Bottom-Up Parsing

* A bottom-up parsor corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top).

* In this process, parser tries to identify R.H.S of production rule and replace it by corresponding L.H.S. This activity is called **Reduction**.

* Consider the Grammar:

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow id$$

input : id * id

So the reductions will be

id * id
↓
F * id
↓
T * id
T * F
T
E ← Start symbol.

The Goal of bottom-up parsing is to construct a derivation in reverse.

Handle Pruning

* The crucial task in bottom-up parsing is to find substring that could be reduced by appropriate non-terminal. Such a substring is called handle.

Example: Consider the grammar

$E \rightarrow E + E$

$E \rightarrow id$

input: id + id + id

Right Sentential Form	Handle	Reducing Productions
id + id + id	id	$E \rightarrow id$
E + id + id	id	$E \rightarrow id$
E + E + id	id	$E \rightarrow id$
E + E + E	E + E	$E \rightarrow E + E$
E + E	E + E	$E \rightarrow E + E$
E		

Here, the underlined are handles.

Thus bottom parser is essentially a process of detecting handles and using them in reduction. this process is called handle Pruning.

Shift - Reduce Parser

It is a form of bottom-up parsing in which stack holds grammar and an input buffer holds the input string to be parsed.

During a left to right scan of the input string, the parser shifts zero or more input symbols onto the stack, until it is ready to reduce a string β of grammar symbols on top of the stack.

The parser reduce β to appropriate production.

The parser repeat this cycle until it has detected an error or until the stack contains the start symbol and the input buffer is empty.

This parser performs following basic operations

(1) Shift :→ Shift the next input symbol onto the top of stack.

(2) Reduce :→ The right end of the string to be reduced must be at the top of stack. Locate the left end of the string within the stack and decide with what non-terminal to replace the string.

(3) Accept :→ If the stack contain start symbol only and input buffer is empty, then successful completion of parsing.

(4) Error :→ A situation in which parser cannot

either shift or reduce the symbols, it cannot even perform the accept action is called an error.

Question:> $E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow id$

Perform shift-Reduce Parsing of the input string $id_1 * id_2$

Stack	Input Buffer	Action
$	$id_1 * id_2 \$$	shift
$ id_1	$* id_2 \$$	reduce by $F \rightarrow id$
$ F	$* id_2 \$$	reduce by $T \rightarrow F$
$ T	$* id_2 \$$	shift
$ $T*$	$id_2 \$$	shift
$ $T*id_2$	$	reduce by $F \rightarrow id$
$ $T*F$	$	reduce by $T \rightarrow T*F$
$ T	$	reduce by $E \rightarrow T$
$ E	$	Accept

Note: The handle will always appear on top of stack never inside.

Q Consider the grammar:

$S \rightarrow TL;$

$T \rightarrow int \mid float$

$L \rightarrow L, id \mid id$

Parse the input string int id, id; using shift reduce parser.

Ans:-

Stack	Input Buffer	Parsing Action
$	int id, id; $	shift
$int	id, id; $	reduce by T→int
$T	id, id; $	shift
$T id	, id; $	reduce by L→id
$TL	, id; $	shift
$TL,	id; $	shift
$TL, id	; $	reduce by L→L,id
$TL	; $	shift
$TL;	$	reduce by S→TL;
$S	$	Accept.

Rules:→

(1) If the incoming operator has more priority than in stack operators, then perform shift.

(2) If the stack operators has same or more priority than the priority of incoming operator then perform Reduce.

Conflicts during Shift-Reducing Parsing

* Shift reduce parsing does not tell about how to find handle. Shift reduce parsing is efficient, when handle is identified.

* Stack contents and next i/p symbol may not decide action:

  (i) Shift/reduce conflict:→ Whether make a shift operation or a reduction.

  (ii) Reduce/Reduce conflict:→ cannot decide which of several reductions to make.

* If a shift-reduce parser cannot be used for a grammar, that grammar is called non-LR(K) grammar.

Q Consider the following grammar :→

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$

Perform shift-Reduce parsing of the input string "$id_1 + id_2 + id_3$".

Solution

Stack	Input Buffer	Parsing Action
$	$id_1 + id_2 + id_3$ $	shift
$ id_1	$+id_2 + id_3$ $	Reduce by $E \rightarrow id$
$ E	$+id_2 + id_3$ $	shift
$ $E+$	$id_2 + id_3$ $	shift
$ $E+id_2$	$+id_3$ $	Reduce by $E \rightarrow id$
$ $E+E$	$+id_3$ $	shift
$ $E+E+$	id_3 $	shift
$ $E+E+id_3$	$	Reduce by $E \rightarrow id$
$ $E+E+E$	$	Reduce by $E \rightarrow E+E$
$ $E+E$	$	Reduce by $E \rightarrow E+E$
$ E	$	Accept.

Q Consider the following grammar :

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

Parse the input string $(a, (a,a))$ using shift-reduce parser.

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

LR Parsers

- This is the most efficient method of bottom-up parsing which can be used to parse the large class of content free grammars.
- This method is also called LR(K) parsing.

  L → stands for left to right scanning

  R → stands for rightmost derivation in reverse.

  K → number of input symbols. When K is omitted K is assumed to be 1.

