Question 1:-What is the mean by viable prefixes?

Answer

Definition :- The prefixes of right sentential form that can appear on the stack of a shift-reduce parser are called viable prefixes.

or

α is a viable prefix of the grammar if there is ω such that $\alpha\omega$ is a right sentinel form. The set of prefixes of right sentinel form that can appear on the stack of a shift-reduce parser are called viable prefixes.

Ex: Let $S \rightarrow X_1 X_2 X_3 X_4$

$A \rightarrow X_1 X_2$

Let $\omega = X_1 X_2 X_3$

Stack	Input
\$	$X_1 X_2 X_3$
\$ X_1	$X_2 X_3$
\$ $X_1 X_2$	X_3
\$ A	X_3
\$ A X_3	

as we see $X_1 X_2 X_3$ will never appear on the stack. So it is not a viable prefix.

Let $\omega = X_1 X_2$

Stack	Input
\$	$X_1 X_2$
\$ X_1	X_2
\$ $X_1 X_2$	
\$ A	

$\omega = X_1 X_2$ is a viable prefix.

Question 2:What are the problems with top down parsing? Write the algorithms for first and follow?

Answer

Problem with Top-Down Parsing :-

1. Not applicable for left recursive production.
2. Left factoring is another issue associated with Top-Down parsing.
3. Back-Tracking

Algorithm to compute First :-

1. If X is a terminal, then First(X) = X
2. If $X \to \epsilon$ is a production in grammar then add ϵ to First(X)
3. If X is a non-terminal and $X \to Y_1 Y_2 \cdots Y_n$ then
   a. First(X) = First($Y_1 Y_2 \cdots Y_n$) if First(Y_1) does not contain null ϵ
   b. else First(X) = First(Y_1) $- \epsilon \cup$ First($Y_2 Y_3 \cdots Y_n$)
4. If First(Y_1), First(Y_2) — First(Y_n) contains Null (ϵ) then

   First(X) = First(Y_1) $\cup$ First(Y_2) $\cup \cdots \cup$ First(Y_n)

Algorithm to compute Follow :-

1. If S is the start symbol then add $ in Follow(S)
2. If there is a production $A \to aBb$
   a. Follow(B) = First(b) if first(b) does not contain Null
   b. Follow(B) = First(b) $- \epsilon \cup$ Follow(A)
3. If there is a production $A \to aB$, then everything in Follow(A) is in Follow(B)

Question 3:-Perform Shift reduce parsing for the given input strings using the grammar

$S \rightarrow (L) \mid a$

$L \rightarrow L , S \mid S$

I. (a,(a,a))

II. (a,a)

Answer

(i) Parsing of (a, (a, a))

Stack	Input Buffer	Parsing Action
$	(a, (a, a)) $	shift
$ (	a, (a, a)) $	shift
$ (a	, (a, a)) $	Reduce (S→a)
$ (S	, (a, a)) $	Reduce (L→S)
$ (L	, (a, a)) $	shift
$ (L,	(a, a)) $	shift
$ (L, (	a, a)) $	shift
$ (L, (a	, a)) $	Reduce (S→a)
$ (L, (S	, a)) $	Reduce (L→S)
$ (L, (L	, a)) $	shift
$ (L, (L,	a)) $	shift
$ (L, (L, a	)) $	Reduce (S→a)
$ (L, (L, S	)) $	Reduce (L→L,S)
$ (L, (L	)) $	shift
$ (L, (L)	) $	Reduce (S→(L))
$ (L, S	) $	Reduce (L→L,S)
$ (L	) $	shift
$ (L)	$	Reduce S→(L)
$ S	$	Accept

(ii) Similarly do 2nd part- (a, a)

