1

Augmentation :

0. E' → E
1. E → E+T
2. E → T
3. T → T*F
4. T → F .
5. F → id
6. F → (E)

Step 2

{ 0. E' → .E

1 E → .E+T

2 E → .T

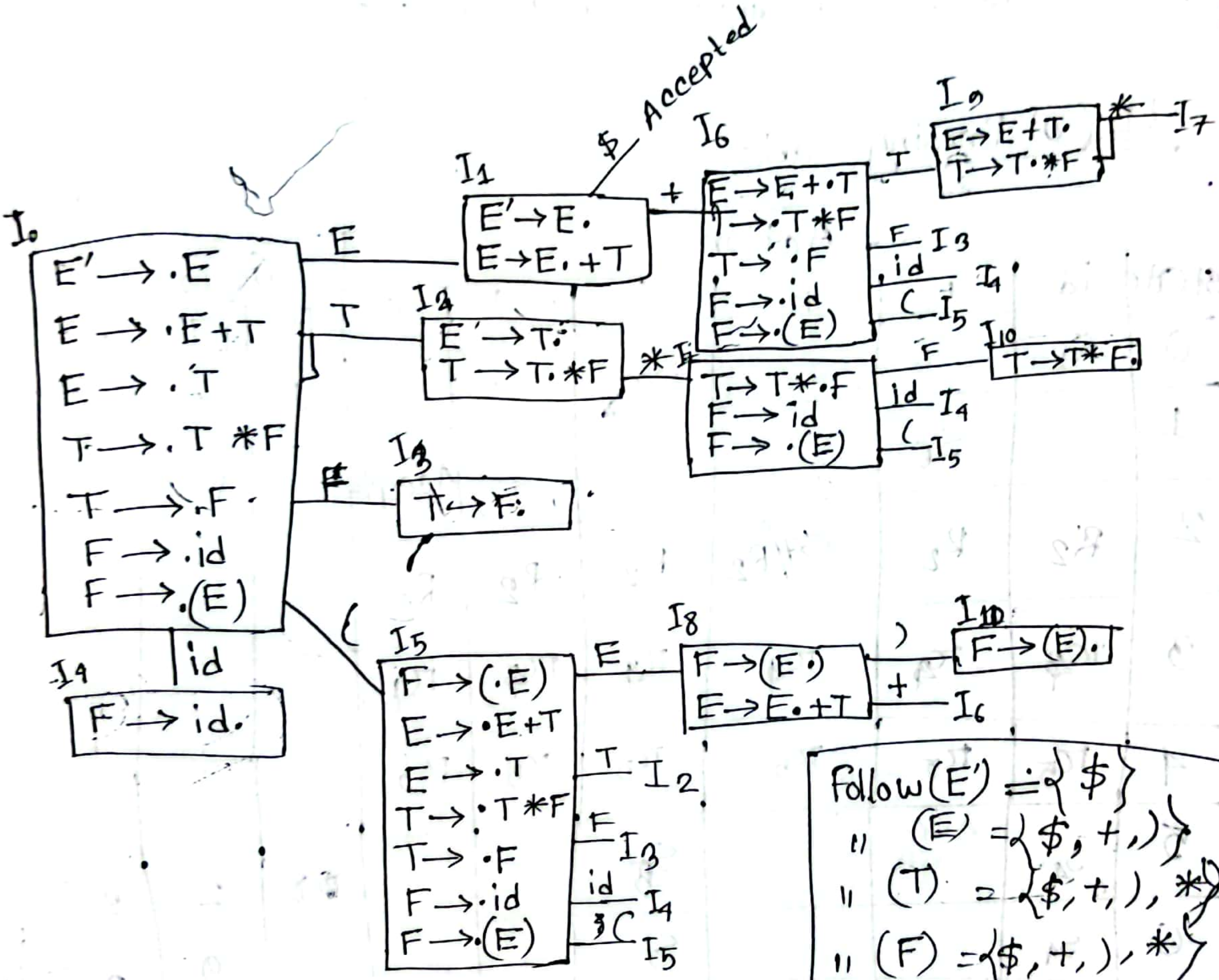
3 T → .T*F

4 T → .F

5 F → .id

6 F → (E)

Step 3 Transition Diagram



Follow(E') = {$}
" (E) = {$, +,)}
" (T) = {$, +,), *}
" (F) = {$, +,), *}

Step 4 SLR Parsing Table

	Action						Go To		
State	id	+	*	(	)	$	E	T	F
0	S4			S5			1	2	3
1		S6				Accepted			
2		r2	S7		r2	r2			
3		r4	r4		r4	r4			
4		r5	r5		r5	r5			
5	S4			S5			8	2	3
6	S4			S5				9	3
7	S4			S5					10
8		S6			S11				
9		r1	S7		r1	r1			
10		r3	r3		r3	r3			
11		r6	r6		r6	r6			

(I) Shift - Reduce conflict $\boxed{S/R}$

(II) Reduce - Reduce conflict $\boxed{R_1/R_2}$

LA(0) Parsing Table :

	Action						Go To		
state	id	+	*	(	)	\$	E	T	F
0	S_4			S_5			1	2	3
1		S_6				Accepted			
2	R_2	R_2	S_7/R_2	R_2	R_2	R_2			
3	r_4	r_4	r_4	r_4	r_4	r_4			
4	r_5	r_5	r_5	r_5	r_5	r_5			
5	S_4			S_5			8	2	3
6	S_4			S_5				9	3
7	S_4			S_5					10
8		S_6			S_{11}				
9	r_1	r_1	S_7/r_1	r_1	r_1	r_1			
10	r_3	r_3	r_3	r_3	r_3	r_3			
11	r_6	r_6	r_6	r_6	r_6	r_6			