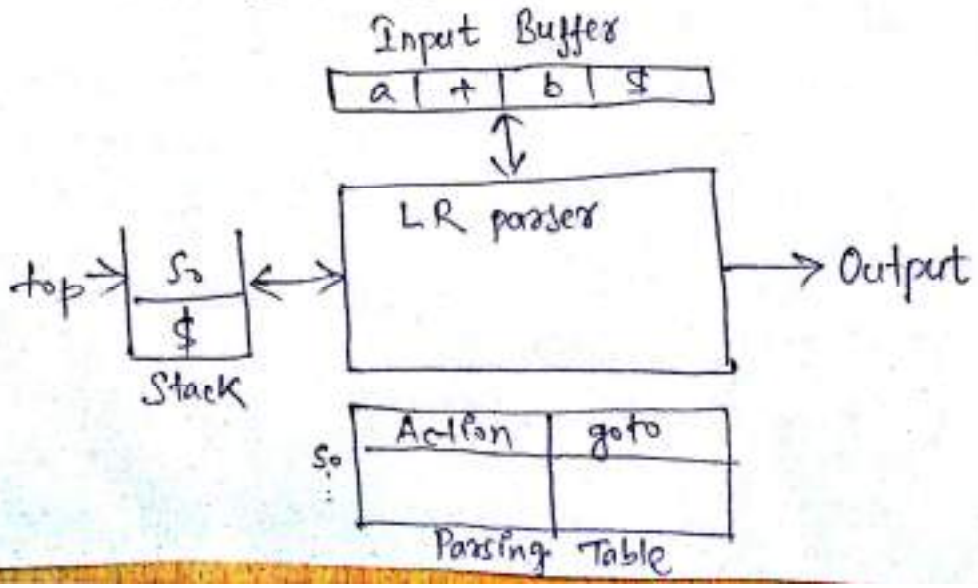
Properties of LR parser :→

(1) LR parsers can be constructed to recognize most of the programming languages for which content free grammar can be written.

(2) The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers.

(3) LR parser works using non backtracking shift reduce technique.

Structure of LR parsers



It consists of input buffer for storing the input string, stack for storing the grammar symbols, output and a parsing table comprised of two parts, action and goto.

The parsing algorithm reads the input symbol one at a time from the input buffer.

The parsing algorithm works as follows :=>

(1) It initializes the stack with start symbol and invokes scanner (lexical analyzer) to get next token.

(2) It determines S_j the state currently on the top of the stack and a_i the current input symbol.

(3) It consult the parsing table for the action $[S_j, a_i]$ which can have one of the four values.

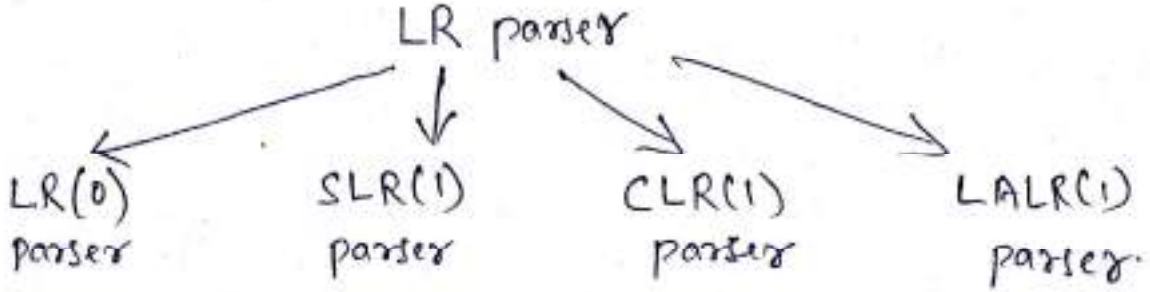
(i) S_i means shift state i.

(ii) r_j means reduce by rule j.

(iii) Accept means successful parsing is done.

(iv) Error.

Types of LR parser



SLR means Simple LR parser

CLR means Canonical LR parser

LALR means LookAhead LR parser

Note: (1) If a grammar is LR grammar, then the grammar should be unambiguous.

But converse need not be true.

(2) For all LR parsers, the parsing algorithm is same but parsing table is different.

LR Parsing Algorithm

If X is the state on top of stack and 'a' is the lookahead symbol.

(i) If action $[X, a] = S_i$, then shift a and i into stack and increment the input pointer.

(ii) If action $[X, a] = r_j$, and if j^{th} production is $\alpha \rightarrow \beta$ then pop $2*|\beta|$ symbols and replace by α.

If S_{m-1} is the state below α, then push goto $[S_{m-1}, \alpha]$

(iii) If action $[X, a]$ = accept, then successful completion of parsing.

(iv) If action $[X, a]$ = blank, then parsing error.

Q. Apply LR parsing algorithm on the following grammar to parse the given input string.

$S \rightarrow AA$

$A \rightarrow aA \mid b$

String $w = aabb\$$

	action			goto	
	a	b	$	S	A
0	s_3	s_4		1	2
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		

Solution:-

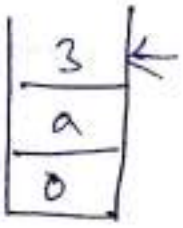
aabb$
↑

Stack

∴ action [0, a] = s_3

shift 'a' and '3' into stack, and increment input pointer.