Question 4:-Construct LR(0) parsing table for the following grammar

$S \rightarrow c\ B \mid c\ c\ A$

$A \rightarrow c\ A \mid a$

$B \rightarrow c\ c\ B \mid b$

Answer

Augment the grammar

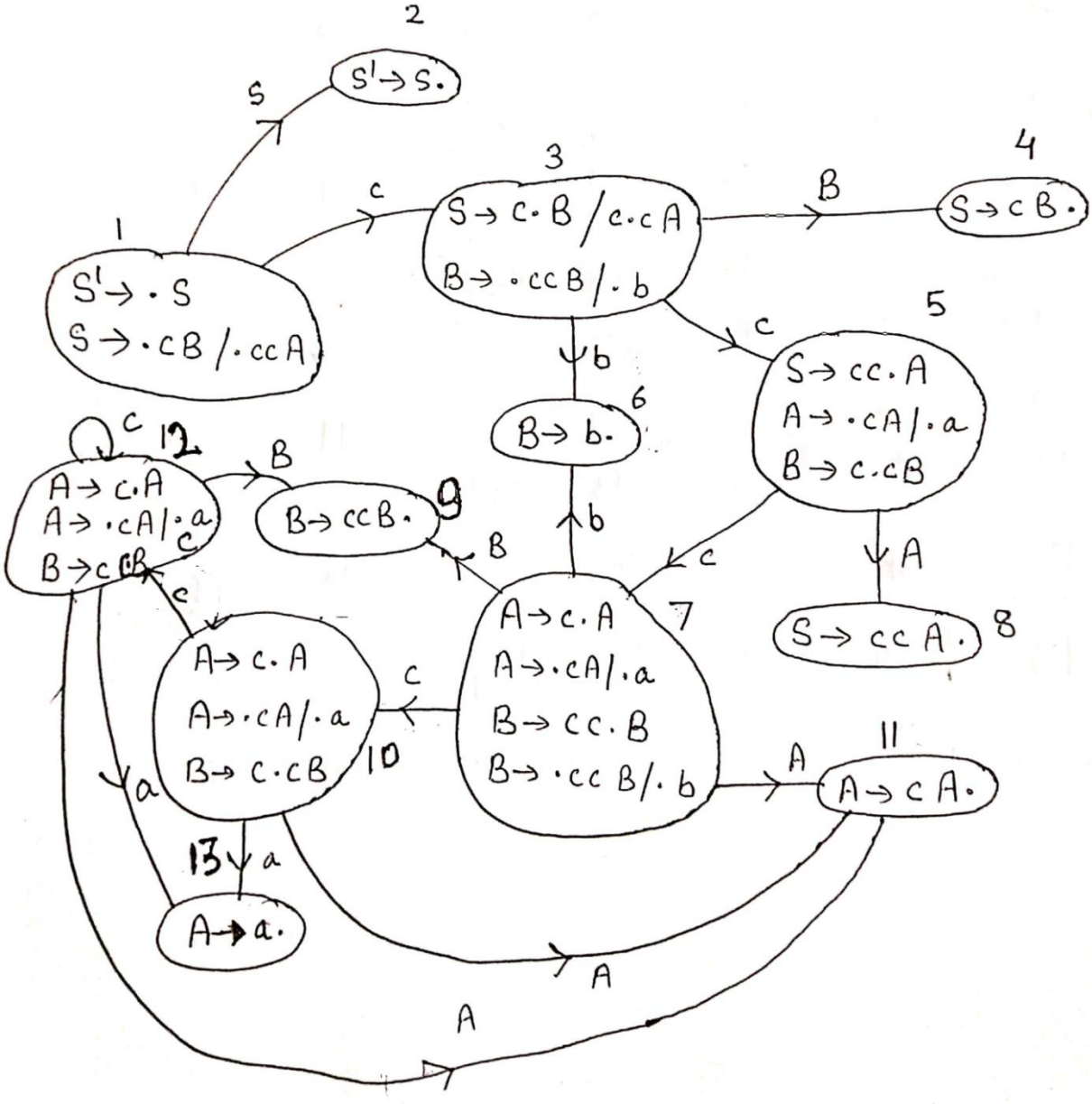
$S' \rightarrow S$

$S \rightarrow cB \ (1) \mid cc A \ (2)$

$A \rightarrow cA \ (3) \mid a \ (4)$

$B \rightarrow cc B \ (5) \mid b \ (6)$

Cannonical collection of LR(0) item is :-



LR(0) Parsing table

State	Action				Goto		
	a	b	c	$	S	A	B
1			S3		2		
2				Accept			
3		S6	S5				4
4	r_1	r_1	r_1	r_1			
5			S7			8	
6	r_6	r_6	r_6	r_6			
7		S6	S10			11	9
8	r_2	r_2	r_2	r_2			
9	r_5	r_5	r_5	r_5			
10	S13		S12			11	
11	r_3	r_3	r_3	r_3			
12	S13		S12			11	9
13	r_4	r_4	r_4	r_4			

Note:-This table is LR(0) table not SLR(1) table . In the question there is some doubt about LR(0) or SLR(1).

Note :- for SLR(1) table construction only reduce entries is change.

Question 5:-Eliminate the left recursion from the following grammar

S→AB

A→ BS | b

B→SA | a

Answer

$S \rightarrow AB$

$A \rightarrow BS \mid b$

$B \rightarrow SA \mid a$

Substitute $S \rightarrow AB$ in the production

$B \rightarrow SA \mid a$

$\Rightarrow B \rightarrow ABA \mid a$ —— (4)

Substitute $A \rightarrow BS \mid b$ in the production no. (4)

then $B \rightarrow BSBA \mid bBA \mid a$ —— (5)

Now eliminate the left recursion from production (5)

$B \rightarrow bBAB' \mid aB'$

$B' \rightarrow SBAB' \mid \epsilon$

Now the final result is

$S \rightarrow AB$

$A \rightarrow BS \mid b$

$B \rightarrow aB' \mid bBAB'$

$B' \rightarrow SBAB' \mid \epsilon$

Question 6:-Explain non recursive predictive parsing. Consider the following grammar and construct the predictive parsing table

E→TE'

E'→+TE'|€

T→FT'

T'→*FT'|€

F→* F|a|b

Answer

Non – recursive predictive parsing :—

The predictive parsing is a special form of recursive discent parsing, where no backtracking is required, so this can predict which production to use to replace the input string. Non-recursive predictive parsing or table-driven is also known as LL(1) parser.

