Date:
P. No:

ANUBHAV SINGH
IC-2K16-54
COMPILER DESIGN
IC-901 A
MCA 9th SEM.
07-FEB-2021

Anubhav

Date:
P. No:

Anubhav

Date:
P. No: 01

Q.1. Explain the concept of Runtime environment.

Runtime environment or RTE is a state of the target machine, which may include software libraries, environment variable, etc. to provide services to the processes running in the system. When s/w developers write programs, they need to test them in the runtime environment. Therefore, s/w development prog often include an RTE component that allows the programmer to test the program while it is running. This allow the program to be run in an envt where the programmer can track the instructions being processed by the program & debug any errors that may arise. If the program crashes, the RTE s/w keeps running & may

Anubhav

Date:
P. No: 02

provide important informations about why the program crashes.

When you see the variety of a S/w program with the initials "RTE" after it, it usually means the S/w include a RTE.

While developer use RTE S/w to build programs. RTE programs are available to everyday computer users as well. Eg. Adobe flash player.

Anubhav

Date:
P. No: 03

Q.2. Find the host language & target language for the compilers of the programming language. FORTRAN, C, C++, JAVA.

Host language :-

C → C

C++ → C & C++

JAVA → C

FORTRAN → Machine language

Target language

C → .obj

C++ → .obj

JAVA → .class

FORTRAN → objectN.O

Anubhav

Date:
P. No: 04

Q.3. Identify the lexemes that make up the token in the following program:

```
int max (i, j)
int i, j;
{
  return i > j ? i : j;
}
```

Ans

Lexeme	Token
int	Keyword
max	Identifier
(	Operator
i	Identifier
,	Operator
j	Identifier
)	Operator
return	Keyword
>	Operator
?	Operator
:	operator
{	operator
}	operator

Anubha

Date:
P. No: 05-

Q.4. consider the following grammer

$E \to E+T \mid T$

$T \to TF \mid F$

$F \to F* \mid a \mid b$

construct the SLR parsing table.

Ans. $E' \to .E$ ← Augumented

$E \to E+T \mid T$

$T \to TF \mid F$

$F \to F* \mid a \mid b$

1) The closure operation

{

$E' \to .E$

$E \to .E+T \mid .T$

$T \to .TF \mid .F$

$F \to .F* \mid .a \mid .b$ } $= I_0$

Compute goto for I_0

goto $(I_0, E) \to$ { $E' \to E.$

$E \to E.+T$ } $= I_1$

Answer

Date:
P. No: 06

goto $(I_0, T) \rightarrow$ { $E \rightarrow T.$
$T \rightarrow T. F$
$F \rightarrow .F* | .a | .b$ } $= I_2$

goto $(I_0, F) \rightarrow$ { $T \rightarrow F.$
$F \rightarrow F.*$ } $= I_3$

goto $(I_0, a) \rightarrow$ $F \rightarrow a. = I_4$
goto $(I_0, b) \rightarrow$ $F \rightarrow b. = I_5$

compute goto for I_1

{ $E' \rightarrow E.$
$E \rightarrow E. + T$ } $= I_1$

goto $(I_1, +) \rightarrow$ { $E \rightarrow E + .T$
$T \rightarrow .TF | .F$
$F \rightarrow .F* | .a | .b$
} $= I_6$

compute goto for I_2
{ $E \rightarrow T.$
$T \rightarrow T.F$
$F \rightarrow .F* | .a | .b$ } $= I_2$

Anuhar

Date:
P. No: 07

goto $(I_2, F) \rightarrow \{ T \rightarrow TF.$
$F \rightarrow F.* \} = I_7$

goto $(I_2, a) \rightarrow \{ F \rightarrow a. \} = I_4$

goto $(I_2, b) \rightarrow \{ F \rightarrow b. \} = I_5$

compute goto for I_6

goto $(I_6, T) \rightarrow \{ E \rightarrow E+T.$
$T \rightarrow T.F$
$F \rightarrow .F* \mid .a \mid .b \} = I_9$