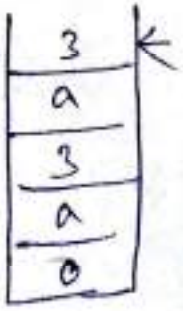
aabb$
 ↑



action [3, a] = s_3

shift 'a' and '3' into stack, and increment input pointer.

aabb$
  ↑



action [3, b] = s_4

shift 'b' and '4' into stack and increment input pointer.

a a b b $
↑

action [4, b] = r_3

$A \rightarrow b$

∴ pop $2 * |\beta|$

$2 \times 1 = 2$ symbols and replace by A.

also push goto [3, A] = 6

a a b b $
↑

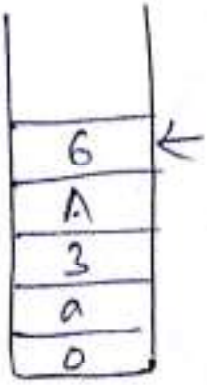
action [6, b] = r_2

$A \rightarrow aA$

∴ pop $2 * |\beta|$

$= 2 \times 2 = 4$ symbols and replace by A

also push goto [3, A] = 6



a a b b $
↑

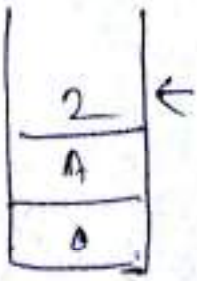
action [6, b] = r_2

$A \rightarrow aA$

∴ pop $2 * |\beta|$

$= 2 \times 2 = 4$ symbols and replace by A

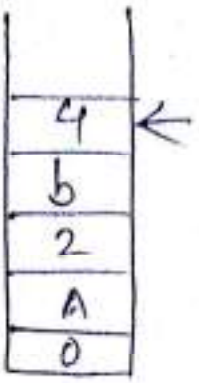
also push goto [0, A] = 2



a a b b $
↑

action [2, b] = s4

shift 'b' and '4' into stack, and increment input pointer



a a b b $

↑

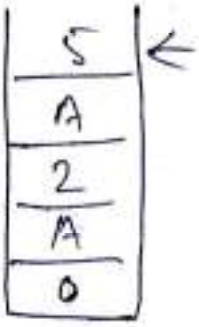
action [4, $] = r3

⇓

A → b

∴ pop 2 * |β|

= 2 × 1 = 2 symbols and replace by A.



and push goto [2, A] = 5

a a b b $

↑

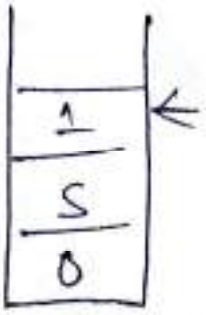
action [5, $] = r1

⇓

S → AA

∴ pop 2 * |β|

= 2 × 2 = 4 symbols and replace by S.



and push goto [0, S] = 1

a a b b $

↑

action [1, $] = accept

∴ succesful parsing.

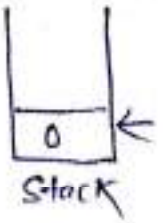
Q

$S \rightarrow AA$

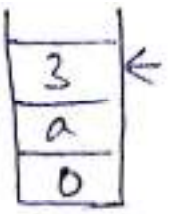
$A \rightarrow aA \mid b$

String $w = abab\$$

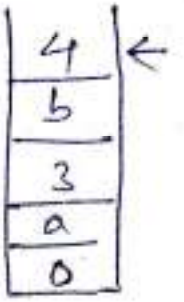
Solution.



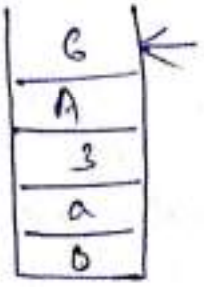
abab$



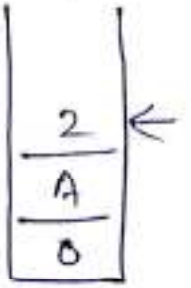
abab$



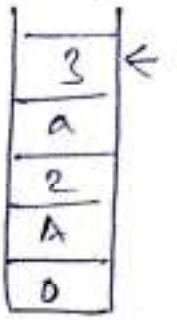
abab$



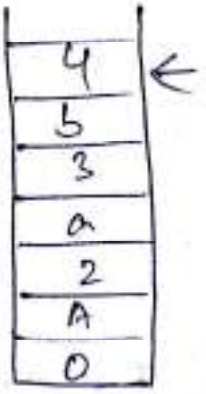
abab$



abab$



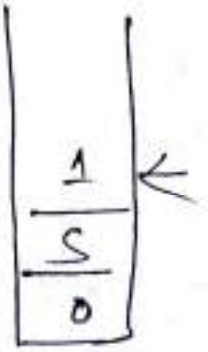
abab$



abab$

abab$

abab$



abab$

action[1, $] = accept

∴ Successful parsing.

LR(0) Parsing Table Construction

⇓

closure()

goto()

$S \to AA$

$A \to aA \mid b$

$S \to .xyz$
$S \to x.yz$
$S \to xyz.$
} LR(0) items

$S \to xyz.$ ⇓

completed LR(0) items
or
Final LR(0) items
or
Reduced LR(0) items

Augmented Grammar

$S' \to S$
$S \to AA$
$A \to aA \mid b$

Closure()

Definition: Closure(I) = ① Add I to closure(I)

② If $A \to .BCD$ is I and $B \to EF$ is in G.

then add $B \to .EF$ to closure(I) also.

③ Repeat 2nd step for every new LR(0) item.

