Q.1. Construct a predictive parsing table for the given grammar. (or)
check whether the given grammar is LL(1) or not.

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

→ step 1:- eliminating left recursion.

$E \rightarrow$ ~~E+T~~ TE'

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

step 2:- left factoring.
no left factoring.

step 3:- first & follow functions.

first (E) = first (T) = first (F) = { (, id }

first (E') = { +, ε }

first (T') = { *, ε }

follow (E) = { $,) }

follow (T) = { +, $,) }

follow (E') = { $,) }

follow (T') = { $, +,) }

follow (F) = { *, $, +,) }

					-1	a.	ε

step 4:- predictive parsing table.

	+	*	(	)	id	$
E			$E \to TE'$		$E \to TE'$	
E'	$E' \to +TE'$			$E' \to \varepsilon$		$E' \to \varepsilon$
T			$T \to FT'$		$T \to FT'$	
T'	$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$		$T' \to \varepsilon$
F			$F \to (E)$		$F \to id$	

As there are no multiple entries in table, hence the given grammar is LL(1).

~~step 5:- Moves made by pt~~

onstruct the first & follow & predictive arse table for the grammar.

$S \to AC\$$

$C \to c \mid \epsilon$

$A \to aBCd \mid BQ \mid \varepsilon$

$B \to bB \mid d$

$Q \to q$.   & input string is abdcdc$.

irst :-

irst $(Q) = \{q\}$

irst $(B) = \{b, d\}$

irst $(A) = \{a, b, d, \varepsilon\}$

irst $(C) = \{c, \varepsilon\}$

irst $(S) = \{a, b, d, c, \varepsilon\}$

ollow :-

ollow $(S) = \{\#\}$

ollow $(A) = \{c, \$\}$

ollow $(B) = \{c, d, q\}$

ollow $(Q) = \{c, \$\}$

	a	b	c	d	q	ε	$	#
S	S→Ac$	S→Ac$	~~S→Ac$~~	S→Ac$		S→Ac$		
A	A→ aBcd	A→ ~~b~~BQ		A→BQ		A→ε		
C			C→c			C→ε		
B		B→bB		B→d				
Q					Q→q			

Moves mode by predictive parser ~~can be~~ on the input abdcdc$ is :-

stack	input	output
# S	abdcdc$#	S→ Ac$
# ~~S~~ $ c A	abdcdc$#	A→ aBεd
# $ c d ε B a	abdcdc $#	
# $ C d ε B	bdcdc $#	B → b B
# $ C d C B b	b dc dc $#	
# $ C d ε B	dcdc $#	B → d
# $ C d ε d	dc dc $#	
# $ C d ε	c dc $#	c → c
# $ c d c	cdc $#	
# $ C d	dc $#	
# $ C	c $#	C → c
# $ c	c $#	
# $	$#	
#	#	accepted.