goto $(I_6, F) \rightarrow \{ T \rightarrow F.$
$F \rightarrow F.* \} = I_3$

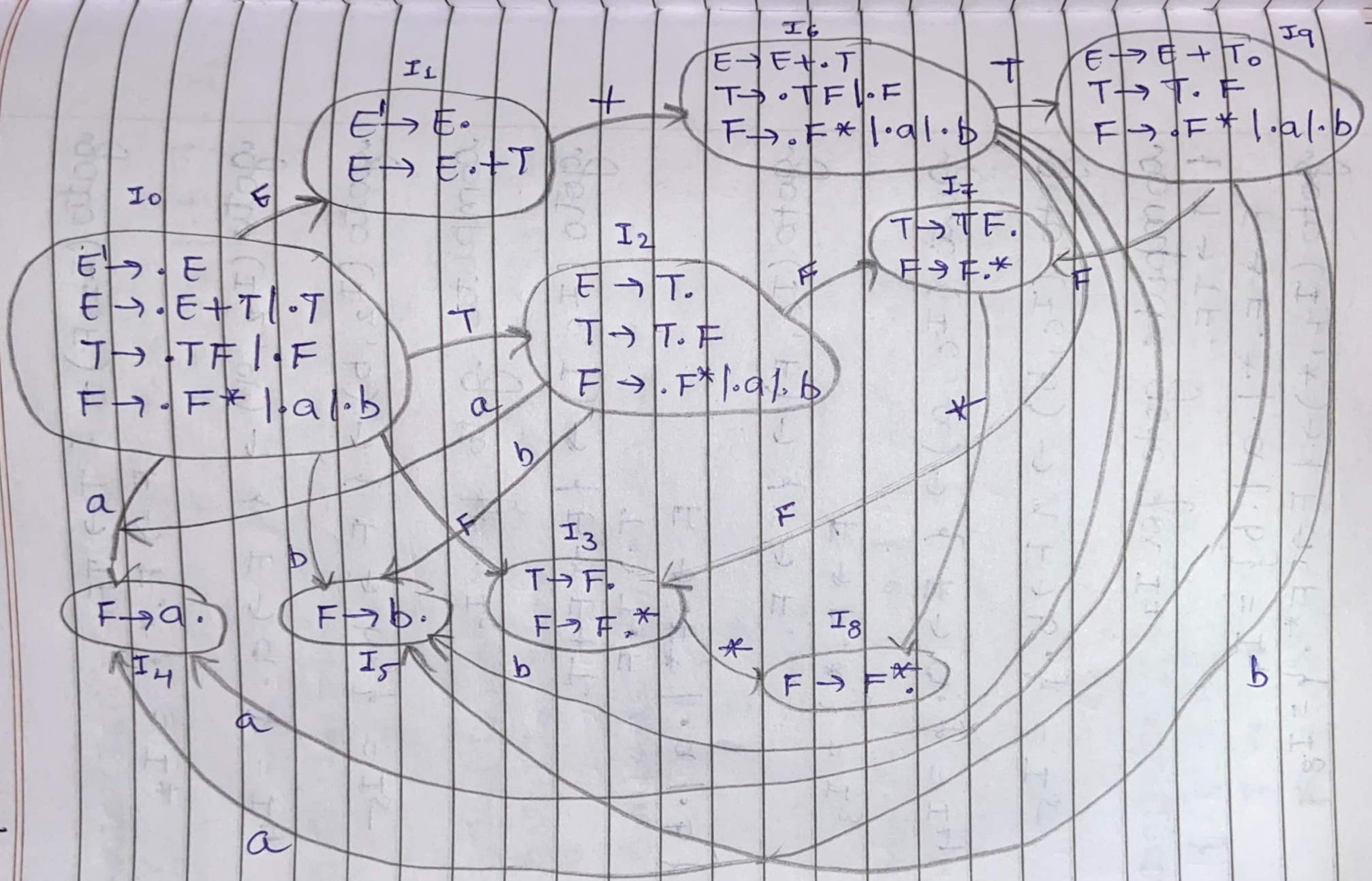
goto $(I_6, a) \rightarrow \{ F \rightarrow a. \} = I_4$

goto $(I_6, b) \rightarrow \{ F \rightarrow b. \} = I_5$

compute goto for I_7
$\{ T \rightarrow TF.$
$F \rightarrow F.* \mid .a \mid .b \} = I_7$
goto $(I_7, *) \rightarrow \{ F \rightarrow F*. \} = I_8$

Date:
P. No: 08

Answer



Anubha

Date:
P. No: 09

State	Action					Goto		
	a	+	b	*	$	E	T	F
I_0	S_4		S_5			1	2	3
I_1		S_6			Accept			
I_2	S_4		S_5		R_2			7
I_3		R_4		S_8	R_4			
I_4	R_6	R_6	R_4		R_6			
I_5	R_7	R_7	R_7		R_7			
I_6	S_4		S_5	S_3			9	3
I_7		R_3		S_8	R_3			
I_8	R_5	R_5	R_5		R_5			
I_9	S_4		S_5		R_1			7