Suppose: Grammar is

$S \to AA$
$A \to aA \mid b$

⇓

Augmented Grammar

$S' \to S$
$S \to AA$
$A \to aA \mid b$

closure ($S' \to .S$) = ① $S' \to .S$
② $S \to .AA$
③ $A \to .aA$
$.b$

closure ($S \to A.A$) = ① $S \to A.A$
② $A \to .aA$
$.b$

closure ($A \to a.A$) = ① $A \to a.A$
② $A \to .aA$
$.b$

closure ($A \to .aA$) = ① $A \to .aA$

Goto()

Definition:→ Goto (I, x) = ① Add I to Goto (I, x) also by moving dot after x.
② Apply closure to step-1.

Suppose Grammar: $S \to AA$
$A \to aA \mid b$

Augmented Grammar
$S' \to S$
$S \to AA$
$A \to aA \mid b$

(1) Goto ($S' \to .S, S$) = ① $S' \to S.$

(2) Goto ($S \to .AA, A$) = ① $S \to A.A$
$A \to .aA$
$.b$

(3) Goto ($A \to .aA, a$) = ① $A \to a.A$
$A \to .aA$
$.b$

Q1 Construct LR(0) parsing table for the following grammar.

$S \rightarrow AA$

$A \rightarrow aA \mid b$

Solution :>

$S \rightarrow AA$ (1)

$A \rightarrow aA$ (2) $\mid b$ (3)

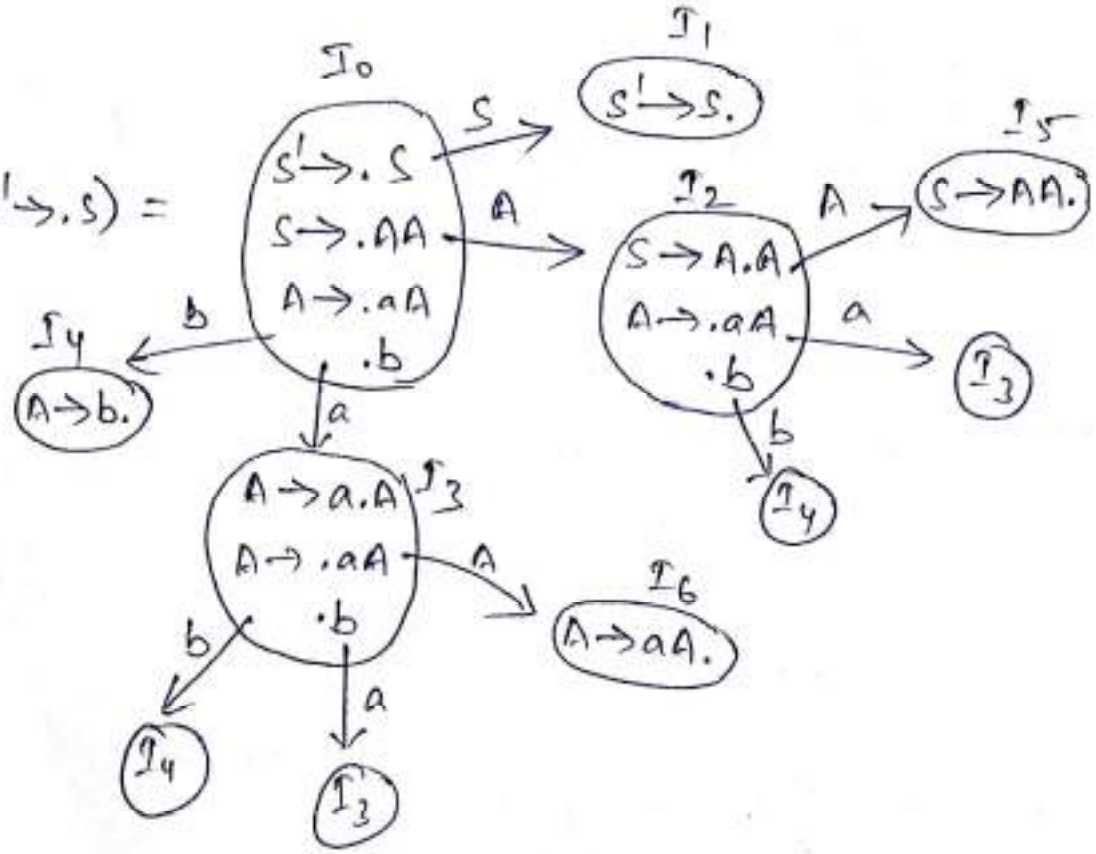
∴ The augmented grammar is

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA \mid b$

∴ closure ($S' \rightarrow .S$) =



	action			goto	
	a	b	$	S	A
0	S_3	S_4		1	2
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		

fig. LR(0) Parsing table.

Given grammar is LR(0) grammar because no entry of the table contain multiple values

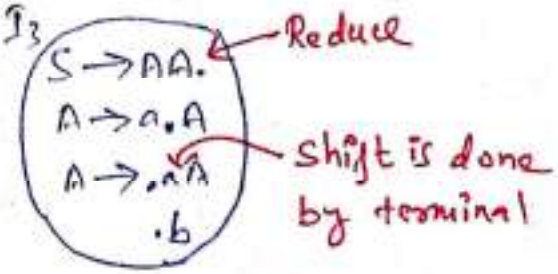
Types of conflicts

① S-R conflict (or) Shift-Reduce conflict

In SR conflict, the shift conflict is considered for terminal only.

