Solutions To Question Bank (Unit - II)

Ques1. Show whether the following grammar is ambiguous or not. Also construct the parse tree using LMD & RMD.

$E \to E+E \mid E*E \mid (E) \mid id$

input string $id_1 + id_2 * id_3$ <(*: id_1, id_2 & id_3 are different occurrences of same terminal symbol id)>

Ambiguity: A grammar G is said to be ambiguous if there exist a string $X \in L(G)$ such that to generate string X using productions of G there exist more than one left-most derivations or more than one right most derivations or more than one parse trees for X.

LMD1 for $id_1 + id_2 * id_3$

$E \Rightarrow E+E$
$\Rightarrow id_1 + E$
$\Rightarrow id_1 + E*E$
$\Rightarrow id_1 + id_2 * E$
$\Rightarrow id_1 + id_2 * id_3$

LMD2

$E \Rightarrow E*E$
$\Rightarrow E+E*E$
$\Rightarrow id_1 + E*E$
$\Rightarrow id_1 + id_2 * E$
$\Rightarrow id_1 + id_2 * id_3$

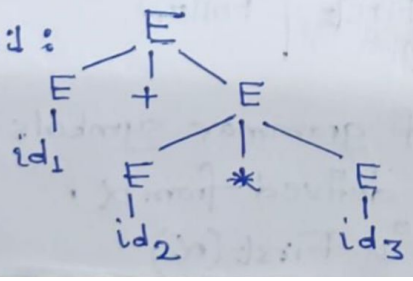
RMD1

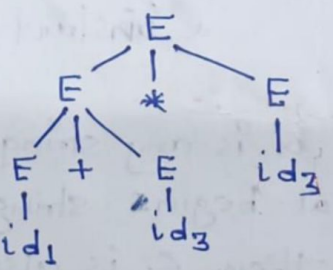
$E \Rightarrow E+E$
$\Rightarrow E+E*E$
$\Rightarrow E+E*id_3$
$\Rightarrow E+id_2*id_3$
$\Rightarrow id_1+id_2*id_3$

RMD2

$E \Rightarrow E*E$
$\Rightarrow E*id_3$
$\Rightarrow E+E*id_3$
$\Rightarrow E+id_2*id_3$
$\Rightarrow id_1+id_2*id_3$

Parse Tree 1:





Hence the Grammer is ambiguous.

Ques2: $S \to iEtSS' \mid a$ First(S) = {i, a}

$S' \to eS \mid E$ First(S') = {e, b}

$E \to b$ First(E) = {b}

LL(1) parsing table

	i	t	a	e	b
S	$S \to iEtSS'$		$S \to a$		
S'				$S' \to eS$	$S \to E$
E					$E \to b$

Since each entry in the table has a unique production.

The given grammar is LL(1)

Q3: $S \to AaBb \mid BbBa$

$A \to \epsilon$ Follow(A) = {a}

$B \to \epsilon$ Follow(B) = {a, b}

Follow(S) = {$}

Remember $A \to \cdot\epsilon$ is same as $A \to \epsilon\cdot$ is same as $A \to \cdot$

State with augmented production

"0"

$S' \to \cdot S$

$S \to \cdot AaBb$

$S \to \cdot BbBa$

$A \to \cdot$

$B \to \cdot$

Let's number the productions

1. $S \to AaBb$
2. $S \to BbBa$
3. $A \to \epsilon$
4. $B \to \epsilon$

In the parsing table for state 0 we need to put r3 under Follow(A) and r4 under Follow(B)

	a	b	$	S	A	B
0	r3/r4	r3				

↳ Here we are getting reduce/reduce conflict

So grammar is not SLR(1)

Q4:

	First( )	Follow( )
$S \to ACB \mid CbB \mid Ba$	{d, g, h, b, a, ε}	{$}
$A \to da \mid BC$	{d, g, h, ε}	{h, g, $}
$B \to g \mid \epsilon$	{g, ε}	{$, a, h, g}
$C \to h \mid \epsilon$	{h, ε}	{g, $, b, h}

Q5 of Q7. E→E+T|T
T→T*F|F
F→(E)|id

Construct LR(0) parsing table for the grammar
Examine the given grammar is SLR(1) or not.

SLR(1) Parsing Table

State	ACTION id	+	*	(	)	$	GOTO E	T	F
0	s5			s4			1	2	3
1		s6				Acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Productions with Number

1. $E \to E+T$
2. $E \to T$
3. $T \to T*F$
4. $T \to F$
5. $F \to (E)$
6. $F \to id$

⇒ Since there is no conflict

⇒ Grammar is SLR(1).

FOLLOW(E) = { $, +,) }

Si: means shift and stack state i

rj: reduce by the production numbered j.

blank entries refer to errors.