Anubha

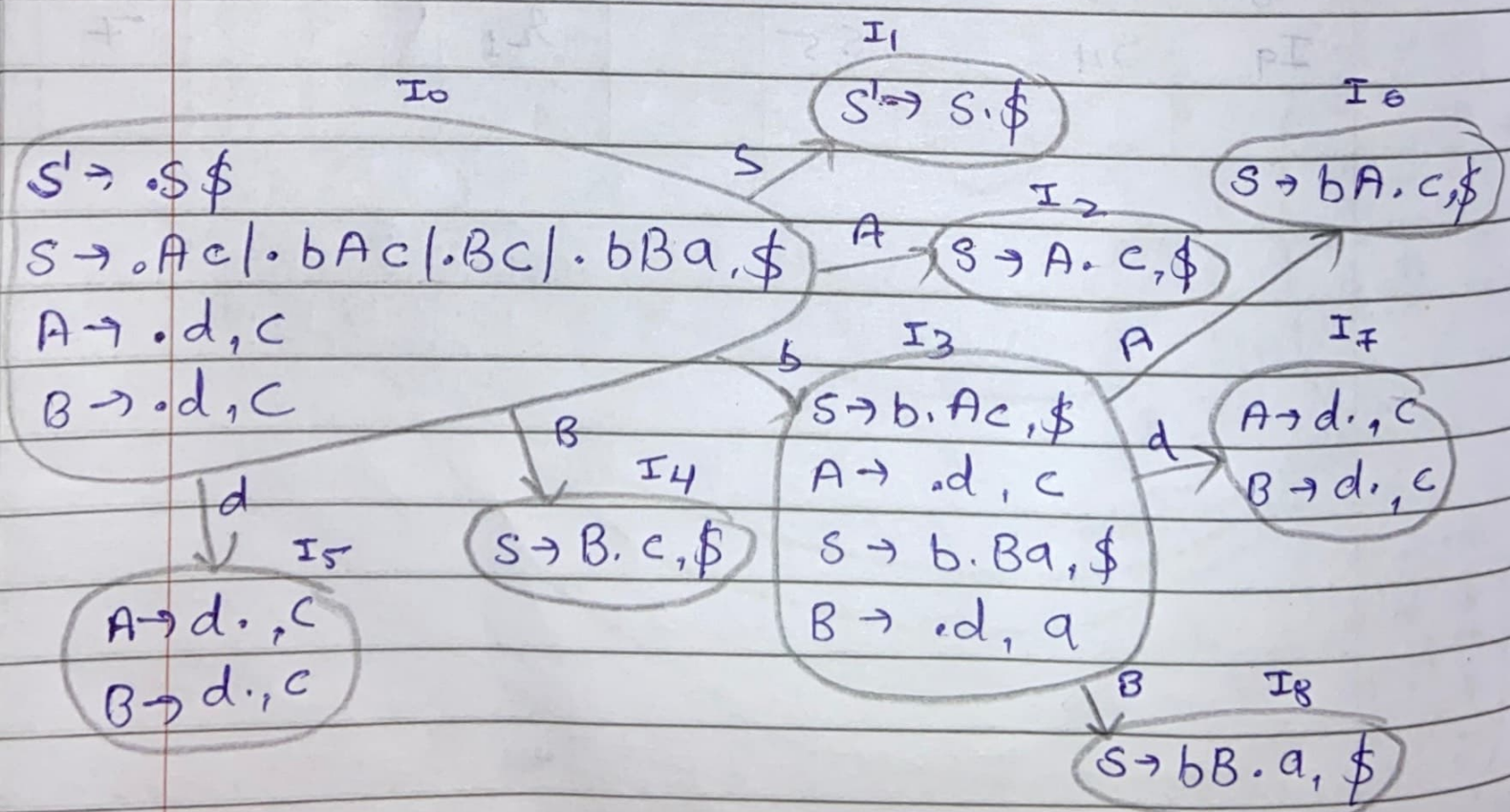
Date:
P. No: 10

Q.5 Show that the following grammer is LR(1) but not LALR(1)

$S \to Ac \mid bAc \mid Bc \mid bBa$
$A \to d$
$B \to d$

Ans. Argumental grammer :-

$S' \to .S, \$$
$S \to .Ac \mid .bAc \mid .Bc \mid .bBa, \$$
$A \to .d, c$
$B \to .d, c$



Anuhar

Date:
P. No: 11

State	Action				Goto		
	b	c	d	$	S	A	B
I_0	S_3		S_5		1	2	4
I_1				Accept			
I_2							
I_3			S_7			6	8
I_4							
I_5							
I_6							
I_7							
I_8							

#Anubhav

Date:
P. No: 12

Q. 6 For the input expression (4*7+1)*2, construct an annoted name tree according to given grammer.

$L \rightarrow E_x$

$E \rightarrow E_1 + T$

$E \rightarrow T$

$T \rightarrow T_1 * F$

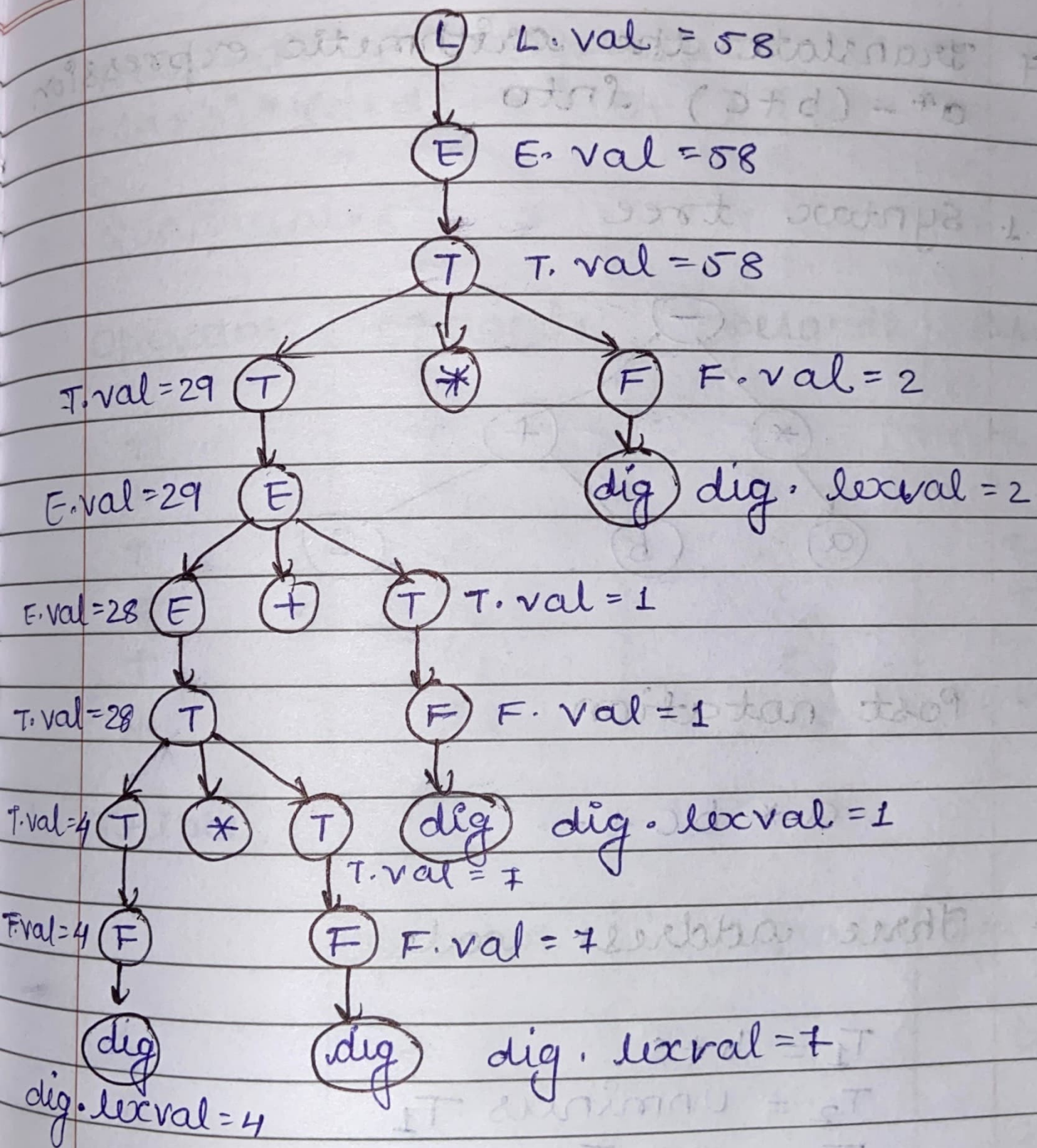
$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow$ digit