Reduce means dot at the end.

e.g. I_3



	action		
	a	b	$
3	S_3/r_1	S_3/r_1	S_3/r_1

multiple entries in one cell.

(2) Reduce-Reduce conflict

I_5 (A → b. , S → aA.)

	action		
	a	b	$
5	r_1/r_3	r_1/r_3	r_1/r_3

multiple entries in one cell.

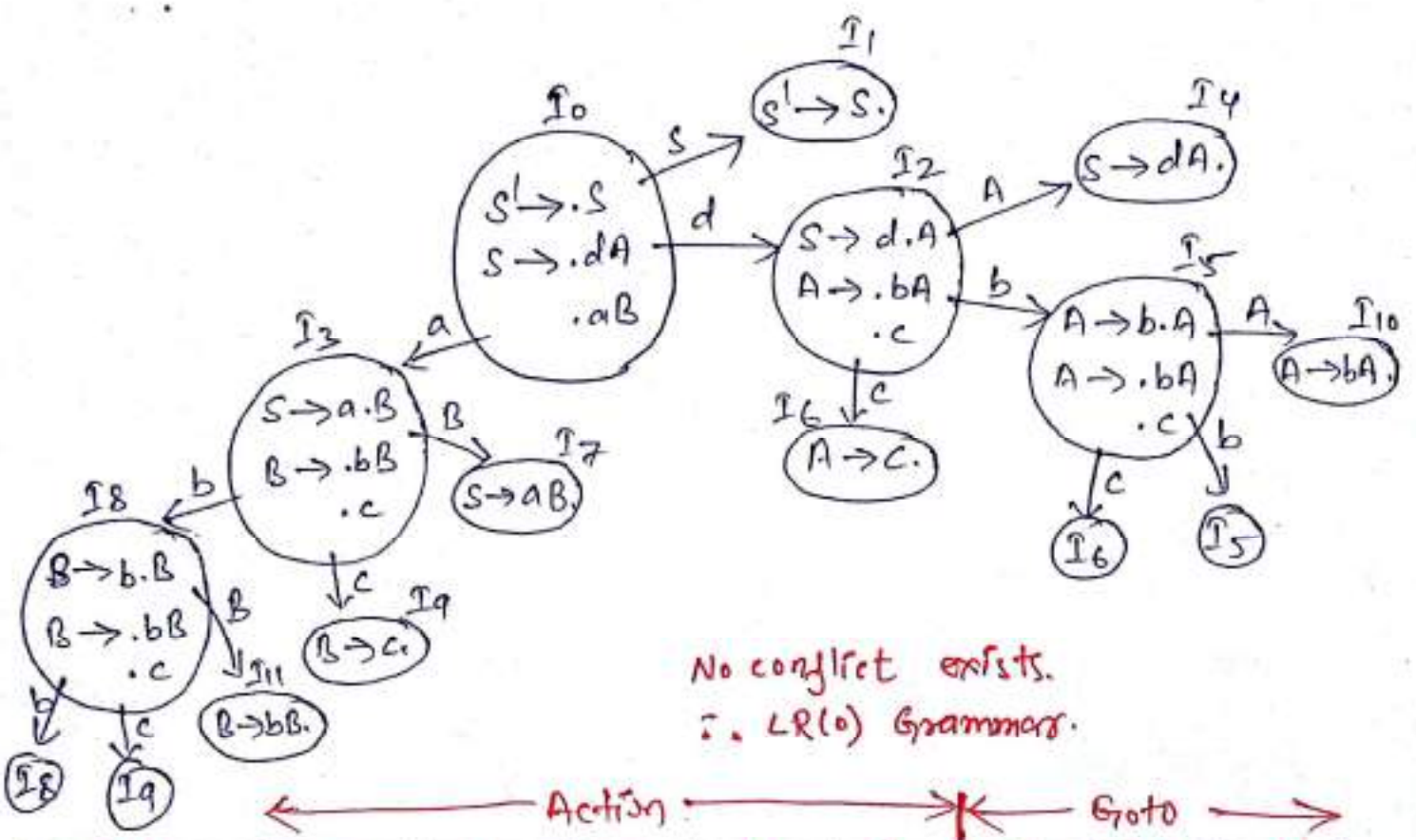
Q2 check for LR(0) Grammar :→

$S \to dA \mid aB$

$A \to bA \mid c$

$B \to bB \mid c$

Soln



No conflict exists.

∴ LR(0) Grammar.