	First	Follow
$E \to TE'$	{*, a, b}	{$}
$E' \to +TE'/\epsilon$	{+, ϵ}	{$}
$T \to FT'$	{*, a, b}	{+, $}
$T' \to *FT'/\epsilon$	{*, ϵ}	{+, $}
$F \to *F \mid a \mid b$	{*, a, b}	{*, +, $}

LL(1) Table

	*	a	b	+	$
E	$E \to TE'$	$E' \to TE'$	$E \to TE'$		
E'				$E' \to +TE'$	$E' \to \epsilon$
T	$T \to FT'$	$T \to FT'$	$T \to FT'$		
T'	$T' \to *FT'$			$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to *F$	$F \to a$	$F \to b$		

Question 7:-Give operator-precedence parsing algorithm. Consider the following grammar and build up operator precedence table. Also parse the input string

(id+(id*id))

E→E+T | T
T→T*F | F
F → (E) | id

Answer

Algorithm:-

Set i/p pointer to the first symbol of ω $

repeat forever

if $ is on top of the stack and i/p points to $

return

else

let a be the topmost terminal symbol on the stack and let b be the symbol pointed to by i/p pointer

if a<b or a=b then

push b onto the stack advance i/p pointer to the next i/p symbol.

else if a >b then

pop the stack until the top stack terminal is related by < to the terminal most recently popped.

else

error

operator precedence parsing table

	+	*	(	)	id	$
+	>	<	<	>	<	>
*	>	>	<	>	<	>
(	<	<	<	=	<	
)	>	>		>		>
id	>	>		>		>
$	<	<	<		<	

Stack	Input	O/P
$	(id+(id*id))$	-
$(	id+(id*id))$	Shift
$(id	+(id*id))$	Shift
$(F	+(id*id))$	Reduce (F→id)
$(T+	(id*id))$	Shift
$(E+(	id*id))$	Shift
$(E+(id	*id))$	Reduce (F→id)
$(E+(F	*id))$	Shift
$(E+(T*	id))$	Shift
$(E+(E*id	))$	Reduce (F→id)
$(E+(E*F)	)$	Shift
$(E+(E*T)	)$	Shift
$(E+(E)	)$	Reduce
$(E+F)	$	Shift
$(E+T)	$	Reduce
$(E)	$	Reduce
$F	$	Reduce
$T	$	Reduce
$E	$	Reduce

Question 8:-For the grammar

S→aAd | bBd | aBe | bAe

A→f

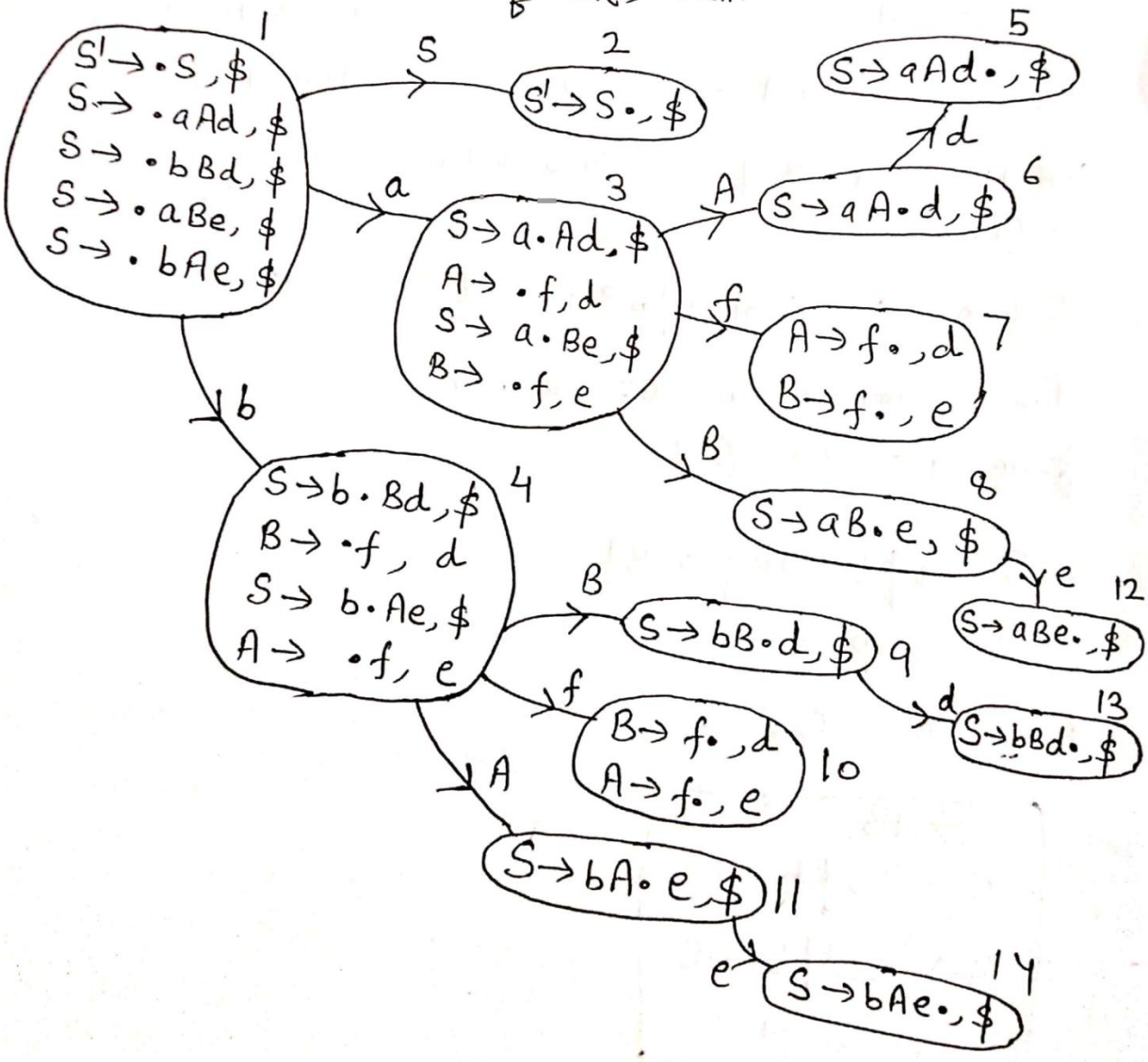
B→f

Construct LR(1) Parsing table . Also draw the LALR table from the derived LR(1) parsing table.