Q. 3. LR (0) parsing.

$S \rightarrow AA$

$A \rightarrow aA \mid b.$

→ step 1 :-

$S \rightarrow AA$ - ①

$A \rightarrow aA$ - ②

$A \rightarrow b$ - ③

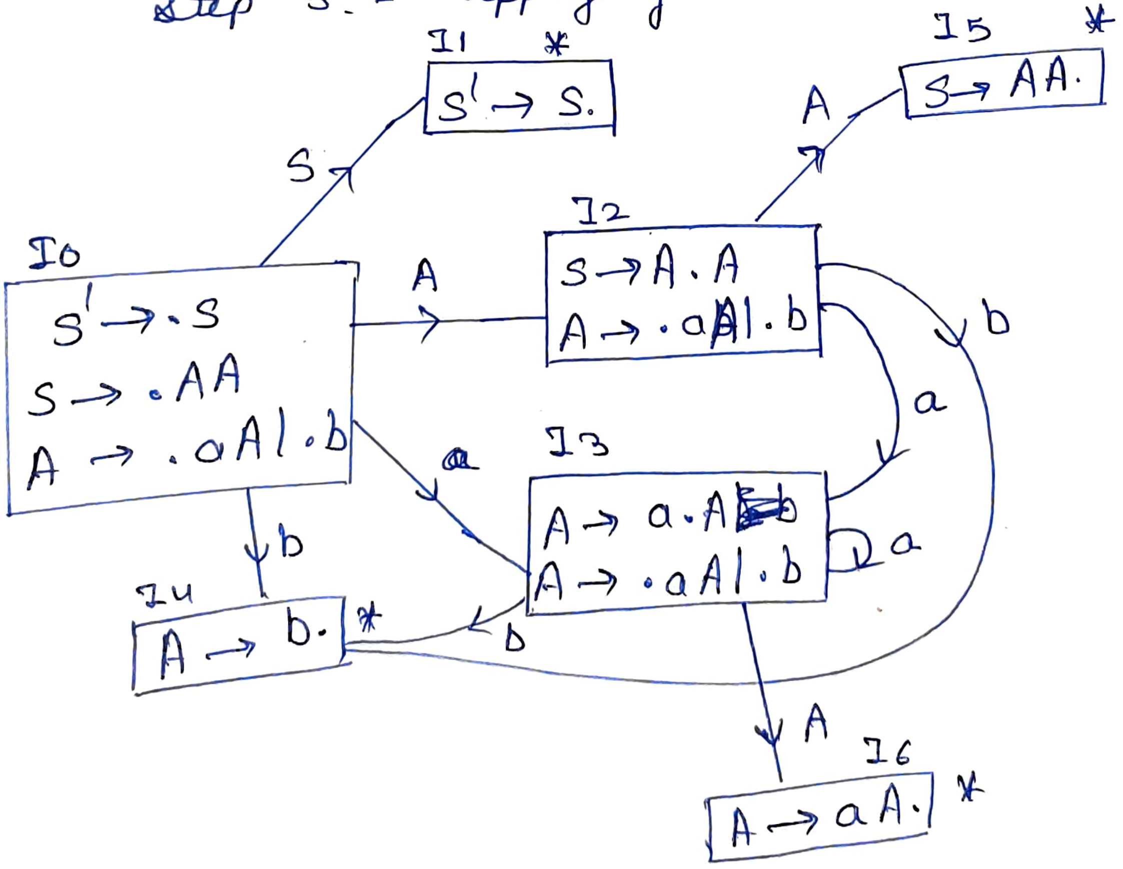
step 2 :- augmented grammar.

$S' \rightarrow .S$

$S \rightarrow .AA$

$A \rightarrow .aA \mid .b$

step 3 :- applying closure.



step 4 :- LR(0) parsing table.

	action			goto	
	a	b	$	A	S
0	s_3	s_4		2	1
1			accept		
2	s_3	s_4		5	
3	s_3	s_4		6	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

SLR(1) parsing.

$S \rightarrow AA$

$A \rightarrow aA \mid b$.