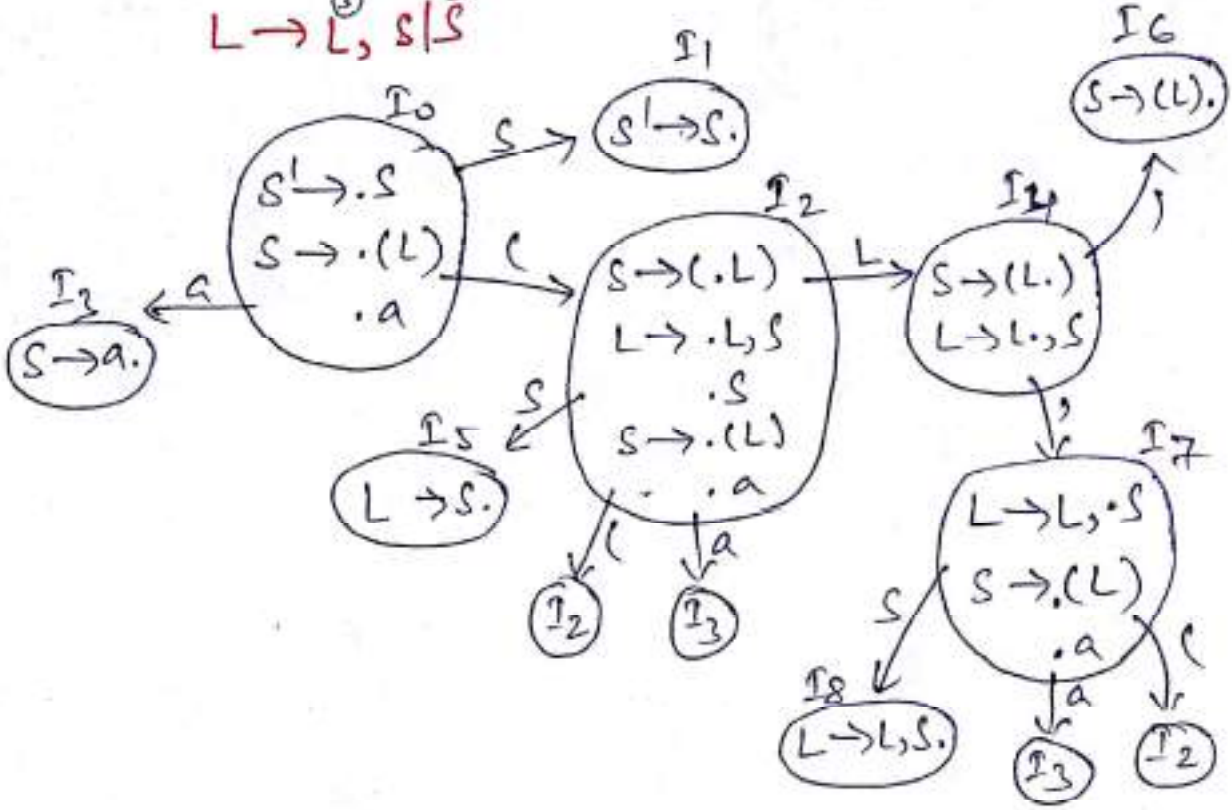
	← Action →					← Goto →		
States	a	b	c	d	$	S	A	B
0	S_3			S_2		1		
1					accept			
2		S_5	S_6				4	
3		S_8	S_9					7
4	r_1	r_1	r_1	r_1	r_1			
5		S_5	S_6				10	
6	r_4	r_4	r_4	r_4	r_4			
7	r_2	r_2	r_2	r_2	r_2			
8		S_8	S_9					11
9	r_6	r_6	r_6	r_6	r_6			
10	r_3	r_3	r_3	r_3	r_3			
11	r_5	r_5	r_5	r_5	r_5			

Q3 Check for LR(0) Grammar.

$S \to (L) \,|\, a$ ① ② ③ ④

$L \to L, S \,|\, S$

Soln



States	Action a	(	)	,	$	Goto S	L
0	S_3	S_2				1	
1					accept		
2	S_3	S_2				5	4
3	r_2	r_2	r_2	r_2	r_2		
4			S_6	S_7			
5	r_4	r_4	r_4	r_4	r_4		
6	r_1	r_1	r_1	r_1	r_1		
7	S_3	S_2				8	
8	r_3	r_3	r_3	r_3	r_3		

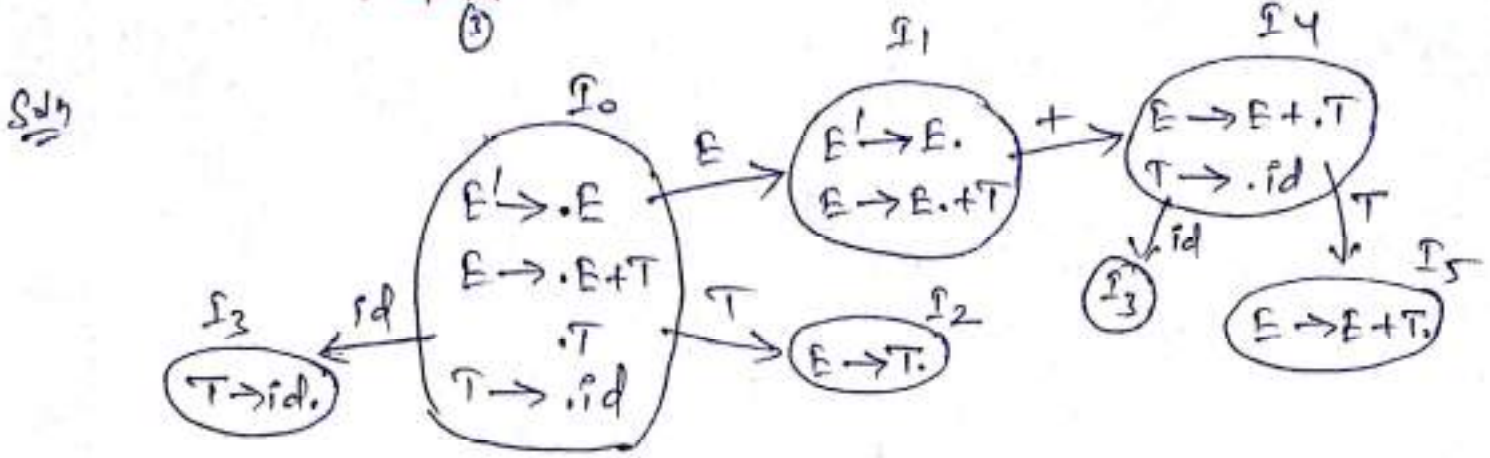
No conflict exists

∴ LR(0) Grammar

Q4 Check for LR(0) Grammar:

$E \to E +^{(1)} T \mid^{(2)} T$

$T \to id$ (3)



States	id	+	$	E	T
	← Action →			← Goto →	
0	S_3			1	2
1		S_4	accept		
2	r_2	r_2	r_2		
3	r_3	r_3	r_3		
4	S_3				5
5	r_1	r_1	r_1		

∴ No conflict exist.

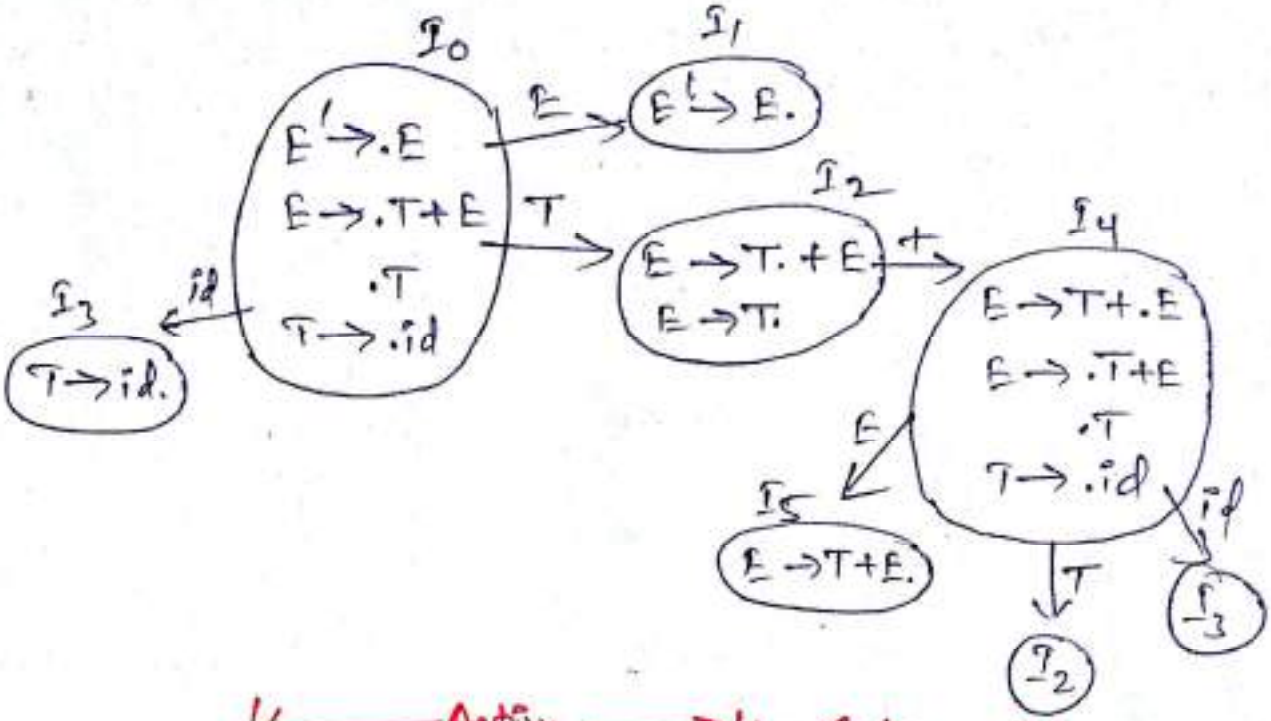
∴ LR(0) Grammar.

Q.5 check for LR(0) Grammar:

$E \to T + E \mid T$

$T \to id$

Solution



	← Action →			← Goto →	
States	id	+	$	E	T
0	s_3			1	2
1			accept		
2	r_2	s_4/r_2	r_2		
3	r_3	r_3	r_3		
4	s_3			5	2
5	r_1	r_1	r_1		