Augmented grammar is :-

S'→ S
S→ a Ad | bBd | aBe | bAe
A→ f
B→ f

Canonical Collection of LR(1) item



LR(1) Parsing table

State	Action						Goto		
	a	b	d	e	f	$	S	A	B
1	S3	S4					2		
2						Accept			
3					S7			6	8
4					S10			11	9
5						r_1			
6			S5						
7			r_5		r_6				
8				S12					
9			S13						
10			r_6		r_5				
11				S14					
12						r_3			
13						r_2			
14						r_4			

Because in the canonical collection of LR(1) item there is no state who's LR(0) item is similar so LALR Parsing table is similar to LR(1) table.

Question:-9 Consider the following grammar

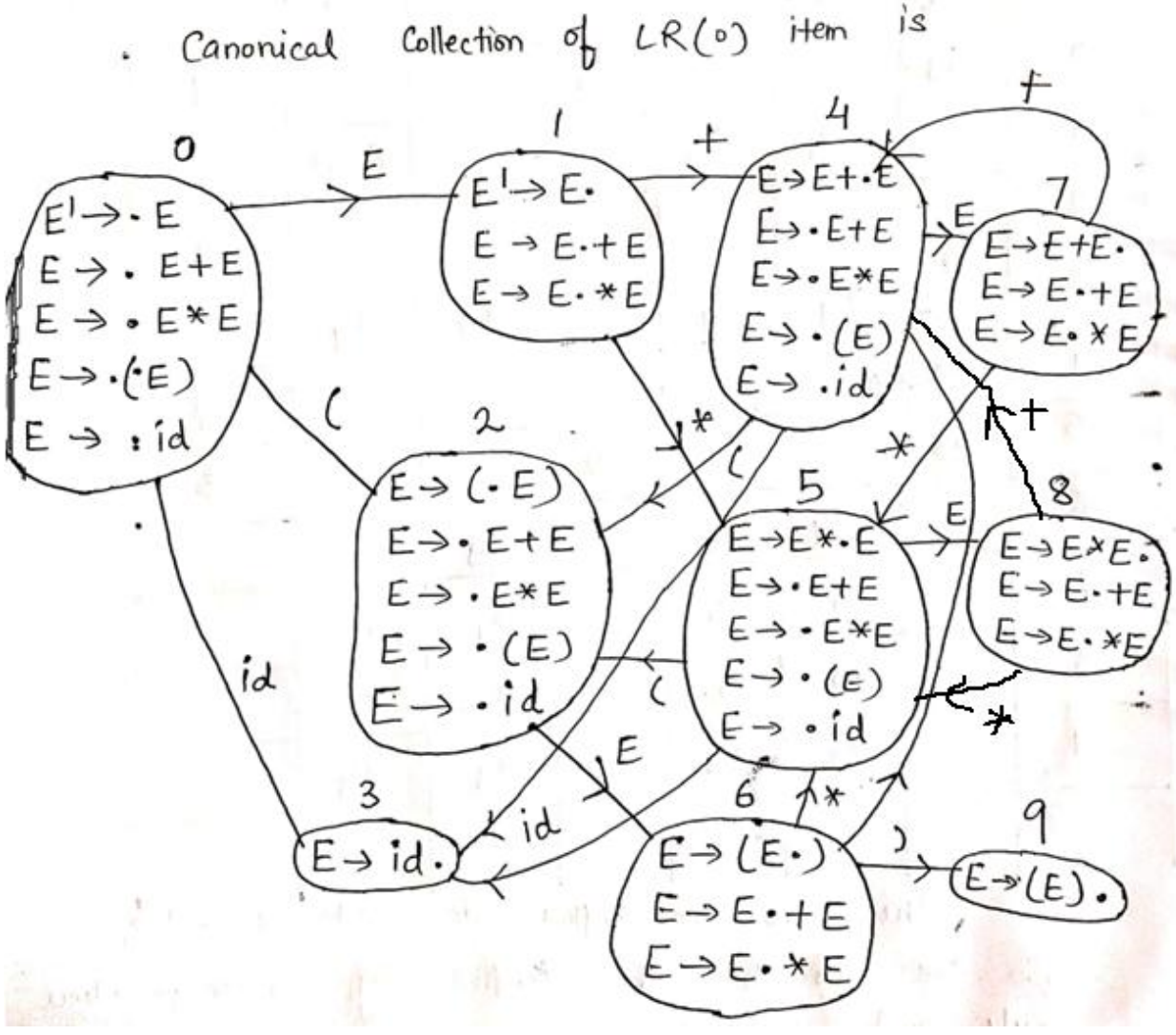
E→E+E

E→E*E

E→(E) | id

Construct SLR parsing table and suggest your final parsing table.

Answer



SLR Parsing table

State	Action						Goto
	id	+	*	(	)	$	E
0	S_3			S_2			1
1		S_4	S_5			Acept	
2							6
3		r_4	r_4		r_4	r_4	
4	S_3			S_2			7
5	S_3			S_2			8
6		S_4	S_5		S_9		
7		r_1	S_5		r_1	r_1	
8		r_2	S_5		r_2	r_2	
9		r_3	r_3		r_3	r_3	

There is two conflict in state 7 and 8

So we resolve the conflict by operator precedence rule and construct above table.