* Grammar

$S \longrightarrow AA$

$A \longrightarrow aA$

$A \longrightarrow b$

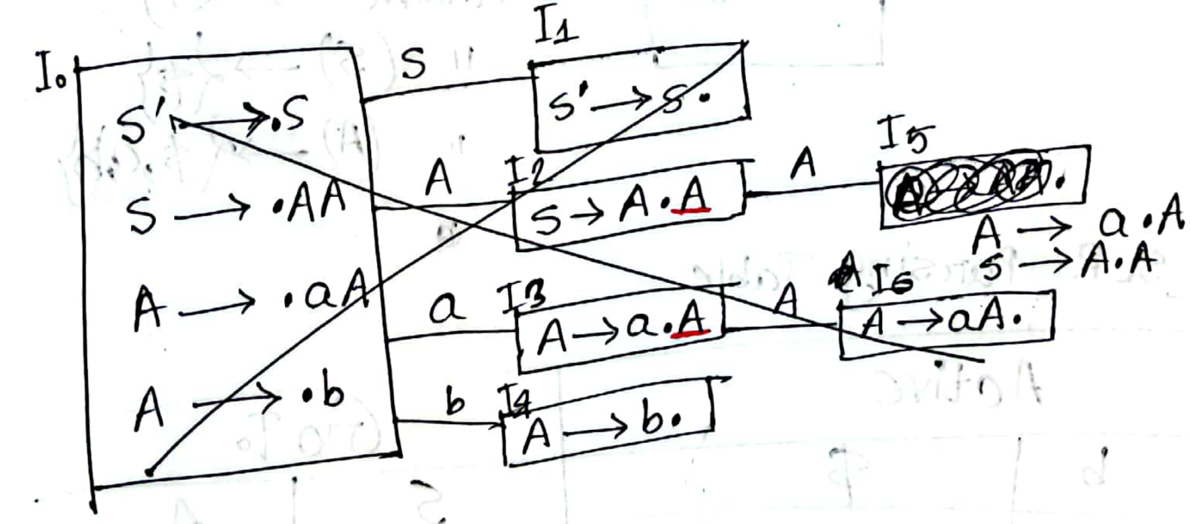
Step1 Augmentation

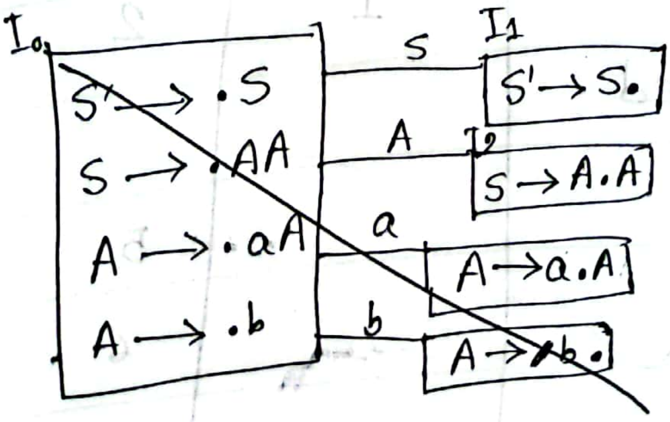
0. $S' \longrightarrow S$