*: To construct the LR(0) parsing Table.

In the action part under each terminal we need to place the reduce move.

for the states having productions with • (dot) at the end of rhs of a production

eg: Consider state 11 contains production $F \to (E)\bullet$ Number: 5

State 9 " " $E \to E+T\bullet$ number: 1

State	id	+	*	(	)	$	E	T	F
9	r1	r1	s7/r1	r1	r1	r1	—	—	—
11	r5	r5	r5	r5	r5	r5	—	—	—
3	r6 r4	r6 r4	r6 r4	r6 r4	r6 r4	r6 r4			

Complete LR(0) parsing table by your own.

Q6: Operator Precedence Parser:

- A grammar that is used to define mathematical operators is called an operator grammar with restrictions.
  1. Operator grammar can not have null productions of the form $A \to \epsilon$
  2. In the right hand side of the production rules; Two non terminals can not be adjacent (side by side)

eg: $E \to EAE$, $A \to + \mid *$ → This is not an operator grammar since in the production $E \to EAE$; The non terminals E & A are adjacent to each other.

⇓ Can be converted to operator grammar as

$E \to E+E \mid E*E$

eg: $S \to SAS \mid a$, $A \to bSb \mid b$ → Not an operator grammar as in $S \to SAS$ S & A are side by side (adjacent to each other)

⇓ operator grammar

$S \to SbSbS \mid SbS \mid a$.

- An operator precedence parser is a bottom up parser used to parse operator grammars.
- Operator precedence parser is capable of parsing some of the ambiguous grammar.
  - $a \doteq b$ ⇒ a is having higher precedence than b
  - $a \lessdot b$ ⇒ a is having lower precedence than b
- Uses operator relation table.
- Rows & columns have terminals

eg: $E \to E+E \mid E*E \mid id$

Rules: id has highest precedence

table	id	+	*	$
id	-	⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	-

table[i][j] = ⋗ if precedence(i) > precedence(j)

table[i][j] = ⋖ if precedence(i) < precedence(j)