step 1:-

$S \rightarrow AA$ - ①

$A \rightarrow aA$ - ②

$A \rightarrow b$ - ③

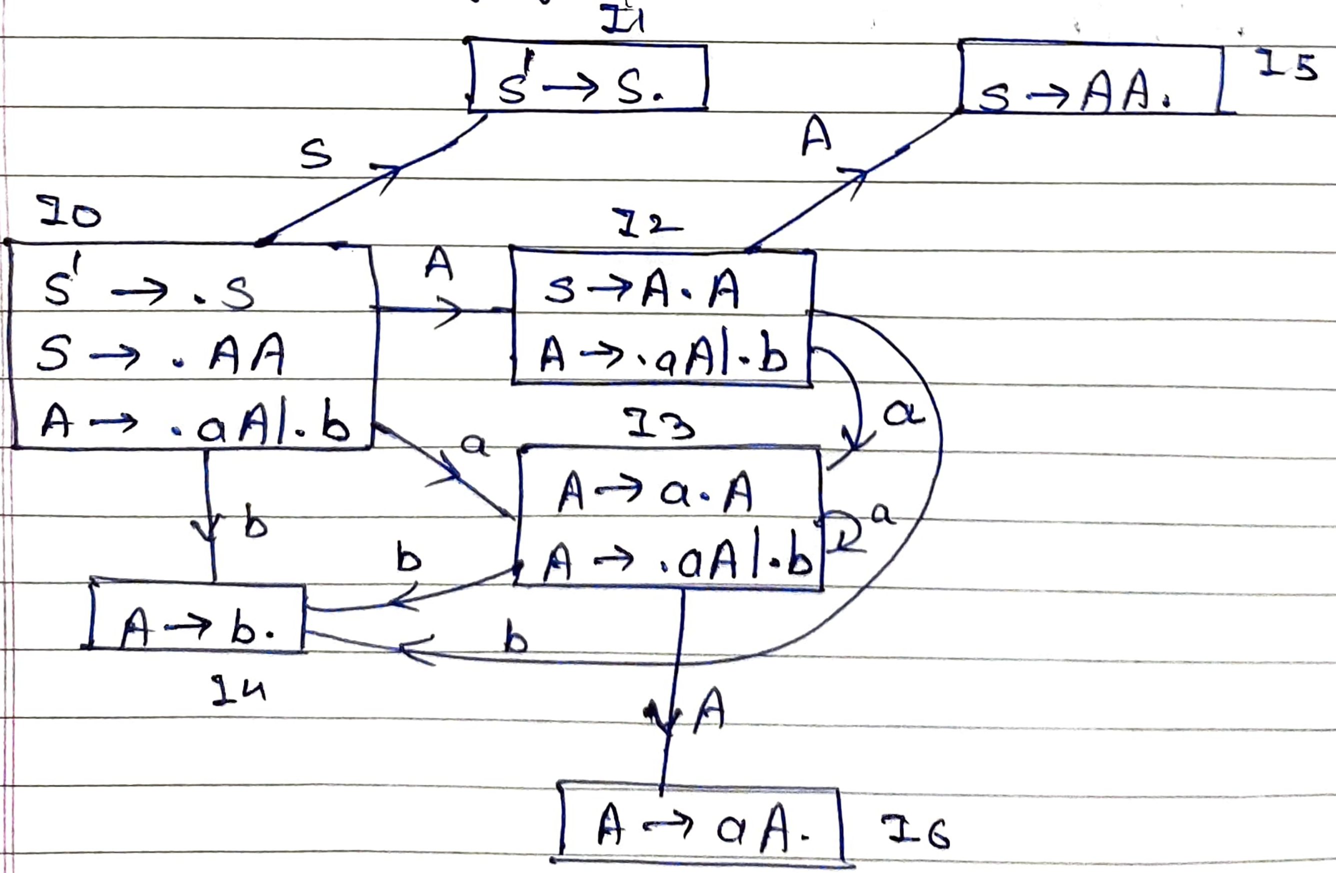
step 2:- augmented grammar.

$S' \rightarrow .S$

$S \rightarrow .AA$

$A \rightarrow .aA \mid .b$.

step 3:- applying closure



step 4 :- SLR(1) parsing table.
goto is same as LR(0)
shift actions are same as LR(0)
reduce actions are different than LR
don't write reduce action for whole row. Write only in follow's column.

	action			goto	
	a	b	~~€~~ $	A	S
0	S3	S4		2	1
1			accept		
2	S3	S4		5	
3	S3	S4		6	
4	r4				
5			r5		
6	r6				

follow(S) = $
follow(A) = a

CLR parsing table.