1. $S \rightarrow AA$
2. $A \longrightarrow aA$
3. $A \longrightarrow b$

Step2

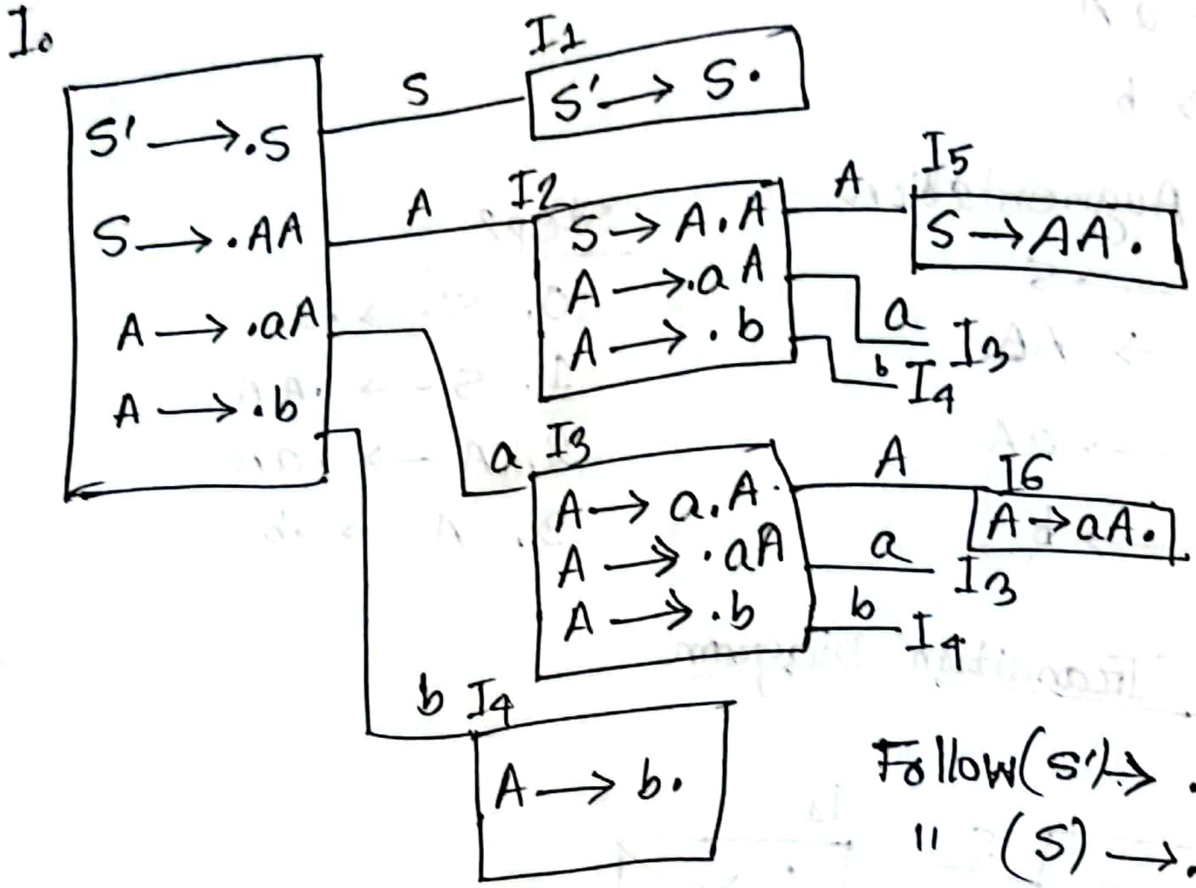
0. $S' \rightarrow .S$
1. $S \rightarrow .AA$
2. $A \rightarrow .aA$
3. $A \rightarrow .b$