*. if any operator is left associative

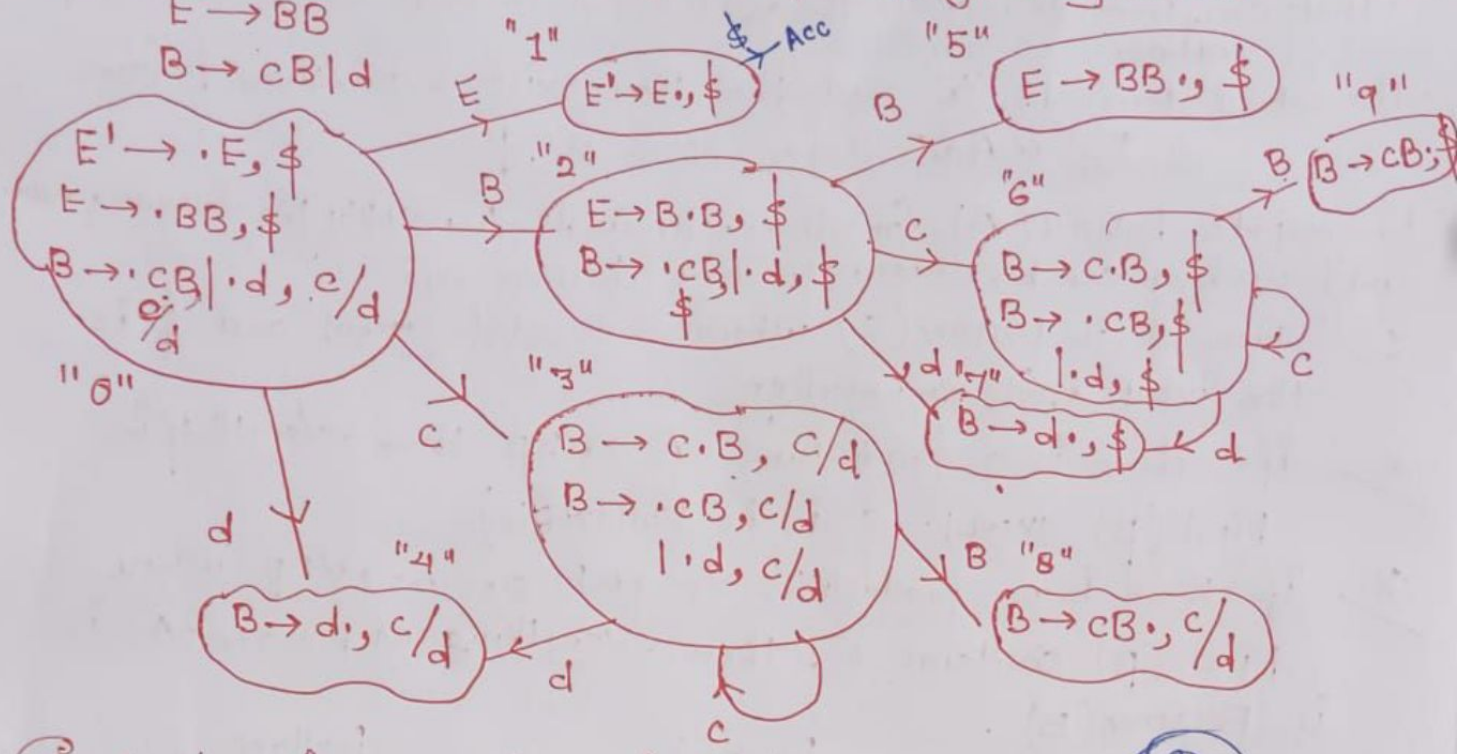
eg: consider + is left-associative

table[+][+] = ⋗ since left-side + is having higher precedence than right side +

Eg: Construct the CLR(1) parser for the given grammar

S→AA
A→aA|b

E → BB
B → cB | d



Construction of CLR(1) parsing table.

(29)

State	ACTION c	ACTION d	ACTION $	GOTO E	GOTO B
0	S3	S4		1	2
1			Accept		
2	S6	S7			5
3	S3	S4			8
4	r3	r3			
5			r1		
6	S6	S7			9
7			r3		
8	r2	r2			
9			r2		

Q10: $S \to AB \mid d$
$A \to aA \mid b$
$B \to bB \mid c$

First(B) = {b, c}
First(A) = {a, b}
First(S) = {a, b, d}

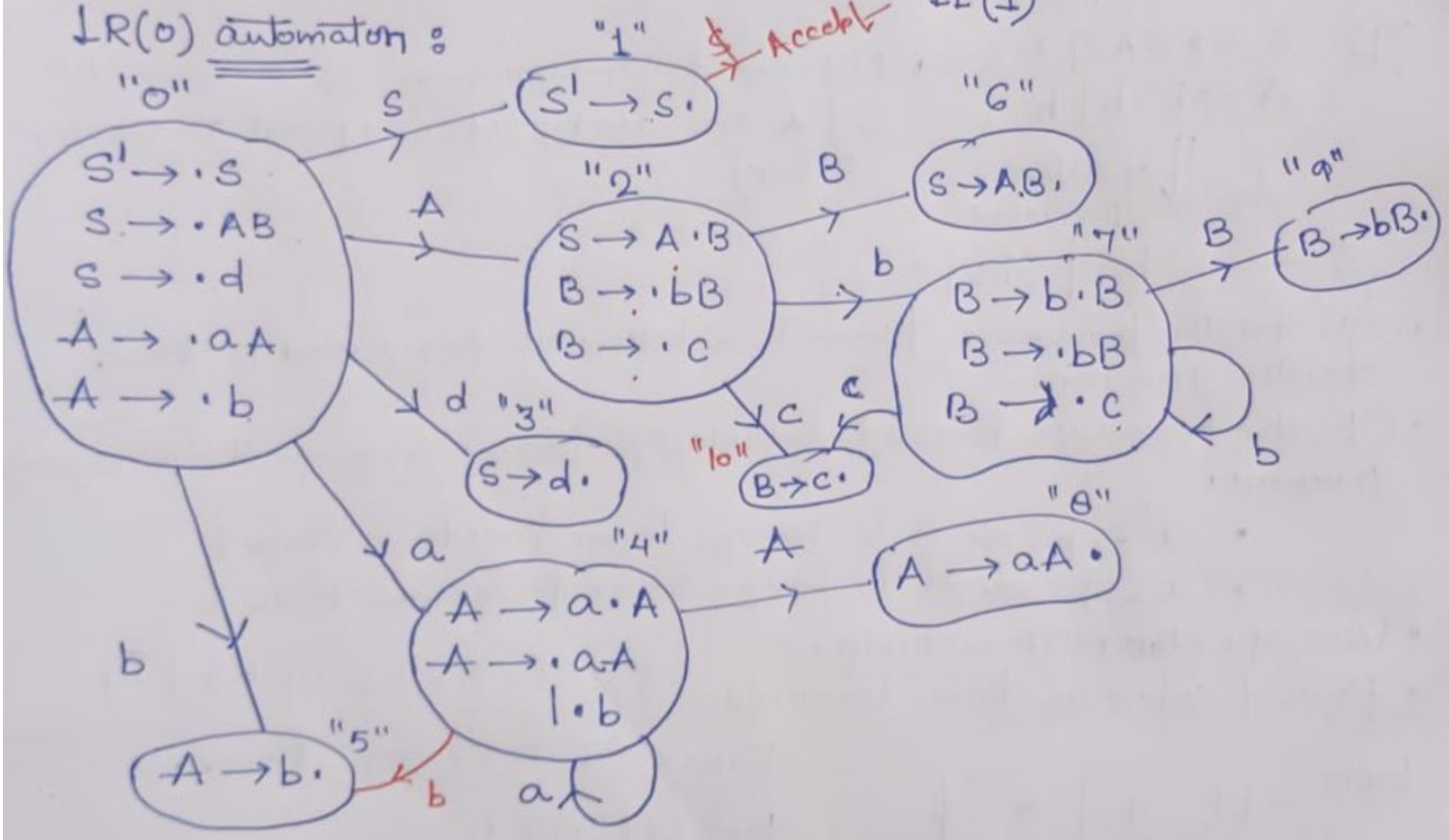
Follow(S) = {$}
Follow(A) = First(B) ∪ Follow(A) = {b, c}
Follow(B) = Follow(S) ∪ Follow(B) = {$}