Anuthar

Date:
P. No: 13

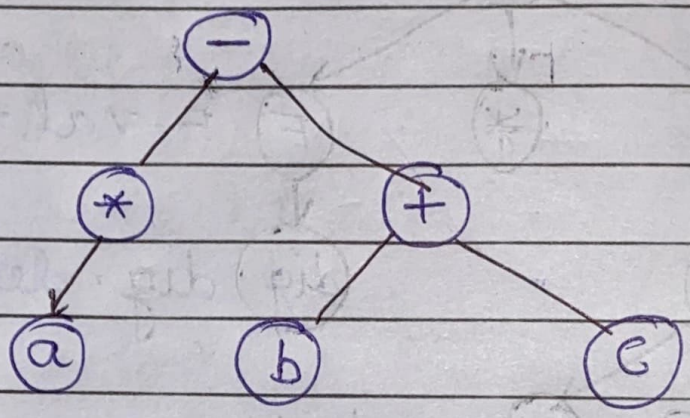


Answer

Date:
P. No: 14

Q.7 Translate the arithmetic expression $a * -(b+c)$ into

Ans.1. Syntax tree



2. Post notation

$a * bc + -$

3. Three address code

$T_1 = b + c$

$T_2 = \text{unminus } T_1$

$T_3 = a * T_2$

Answer

Date:
P. No: 15

Q.8. Translate the expression $-(a+b)*(c+d)+(a+b+c)$ into

a. Quadruples

Operator	Operand1	Operand2	Result
+	a	b	T_1
−	T_1		T_2
+	c	d	T_3
*	T_2	T_3	T_4
+	T_1	c	T_5
+	T_4	T_5	T_6

b. Triples

	operator	operand1	operand2
1	+	a	b
2	−	1	
3	+	c	d
4	*	2	3
5	+	1	c
6	+	3	5

Answer

Date:
P. No: 16

c. Indirect Triples

Index	Pointer		operator	opr 1	opr 2
[100]	1	1	+	a	b
[101]	2	2	-	[100]	
[102]	3	3	+	c	d
[103]	4	4	*	[101]	[102]
[104]	5	5	+	[101]	e
[105]	6	6	+	[102]	[104]

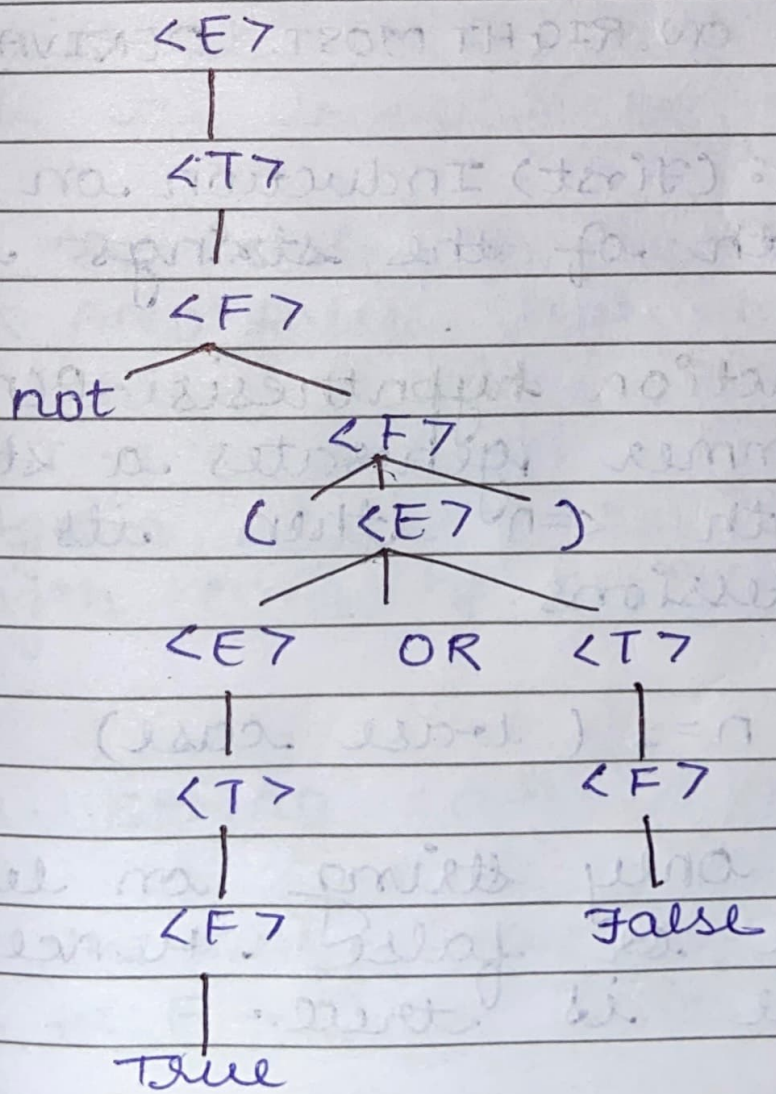
Anubhav

Date:
P. No: 17

Q.9. Consider the following grammer

E → E or T | T
T → T & F | F
F → not F | (E) | true | false

Ans 9. Construct a parse tree for the sentence not (true or false)



Anubha

Date:
P. No: 18

b. Show that the grammer generates all boolean expressions.