Step 3 Transition Diagram





Step 3 Transition ~~Table~~ Diagram



Follow(S') → {\$}
" (S) → {\$}
" (A) → {\$, a, b}

Step4 SLR Parsing Table

	Active			GoTo	
state	a	b	\$	S	A
0	S_3	S_4	~~Accepted~~	1	2
1			Accepted		
2	S_3	S_4			5
3	S_3	S_4			6
4	r_3	r_3	r_3		
5			r_1		
6	r_2	r_2	r_2		

~~Step5~~

LR(0) Parsing Table

	Active			GoTo	
state	a	b	$	S	A
0	S3	S4		1	2
1			Accepted		
2	S3	S4			5
3	S3	S4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

Lab

command prompt

→ Ctrl Alt t

→ . pwd

list of file and list of folder

→ ls

downloads

→ cd Downloads

update

→ sudo apt-get update

flex install

→ sudo apt-get install flex

To create folder / directory

→ mkdir flex

→ ls

To return

→ cd ..

To create a flex file

→ touch 1.l

Ch → change

~~Ch mod~~

Chmod

To edit 1.l

→ gedit 1.l

Execute

→ chmod +x 1.l

code study
→ flex 1.l
→ gcc lex.yy.c
→ ./a.out
→ abcd

Rule section

identifier ^[a]+ [b]*

floating point : [digi]* . [digi]+
0-9 0-9

Lec

Grammar LR(CLR)