Question 10:-State the problem associated with top down parsing?

Answer

Refer to answer of question 2

Question 11:-What is the role of left recursion.

Answer

Left Recursion :- A production of a grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS. A grammar containing a production having left recursion is called as Left Recursive Grammar.

Ex: $A \rightarrow A\alpha$

left recursion is used in the production when we need to make left associative of a grammar

Drawback :- left recursion is not immlemented through the funn because of infinte looping problem.

Question 12:-Construct an SLR(1) parsing table for the following grammar

S→ A)

A→ A , P) (P , P

P→{num,num}

Answer

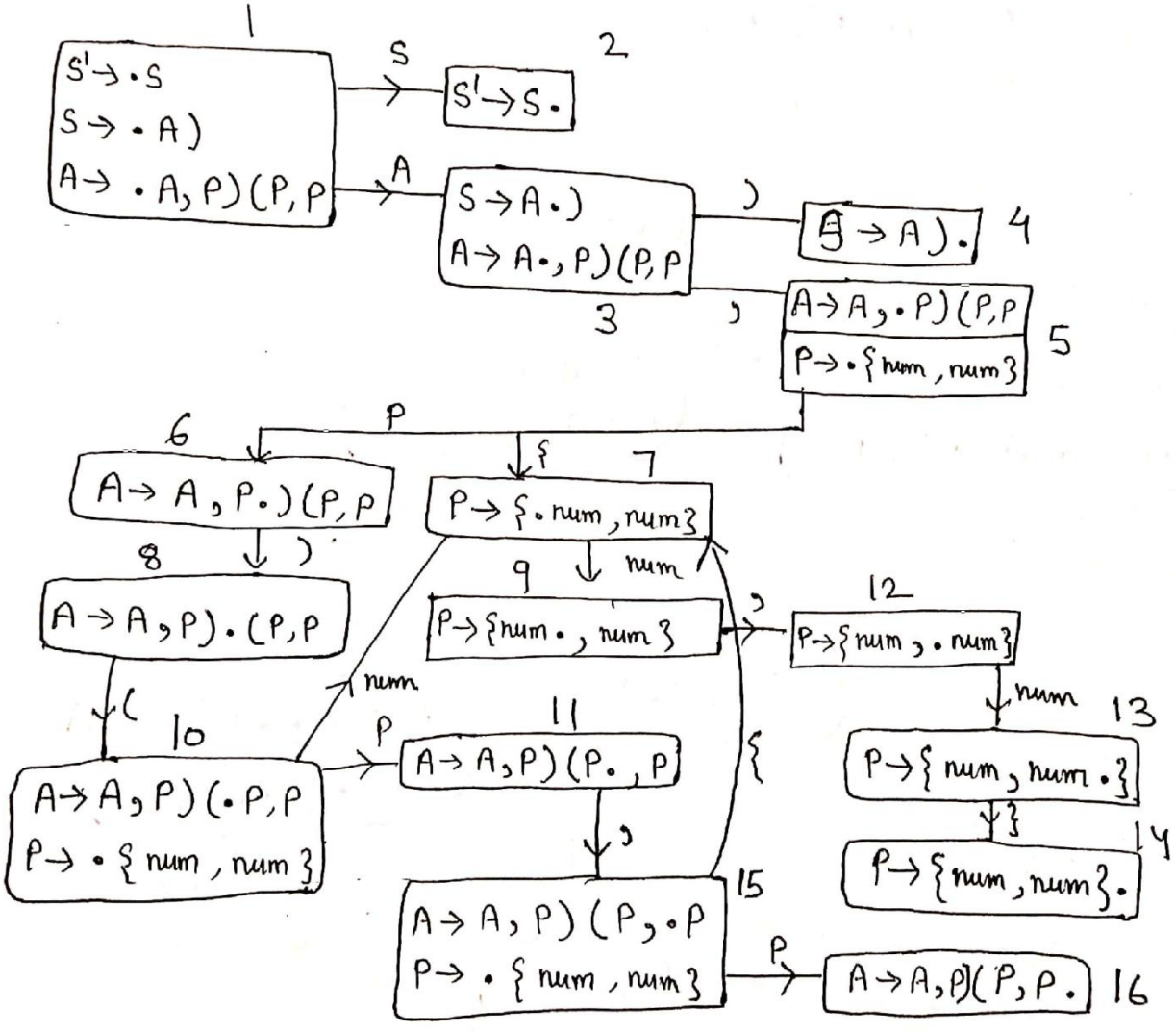
Augmented grammar :-

S' → S

S → A) ——— (1)

A → A , P) (P , P ——— (2)

P → { num , num } ——— (3)