CLR & LALR use LR1 canonical items

~~$S \rightarrow aAd \mid aBe \mid bAe$~~

$S \rightarrow aAd \mid bBd \mid aBe \mid bAe$

$A \rightarrow c$

$B \rightarrow c$

step 1:-

$S \rightarrow aAd$ - ①

$S \rightarrow bBd$ - ②

$S \rightarrow aBe$ - ③

$S \rightarrow bAe$ - ④

$A \rightarrow c$ - ⑤

$B \rightarrow c$ - ⑥

step 2:- Augmented grammar $ is lookahed

$S' \rightarrow .S, \$$

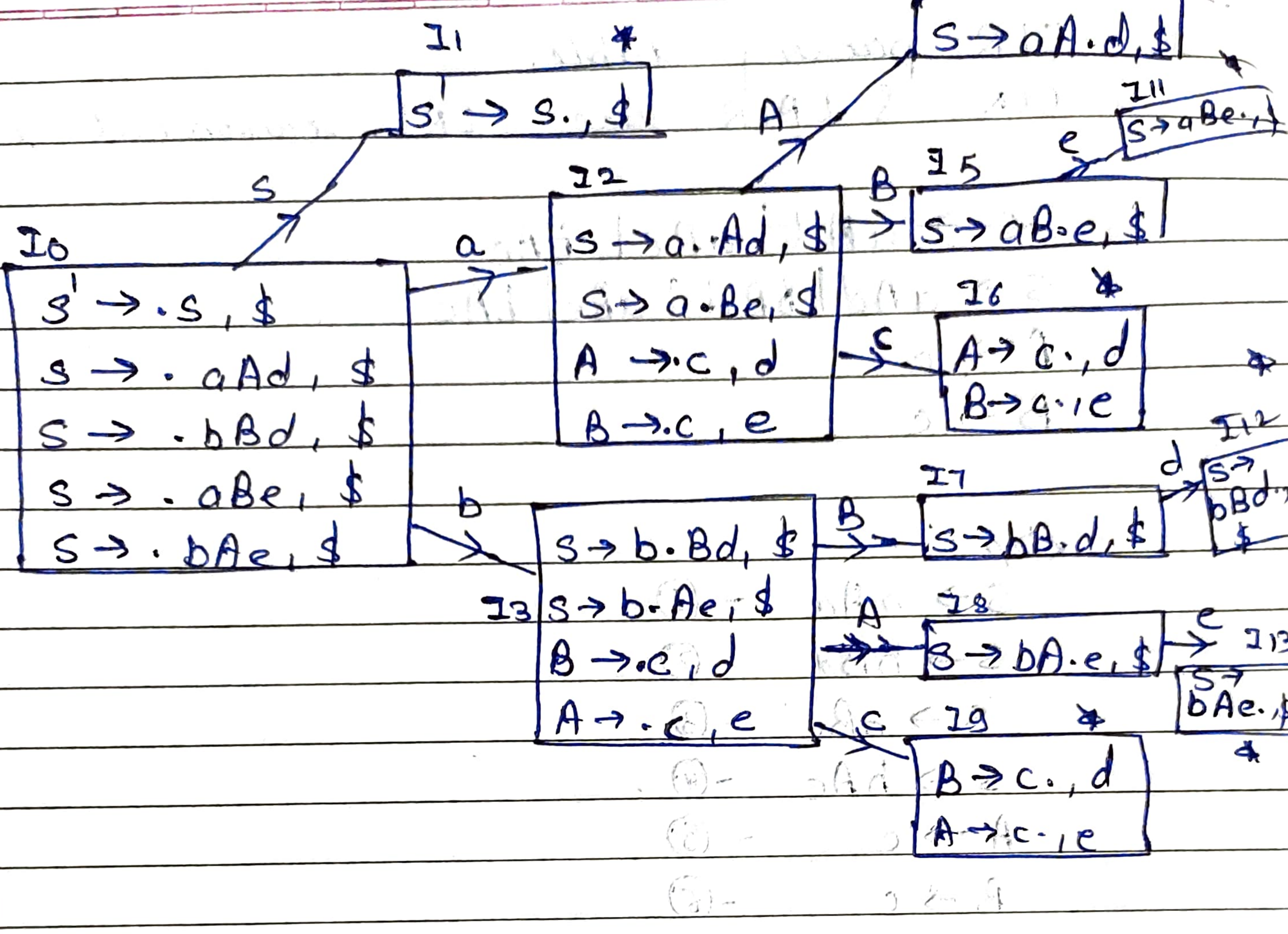
$S \rightarrow .aAd, \$$

$S \rightarrow .bBd, \$$

$S \rightarrow .aBe, \$$

$S \rightarrow .bAe, \$$

step 3:- closure apply.



step 4:- CLR parsing table.
goto is same as LR(0)
shift moves are same as LR(0).
reduce moves should not be written in whole row. Reduce moves should be written only in lookahead columns.

	action						goto		
	a	b	c	d	e	$	A	B	S
0	S_2	S_3				~~accept~~			1
1						accept			
2			S_6				4	5	
3			S_9				8	7	
4				S_{10}					
5					S_{11}				
6				r_5	r_6				
7				S_{12}					
8					S_{13}				
9				r_6	r_5				
10					~~[illegible]~~	r_1			
11						r_3			
12						r_2			
13						r_4			