S → L=R
S → R
L → *R
L → id
R → L

Step1

0. S' → •S
1. S → L=R
2. S → R
3. L → *R
4. L → id
5. R → L

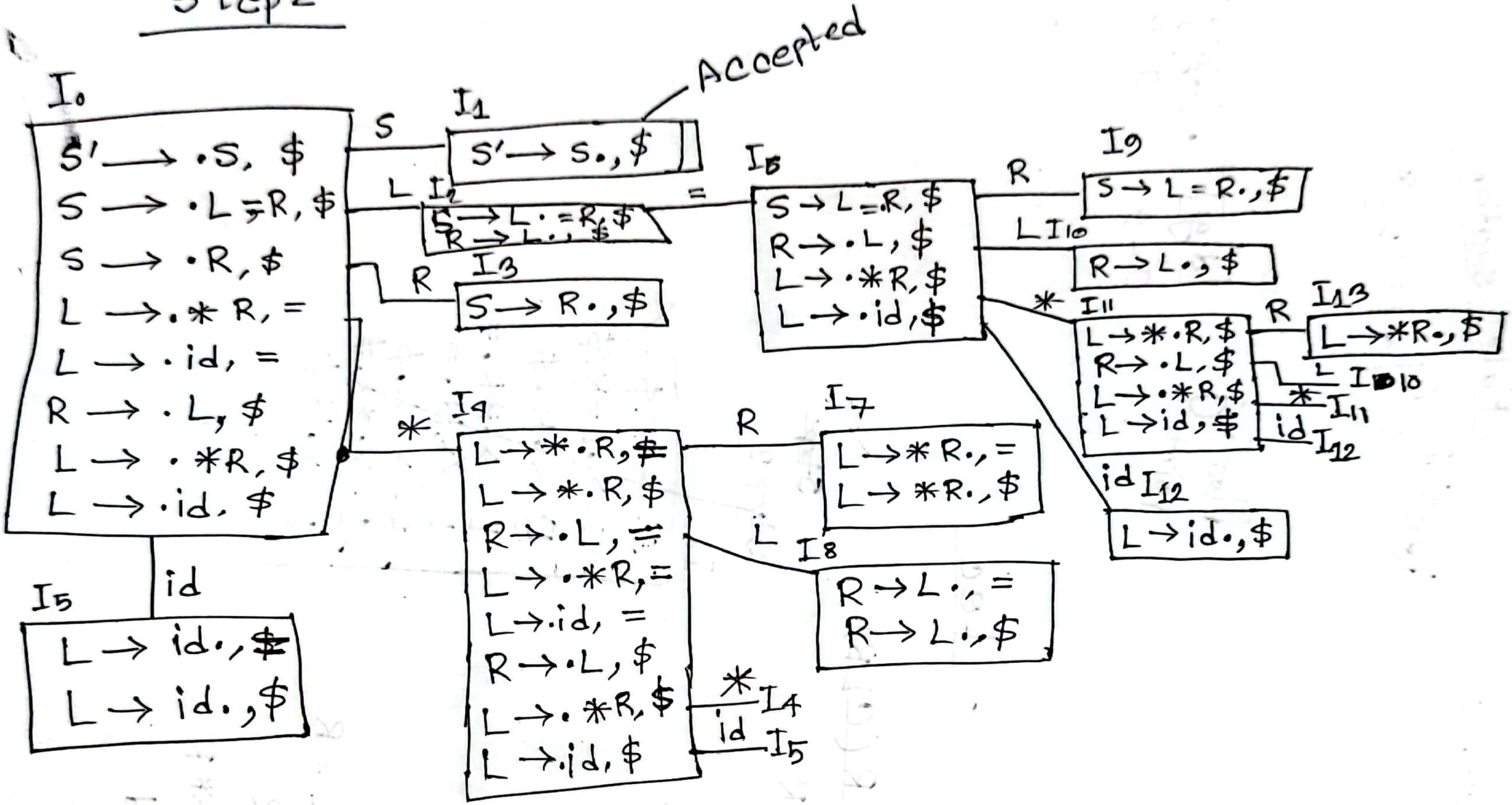
~~Step~~

Step 2

I₀

~~S' → •S, $~~
~~S → •L=R, $~~
~~S → •R, $~~
~~L → •*R, =~~
~~L → •id, =~~
~~R → •L, $~~
~~L → •*R, $~~
L → •id, $

Step2


Accepted
I0
S'→.S, $
S→.L=R, $
S→.R, $
L→.*R, =
L→.id, =
R→.L, $
L→.*R, $
L→.id, $
S
I1
S'→S., $
L I2
S→L.=R, $
R→L., $
R
I3
S→R., $
id
I5
L→id., =
L→id., $
*
I4
L→*.R, =
L→*.R, $
R→.L, =
L→.*R, =
L→.id, =
R→.L, $
L→.*R, $
L→.id, $
* I4
id I5
R
I7
L→*R., =
L→*R., $
L I8
R→L., =
R→L., $
=
I6
S→L=.R, $
R→.L, $
L→.*R, $
L→.id, $
R
I9
S→L=R., $
L I10
R→L., $
* I11
L→*.R, $
R→.L, $
L→.*R, $
L→id, $
R
I13
L→*R., $
L I10
* I11
id I12
id I12
L→id., $

Step-3 (CLR)

	Active				GOTO		
State	*	id	=	$	S	L	R
0	S4	S5			1	2	3
1				~~acc~~ ~~$ Ac~~ Accepted			
2			S6	r5			
3				r2			
4	S4	S5				8	7
5		~~r4~~	r4	r4			
6	S11	S12				10	9
7			r3	r3			
8			r5	r5			
9				r1			
10				r5			
11	S11	S12				10	13
12				r4			
13				r3			

Step4 LALR(1)