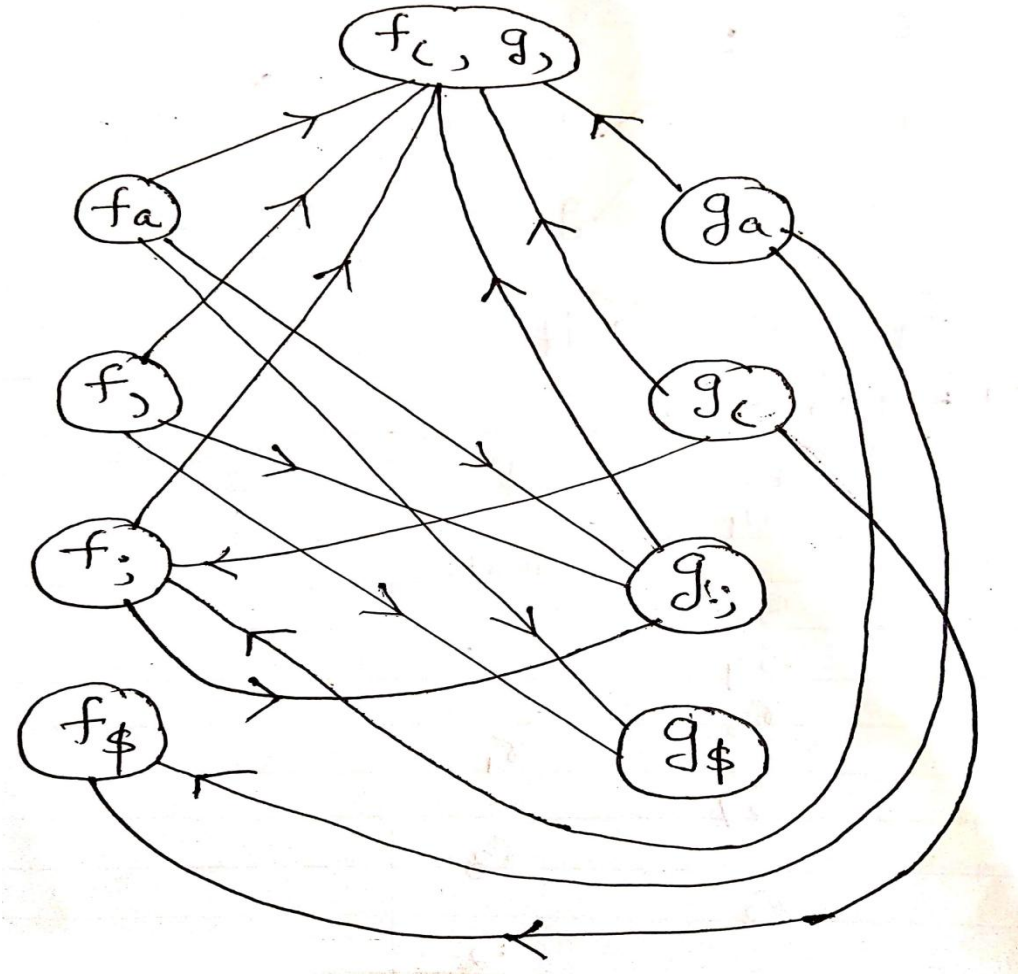
SLR(1) Parsing table

State	Action							Goto		
	(	)	{	}	,	num	$	S	A	P
1								2	3	
2							Acc			
3		S_4			S_5					
4							r_1			
5			S_7							6
6		S_8								
7						S_9				
8	S_{10}									
9					S_{12}					
10						S_7				11
11					S_{15}					
12						S_{13}				
13				S_{14}						
14		r_3			r_3					
15			S_7							16
16		r_2			r_2					

Question 13:-Consider the following operator precedence matrix draw precedence graph and compute the precedence function

	a	(	)	;	$
a			>	>	>
(	<	<	=	<	
)			>	>	>
;	<	<	>	>	
$	<	<			

Answer



	a	(	)	;	$
f	2	0	2	2	0
g	3	3	0	1	0

Question 14:-Show that the following grammar

S→Aa | bAc | Bc | bBa

A→d

B→d

is LR(1) but not LALR(1)

Answer

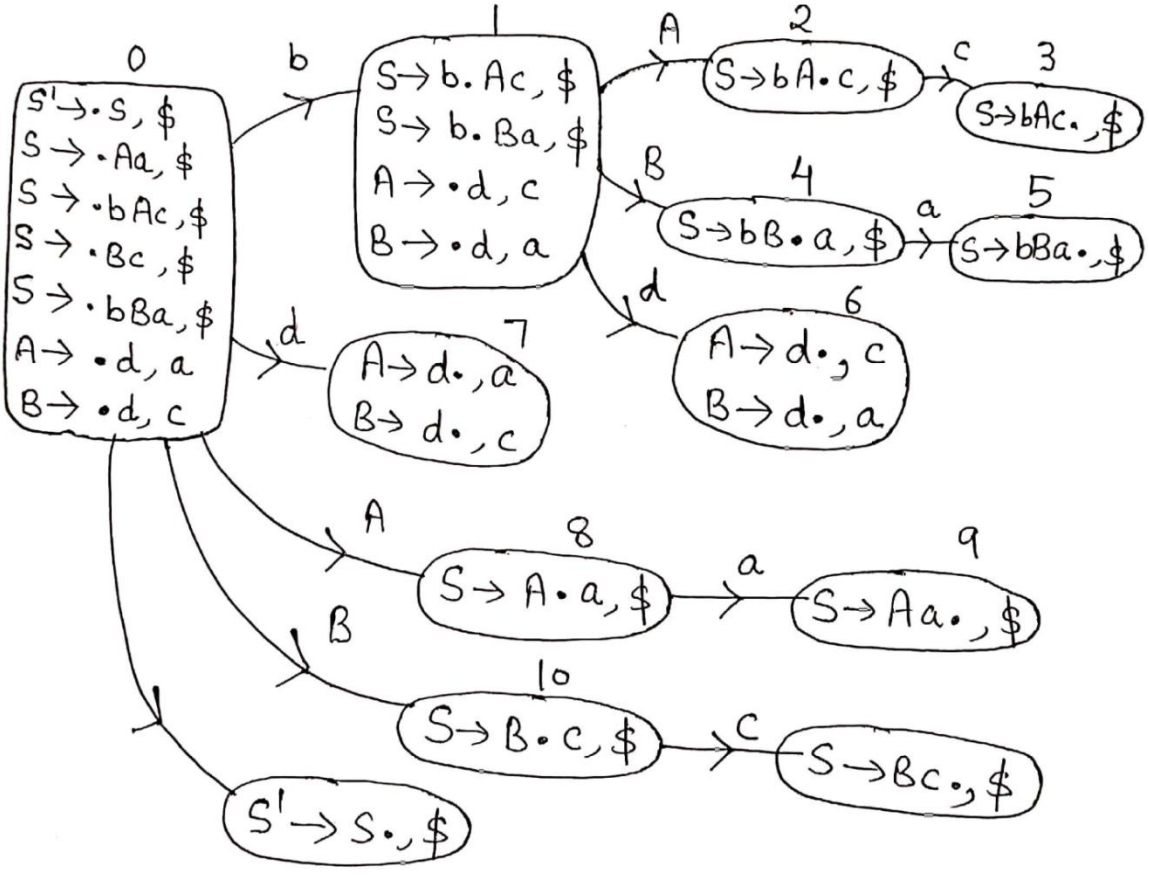
Augmented grammar is.