Ans. First show that the grammer generates boolean expression Then we show that, given any boolean expression this grammer can generate it.

FOCUS ON RIGHT MOST DERIVATIONS

Proof: (First) Induction on the length of the strings derived.

Induction hypothesis P(n): The grammer generates a string of length <=n then its a boolean expressions.

let n=1 (base case)

The only string on length 1 are true or false. Hence, base case is true.

Anubhav

Date:
P. No: 19

let $n > 1$ assumming P(n). Now consider a string of length n+1.

case 1:- Either the string ends with a OR true or OR false. If this happens then the derivation was due to the production $E \to E$ OR T, now clearing the string generated by E is length (n-1) by induction hypothesis the string of length (n+1 is a boolean expression).

case 2:- The string end with AND true or AND false. Then the only way this could happen is

$E \to T \to T$ AND F.

Again since T derives strings of length (n-1) & $E \to T$ we are done.

case 3:- String ends with)

The only way this can happen is $F \to [E]$. Clearly, E

Anuhar

Date:
P. No: 20

derives strings of length n-1 we are done.
Now, to show the second part use induction. That is given a string which is boolean expression, show by induction on the length of the string you can derive it.

c) Is this grammer ambiguous? why?

No, the grammer is not ambiguous. as for the given grammer their does not exists more than one leftmost / rightmost derivative or more than one parse tree.

Anubhav

Date:
P. No: 21

Q.10 Construct an SLR parsing table for the grammer

E → E or T | T
T → T and F | F
F → not F | (E) | true | false

1. Augumented grammer :-

E' → E
E → E or T
E → T
T → T and F
T → F
F → not F
F → (E)
F → true
F → false

2. Closure Operation :-

I0 : E' → . E
E → . E or T
E → . T
T → . T and F
T → . F
F → . not F

Ayushar

Date:
P. No: 22

$F \rightarrow .(E)$
$F \rightarrow .\text{true}$
$F \rightarrow .\text{false}$

Now applying goto action

goto $(I_0, E) \rightarrow \{ E' \rightarrow E.$
$E \rightarrow E. \text{ or } T \} = I_1$

goto $(I_0, T) \rightarrow \{ E_0 \rightarrow T_0$
$T \rightarrow T_0 \text{ and } F \} = I_2$

goto $(I_0, F) \rightarrow \{ T \rightarrow F_0 \} = I_3$

goto $(I_0, \text{not}) \rightarrow \{ F \rightarrow \text{not} . F$
$F \rightarrow .\text{not } F$
$F \rightarrow .(E)$
$F \rightarrow .\text{true}$
$F \rightarrow .\text{false} \} = I_4$

goto $(I_0, C) \rightarrow \{ F \rightarrow (.E)$
$E \rightarrow .E \text{ or } T$
$E \rightarrow .T$
$T \rightarrow .T \text{ and } F$

Anuhar

Date:
P. No: 23

T → . F
F → . not F
F → . (E)
F → . true
F → . false } = I_5

goto (I_0, true) → { F → true . . } = I_6

goto (I_0, false) → { F → false . } = I_7

goto (I_1, or) → { E → E or . T
T → . T and F
T → . F
F → . not F
F → . (E)
F → . true
F → . false } = I_8

goto (I_2, and) → { T → T and . F
F → . not (F)
F → . (E)
F → . true
F → . false } = I_9

Answer

Date:
P. No: 24

goto $(I_4, F) \rightarrow \{E \rightarrow not\ F \cdot\} = I_{10}$

goto $(I_4, C) \rightarrow \{E \rightarrow (\cdot E)$
$E \rightarrow \cdot E\ or\ T$
$E \rightarrow \cdot T$
$T \rightarrow \cdot T\ and\ F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot not\ (F)$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot true$
$F \rightarrow \cdot false\ \} = I_5$

goto $(I_4, true) \rightarrow \{F \rightarrow true \cdot\} = I_6$

goto $(I_4, false) \rightarrow \{F \rightarrow false \cdot\} = I_7$

goto $(I_4, not) \rightarrow \{F \rightarrow not \cdot F$
$F \rightarrow \cdot not\ F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot true$
$F \rightarrow \cdot false\ \} = I_4$

goto $(I_5, E) \rightarrow \{F \rightarrow (E \cdot)$
$E \rightarrow E \cdot or\ T\ \} = I_{11}$

Anurha

Date:
P. No: 25

goto $(I_5, T) \rightarrow \{E \rightarrow T.$
$T \rightarrow T.$ and $F\} = I_{12}$

goto $(I_5, F) \rightarrow \{T \rightarrow F.\} = I_3$

goto $(I_5, \text{not}) \rightarrow \{F \rightarrow \text{not} \cdot F$
$F \rightarrow \cdot \text{not} F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \text{true}$
$F \rightarrow \cdot \text{false}\} = I_4$

goto $(I_5, C) \rightarrow \{F \rightarrow (\cdot E)$
$E \rightarrow \cdot E$ or T
$E \rightarrow \cdot T$
$E \rightarrow \cdot T$ and F
$T \rightarrow \cdot F$
$F \rightarrow \cdot \text{not} (F)$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \text{true}$
$F \rightarrow \cdot \text{false}\} = I_5$

goto $(I_8, T) \rightarrow \{E \rightarrow E$ or $T.$
$T \rightarrow T.$ and $E\} = I_{13}$