LL(1) Parsing Table

	a	b	c	d	$
S	S→AB	S→AB		S→d	
A	A→aA	A→b			
B		B→bB	b→c		

Each entry has single production ⇒ Grammar is LL(1)

LR(0) automaton :



Consider the states having final items
State no. (5) only one production with final item.
(1) (3) (6) → no shift.
(9) (8) (10)

No conflict – Grammar is LR(0)

Q12: Explain translation rule for expression 2+5*4.

Grammar to generate given string

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

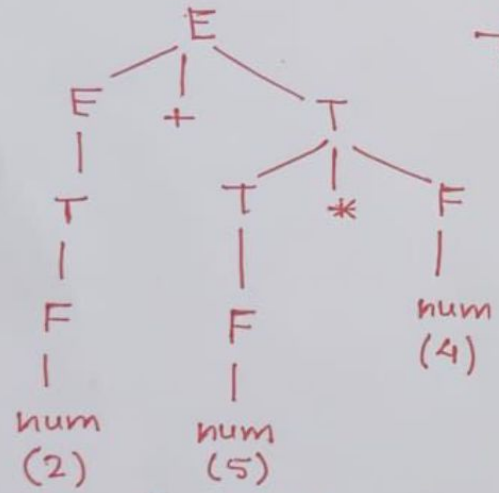
$F \rightarrow num$

Translation Rules: By giving the attribute equations and order of evaluation.

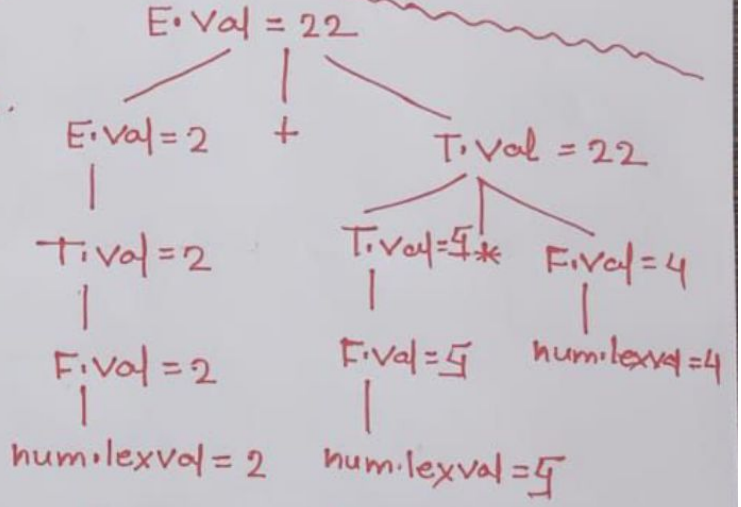
$E \rightarrow E+T$	{ E.value = E.value + T.value }
$E \rightarrow T$	{ E.value = T.value }
$T \rightarrow T*F$	{ T.value = T.value * F.value }
$T \rightarrow F$	{ T.value = F.value }
$F \rightarrow num$	{ F.value = num.lexicalvalue }

~~Traversal~~

Parse Tree for the string 2+5*4



Annotated Parse tree.



Evaluation with bottom up parsing stack: Use of value stack.

$E \rightarrow E+T$ { stack[top-2].val = stack[top-2].val + stack[top].val; top = top-2; }

$E \rightarrow T$

$T \rightarrow T*F$ { stack[top-2].val = stack[top-2].val * stack[top].val; top = top-2; }

$T \rightarrow F$

$F \rightarrow (E)$ { stack[top-2].val = stack[top-1].val; top = top-2; }

$F \rightarrow num$