S'→S

S→ Aa | bAc | Bc | bBa

A→ d

B→ d



No conflicts in any state, so grammar is LR(1)

But grammar is not LALR(1) because on merging state 6 and 7 we get reduce-reduce conflict on lookahead a, c

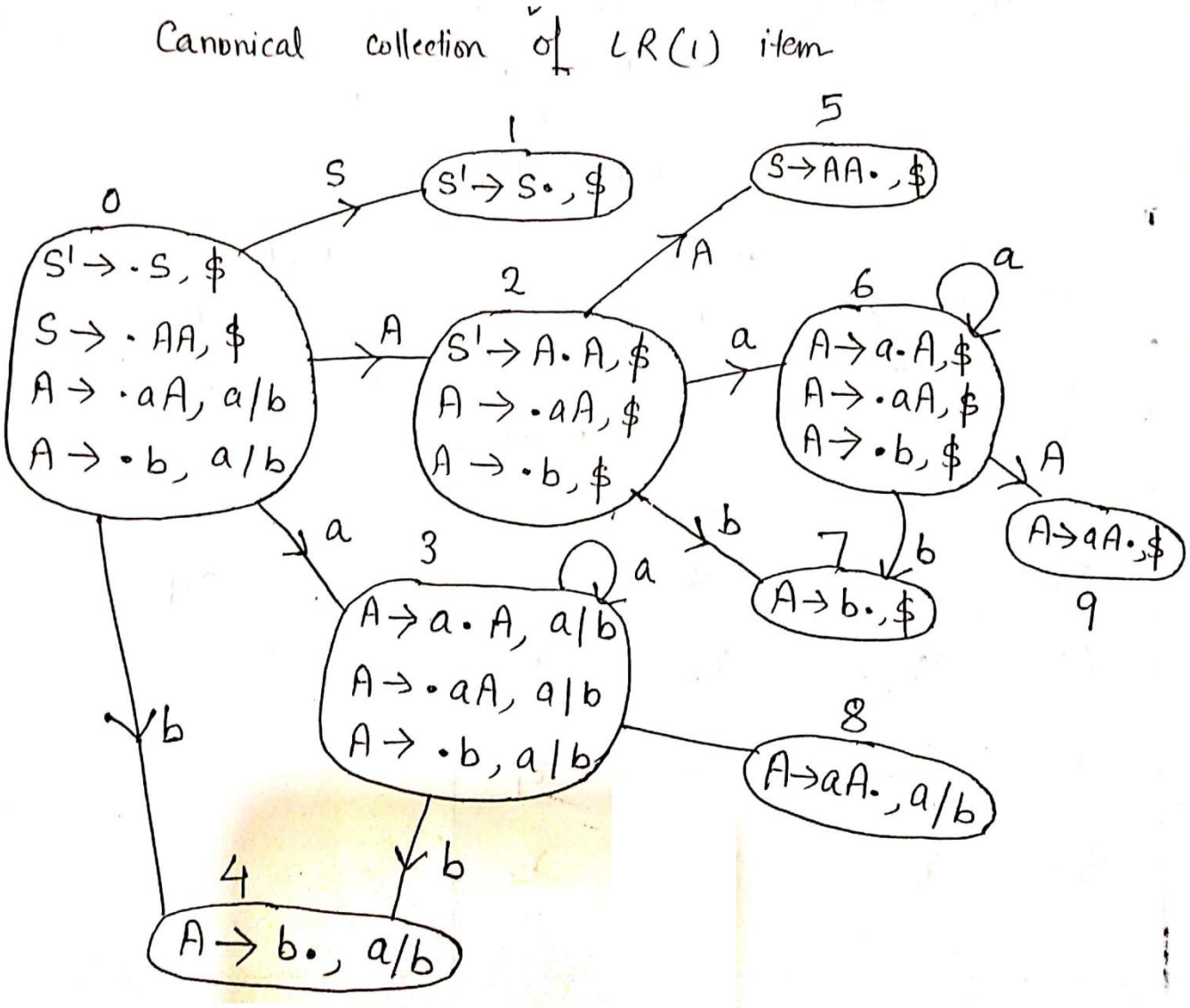
Question 15:-Construct LALR parsing table for the following grammar.