Anubhav

Date:
P. No: 26

goto $(I_8, F) \rightarrow \{T \rightarrow F.\} = I_8$

goto $(I_8, not) \rightarrow \{F \rightarrow not \cdot F$
$F \rightarrow .notF$
$F \rightarrow .(E)$
$F \rightarrow .true$
$F \rightarrow .false\} = I_4$

goto $(I_8, C) \rightarrow \{F \rightarrow C \cdot E)$
$E \rightarrow E$ or T
$E \rightarrow .T$
$T \rightarrow .T$ and E
$T \rightarrow .F$
$F \rightarrow .not(F)$
$E \rightarrow .(E)$
$F \rightarrow .true$
$F \rightarrow .false\} = I_5$

goto $(I_8, true) \rightarrow F \rightarrow true. = I_6$

goto $(I_8, false) \rightarrow F \rightarrow false. = I_7$

goto $(I_9, F) \rightarrow T \rightarrow T$ and $F. = I_{14}$

Grammar

Date:
P. No: 27

goto (Iq , not) → { F → not F
F → . not F
F → . (E)
F → . true
F → . false } = I4

goto (Iq, C) → { F → (. E)
E → . E OR T
E → . T
T → . T and F
T → . F
F → . not (F)
F → . (E)
F → . true
F → . false } = I5

goto (Iq, true) = { F → true . } = I6

goto (Iq, false) = { F → false . } = I7

goto (I11 ,)) ⇒ F → (E) . = I15

Answer

Date:
P. No: 28

goto (I_{11}, Or) = { E → E Or . T
T → . T and F
T → . F
F → . not (F)
F → . (E)
F → . true
F → . false } = I_8

goto (I_{12}, and) → { T → T and . F
F → . not (F)
F → . (E)
F → . true
F → . false } = I_9

goto (I_{13}, and) → { T → T and . F
F → . not(F)
F → . (E)
F → . true
F → . False } = I_9

Anubhav

Date:
P. No: 29

state	Action								Goto		
	OR	AND	NOT	TRUE	FALSE	(	)	$	E	T	F
I_0			S_4	S_6	S_7	S_5			1	2	3
I_1	S_8							acc			
I_2	r_2	S_9					r_2	r_2			
I_3	r_4	r_4					r_4	r_4			
I_4				S_6	S_7	S_5					10
I_5						S_5			11	12	3
I_6	r_7	r_7					r_7	r_7			
I_7	r_8	r_8					r_8	r_8			
I_8				S_6	S_7	S_5				13	3
I_9			S_4	S_6	S_7	S_5					14
I_{10}	r_5	r_5					r_5	r_5			
I_{11}	S_8						S_{15}				
I_{12}	r_3	S_9/r_3					r_3	r_3			
I_{13}	r_1	S_9					r_1	r_1			
I_{14}	r_3	r_3					r_3	r_3			
I_{15}	r_6	r_6					r_6	r_6			

Anuman

Date:
P. No: 30

Q. 11. Translate the executable statement of the following C program into three address code.

```
main()
{
  int i;
  int a[10];
  i=1;
  while (i<=10)
  {
    a[i]=0;
    i = i+1;
  }
}
```

Sol:-

```
a[10]
i=1
L1: if i<=10 goto L2
    goto last
L2: T1 = 4 x i
    T2 = a[T1] = 0
    T3 = i+1
    i = T3
    goto L2
last
```

Answer

Date:
P. No: 31

Q.12 Show that no (LR(1)) Grammer is ambiguous.

Ans. According to definition, all LR(1) grammer are unambiguous. But the converse is not true. The fact that the grammer is unambiguous does not say anything about whether it can be parsed with an LR(1) parser.

The grammer you present is not LR(1) although all language itself is. In fact, the language is regular:

$(aa)^*$

But that's not true for the language of even length palindromes which has a rather similar unambiguous CFG:

$S \rightarrow \epsilon$

$S \rightarrow aSa$

$S \rightarrow bSa$

Anuran

Date:
P. No: 32

Intuitively, the problem with parsing palindromes deterministically is that we have to start popping the stack at the middle of the sentence. But we can't tell where the middle of sentence is until we reach the end & since there is no limit on lenght on the length of sentence, the end could be arbitarily distant from the middle. So, no finite lookahead is sufficient to make the decision.

A context free language is LR(x) predisely, if it is deterministic.

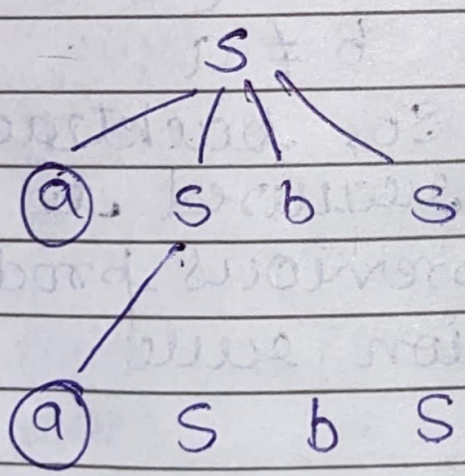
Anuhar

Date:
P. No: 33

Q.13 consider the following grammer:
$S \rightarrow aSbS \mid bSaS \mid \epsilon$
construct a recursive-descent parser with backtracking for the grammer. can a predictive parser be generated for this grammer?