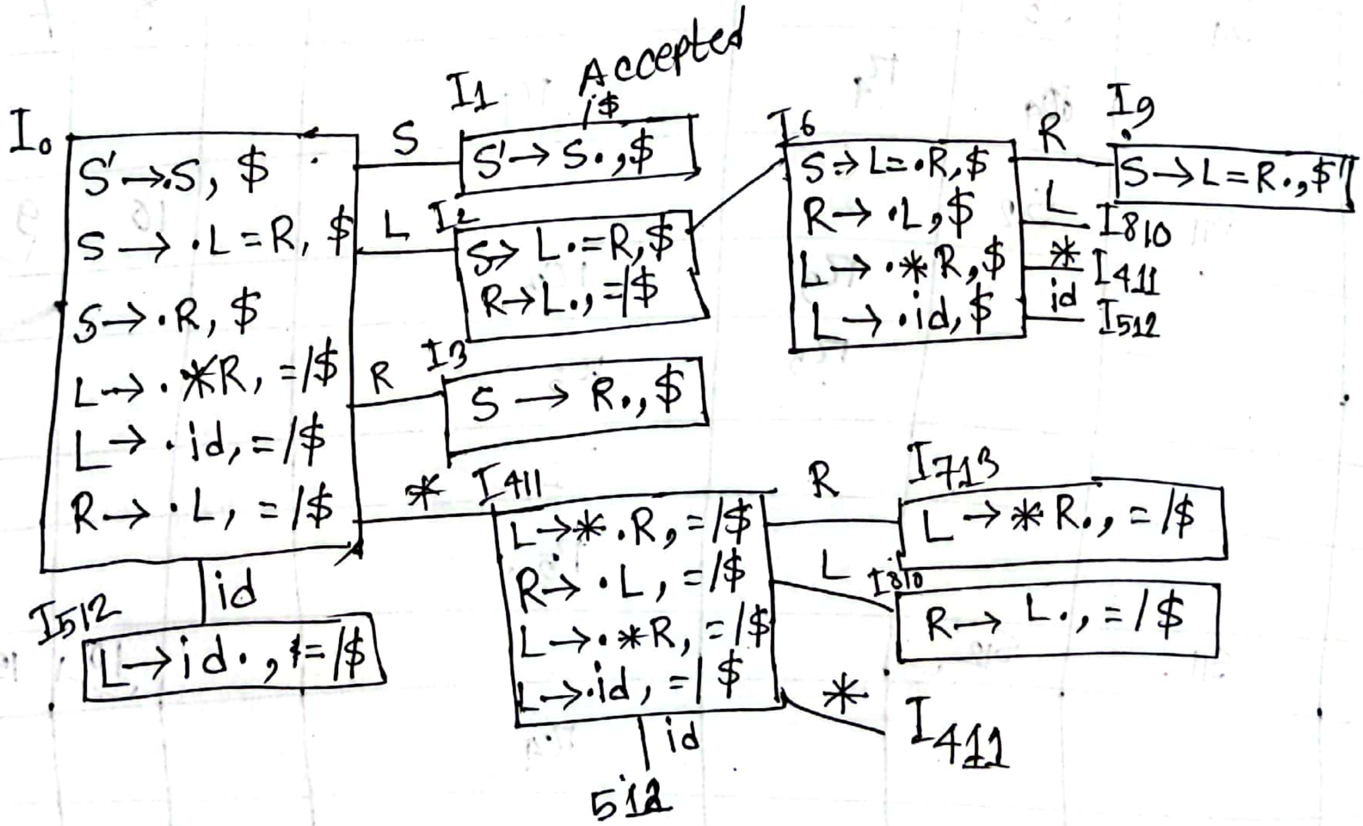
Same cores

I_4 & $I_{11} \longrightarrow I_{411}$

I_5 & $I_{12} \longrightarrow I_{512}$

I_7 & $I_{13} \longrightarrow I_{713}$

I_8 & $I_{10} \longrightarrow I_{810}$



$S \longrightarrow AA$

$A \longrightarrow aA \mid b$

LALR(1)

① Annotated parse tree

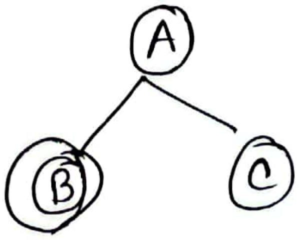
② Dependency Graph.

· Syntax Directed Definition:

2 types Attributes

(i) Synthesized Attributes

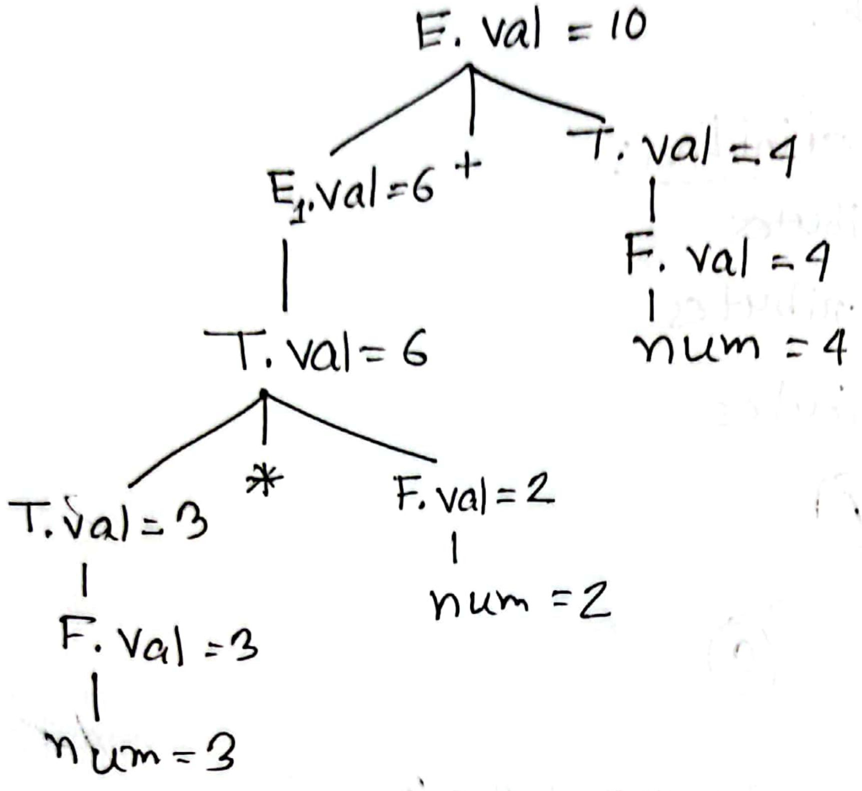
(ii) Inherited Attributes



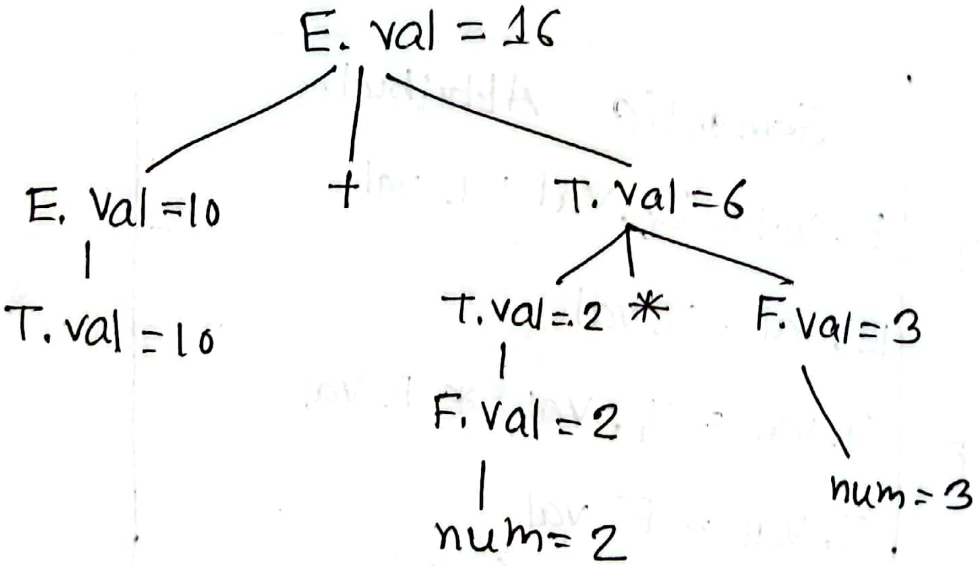
Input : a) 3 * 2 + 4   b) 5 * 2 + 2 * 3

Production Rules	Semantic Attributes
$E \rightarrow E_1 + T$	$E.Val = E_1.Val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.Val$
$T \rightarrow F$	$T.Val = F.val$
$F \rightarrow num$	$F.Val = Value(num)$
$F (E)$	$F.Val = E.val$

Soln: a) Annotated parse tree:



b)


E. val = 16
E. Val = 10
+
T. Val = 6
T. val = 10
T. val = 2
*
F. val = 3
F. val = 2
num = 3
num = 2