S→AA

A→aA

A→b

Answer



LALR(1) Table

State	Action			Goto	
	a	b	$	S	A
0	S_{36}	S_{47}		1	2
1			Accept		
2	S_{36}	S_{47}			5
36	S_{36}	S_{47}			89
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

Question 16:-Check whether left recursion exists for the following grammar or not and if exist eliminate it.

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid \epsilon$

Step (1) :-
First eliminate left recursion from $S \rightarrow Aa / b$
This is already free from left recursion

Step (2) :-
Substituting the productions of S in $A \rightarrow Sd$, we get the following grammar

$S \rightarrow Aa / b$
$A \rightarrow Ac / Aad / bd / \epsilon$

Step (3) :- Now eliminating left recursion from the production of A, we get the following grammar

$S \rightarrow Aa / b$
$A \rightarrow bdA' / A'$
$A' \rightarrow cA' / adA' / \epsilon$

This is the final grammar after eliminating left recursion.

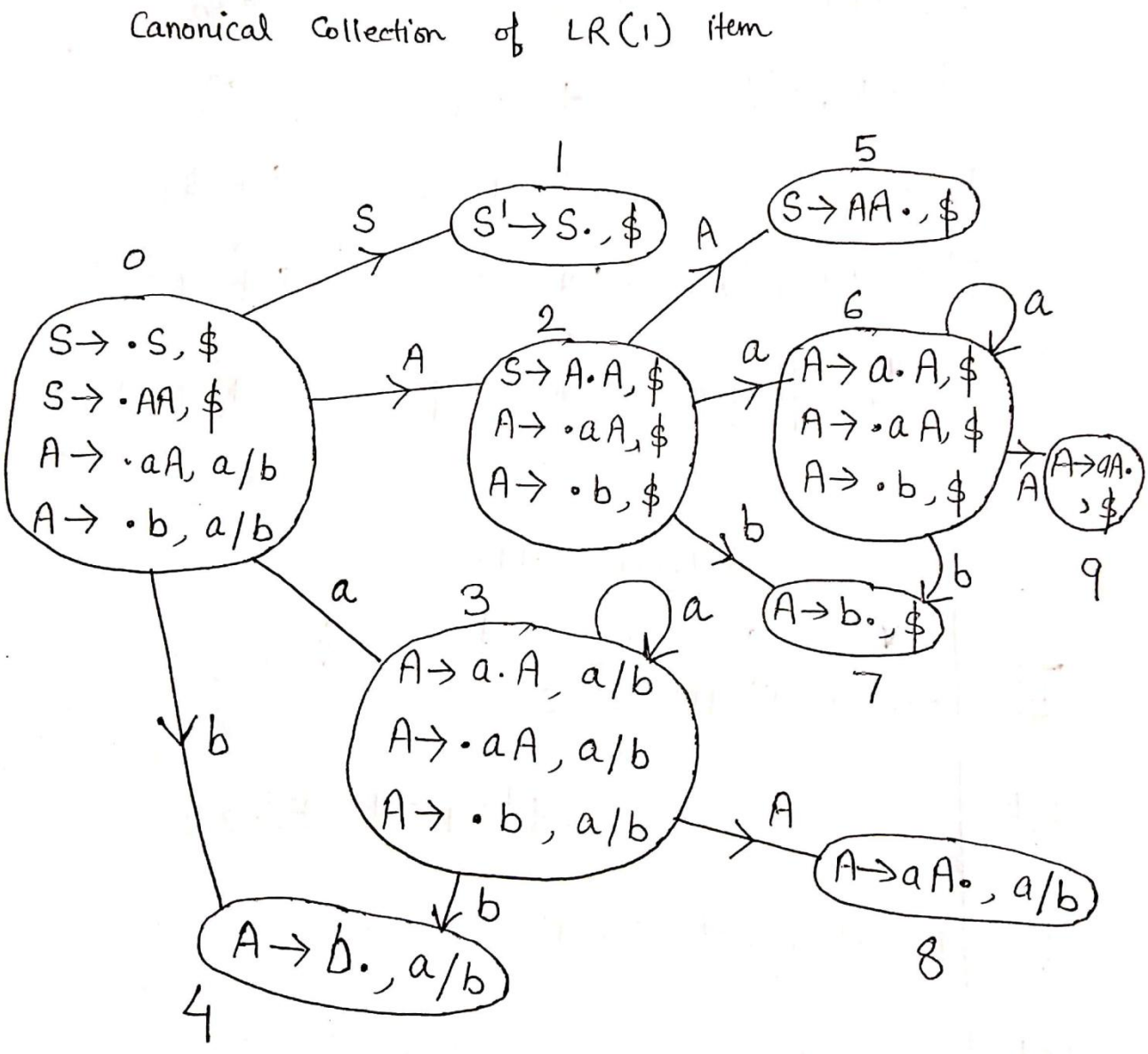
Question 17:-Construct CLR parsing table for the following grammar.

S→AA

A→aA

A→b

Answer



CLR (1) Table

State	Action			Goto	
	a	b	$	S	A
0	S_3	S_4		1	2
1			Accept		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

Prepared By:

Prabhat Shukla

UCER, CS/IT Department

Prayagraj

Note: For any discrepancy contact

Whatsapp no.: - 8957419537

Email Id:- shuklaprabhat2k7@gmail.com