Ans. Top down parsing

Input string $w = \underline{a}baab$

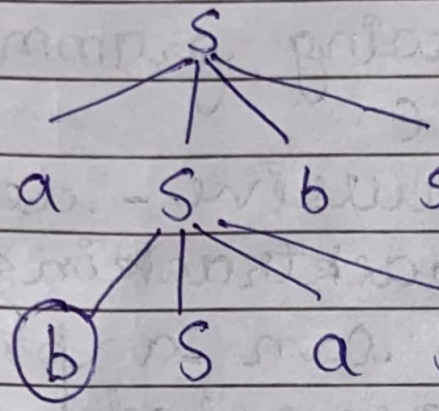


(1) a
As a = a
No backtracking

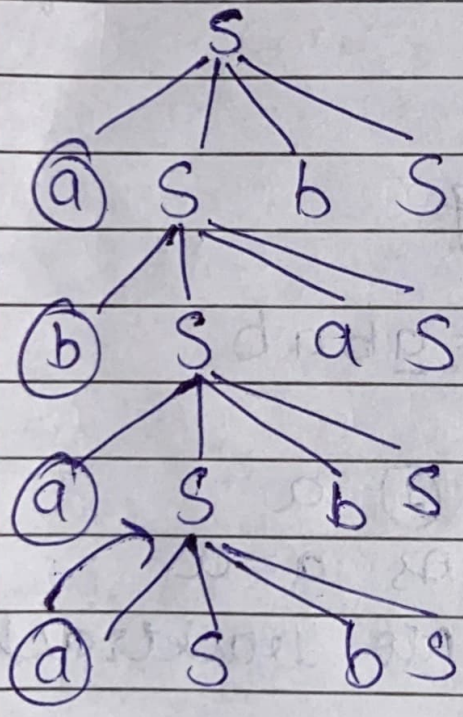
(2) b
As a ≠ b
Backtracking to the previous production rule

Answer

Date:
P. No: 34



as b = b
No back-tracking required



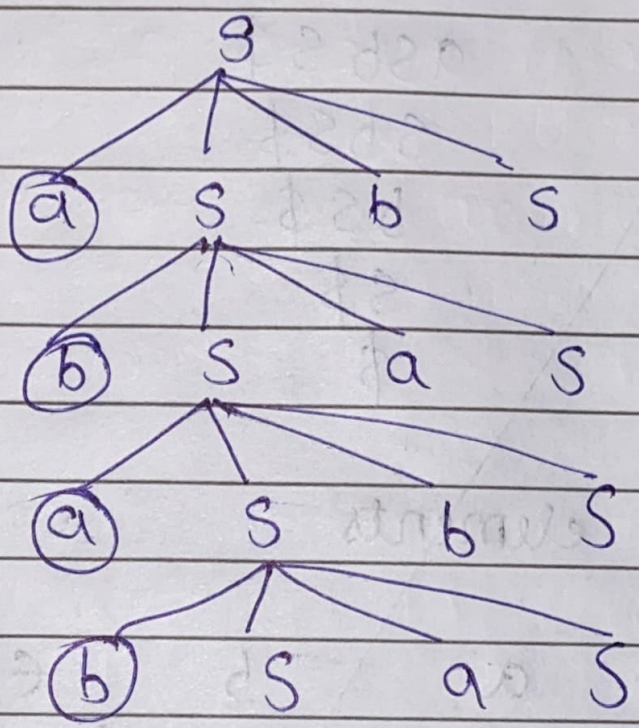
Now (3) a
as a = a
No backtracking required

Now (4) b
b ≠ a
So, backtracking required to previous production rule

Anubhav

Date:
P. No: 35

Final tree:



This is a recursive desent parser for input string w=abab & it gets accepted.

And, yes a predictive parser can be genrated for this grammer.

Anubhav

Date:
P. No: 86

Stack	Input	Action
S$	aSbS$	S → aSbS
aSbS$	aSbS$	pop a
SbS$	SbS$	pop S
bS$	bS$	pop b
S$	S$	pop S
$	$	Accept

stack element ↓

Input →	a	b	∈	$
S	S → aSbS	S → bSaS		
a	pop			
b		pop		
∈			pop	
$				success

Anubhav

Date:
P. No: 37

Q.14 Construct a parse tree and syntax tree for the expression ((a)+(b)) according to:

a. The syntax-directed definition

Production	Semantic Rule
$E \rightarrow E_1 + T$	E.nptr := mknode ('+', E_1.nptr, T.nptr)
$E \rightarrow E_1 - T$	E.nptr := mknode ('-', E_1.nptr, T.nptr)
$E \rightarrow T$	E.nptr := T.nptr
$T \rightarrow (E)$	T.nptr := E.nptr
$T \rightarrow id$	T:nptr := makeleaf (id, id.entry)
$T \rightarrow num$	T.nptr := make leaf (num, num.val)

Anubhav

Date:
P. No: 38

Ans. First lets convert the expression to postfix notation

((a) + (b)) → (a) (b) +

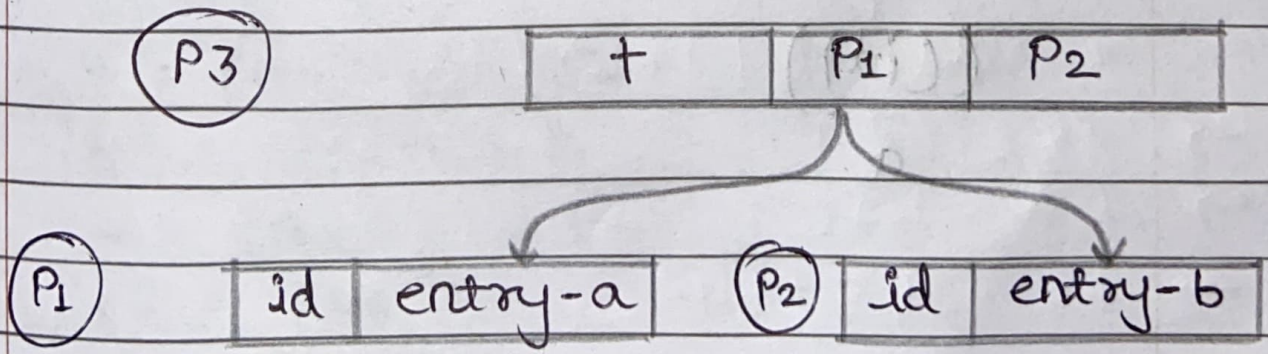
(getting rid of parenthesis)

⇒ ab+

Symbol operation

a P_1 = mkleaf (id, entry-a)
b P_2 = mkleaf (id, entry-b)
+ P_3 = mknodd (+, P_1, P_2)

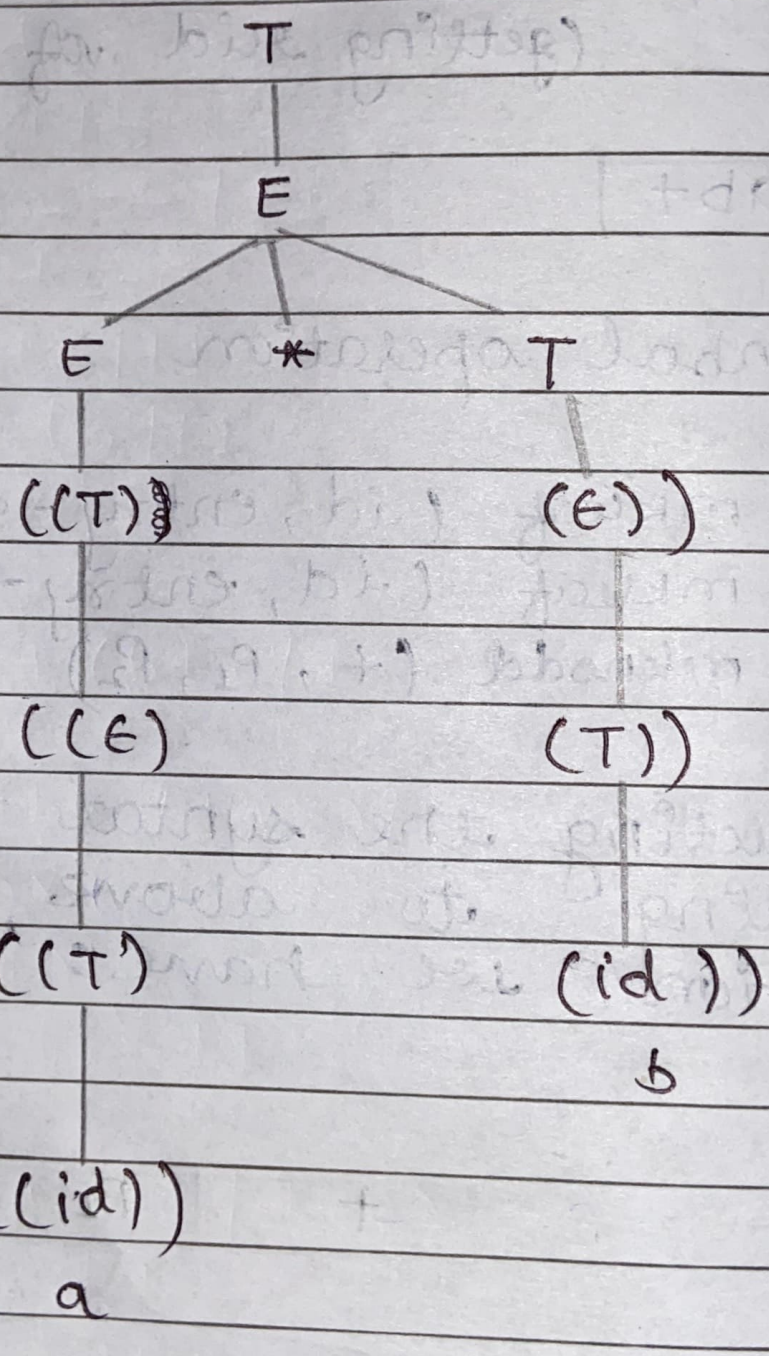
constructing the syntax tree according to above mentioned operations we have:



Anuvhav

Date:
R. No: 39

Parse tree for the expression ((a)+(b)) according to SDD :



Anuhar

Date:
P. No: 40

Q.15 Generate optimal code for the following assignment statements:

a. $x = a + b * c$

$x = a + b * c$

→ $\because t1 = b * c$

$\therefore \quad x = a + t1$

→ $t2 = a + t1$

→ $\boxed{x = t2}$

b. $x = (a * -b) + (c - (d + e))$

$x = (a * -b) + (c - (d + e))$

→ $t1 = d + e$

$x = (a * -b) + (c - t1)$

→ $t2 = c - t1$

$x = (a * -b) + t2$

→ $t3 = \text{uminus } b$

$x = (a * t3) + t2$

→ $t4 = (a * t3)$

$x = t4 + t2$

→ $t5 = t4 + t2$

$\boxed{x = t5}$

Anwhar

c. $x = (a/b-c)/d$

$x = (a/b-c)/d$

→ $t1 = d$

→ $t2 = a/b$

$x = t2-c/t1$

→ $t3 = t2-e$

$x = t3/t1$

→ $t4 = t3/t1$

$\boxed{x = t4}$

d. $x = a+(b+c/d*e)/(f*g-h*i)$

$x = a+(b+c/d*e)/(f*g-h*i)$

→ $t1 = h*i$

→ $t2 = f*g$

→ $t3 = t2-t1$

$x = a+(b+c/d*e)/t3$

→ $d*e = t4$

→ $t5 = b+c$

→ $t6 = t5/t4$

Anuhar

Date:
P. No: 42

$x = a + t6 \mid t3$

→ $t7 = a + t6$

$x = t7 \mid t3$

→ $t8 = t7 \mid t3$

$\boxed{x = t8}$

e. $a[i,j] = b[i,j] - c[a[k,l]] * d[i+j]$

$a[i,j] = b[i,j] - c[a[k,l]] * d[i+j]$

→ $t1 = d[i+j]$

→ $t2 = a[k,l]$

→ $t3 = c[t2]$

→ $t4 = t3 * t1$

$a[i,j] = b[i,j] - t4$

→ $t5 = b[i,j]$

→ $t6 = t5 - t4$

$\boxed{a[i,j] = t6}$