Here Action[2, +] = s_4/r_2

∴ SR conflict exist

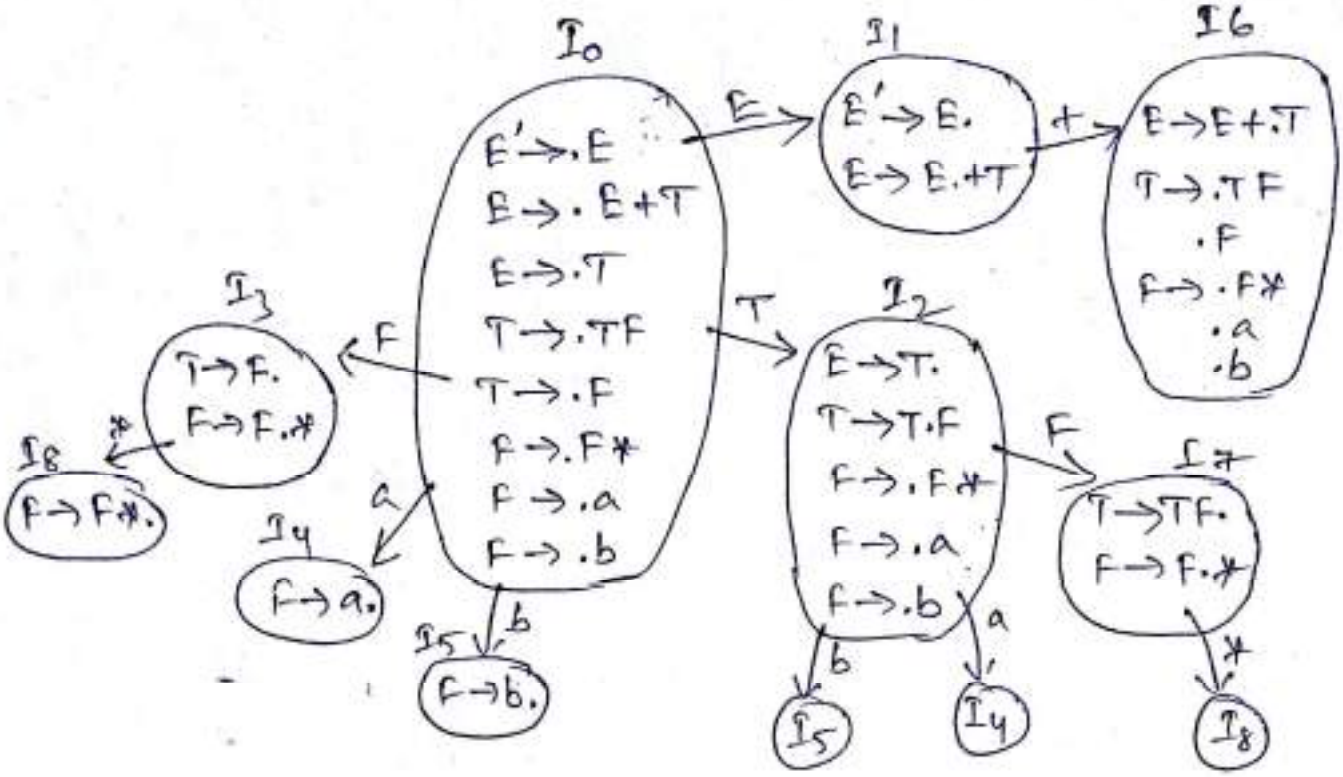
∴ Not LR(0) grammar

Q6 Check for LR(0) grammar.→

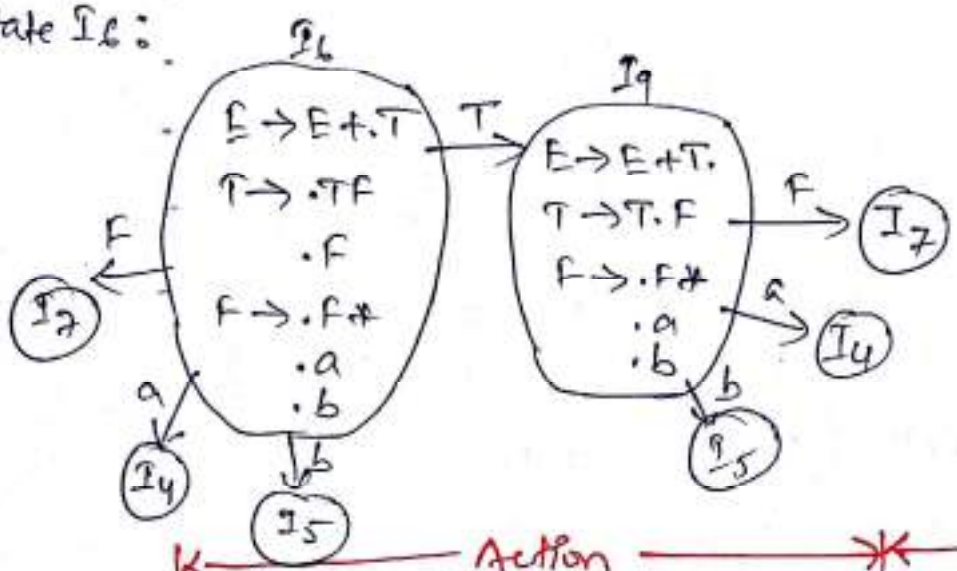
$E \to \overset{①}{E} + T \mid \overset{②}{T}$

$T \to \overset{③}{T} F \mid \overset{④}{F}$

$F \to F^{\overset{⑤}{*}} \mid \overset{⑥}{a} \mid \overset{⑦}{b}$



State I_6:



In LR(0) table, SR conflict exists

∴ Not LR(0) Grammar

	Action					Goto		
States	a	b	+	*	$	E	T	F
0	S_4	S_5				1	2	3
1			S_6		accept			
2	S_4/r_2	S_5/r_2	r_2	r_2	r_2			7
3	r_4	r_4	r_4	S_8/r_4	r_4			
4	r_6	r_6	r_6	r_6	r_6			
5	r_7	r_7	r_7	r_7	r_7			
6	S_4	S_5					9	7
7	r_3	r_3	r_3	S_8/r_3	r_3			
8	r_5	r_5	r_5	r_5	r_5			
9	S_4/r_1	S_5/r_1	r_1	r_1	r_1			7

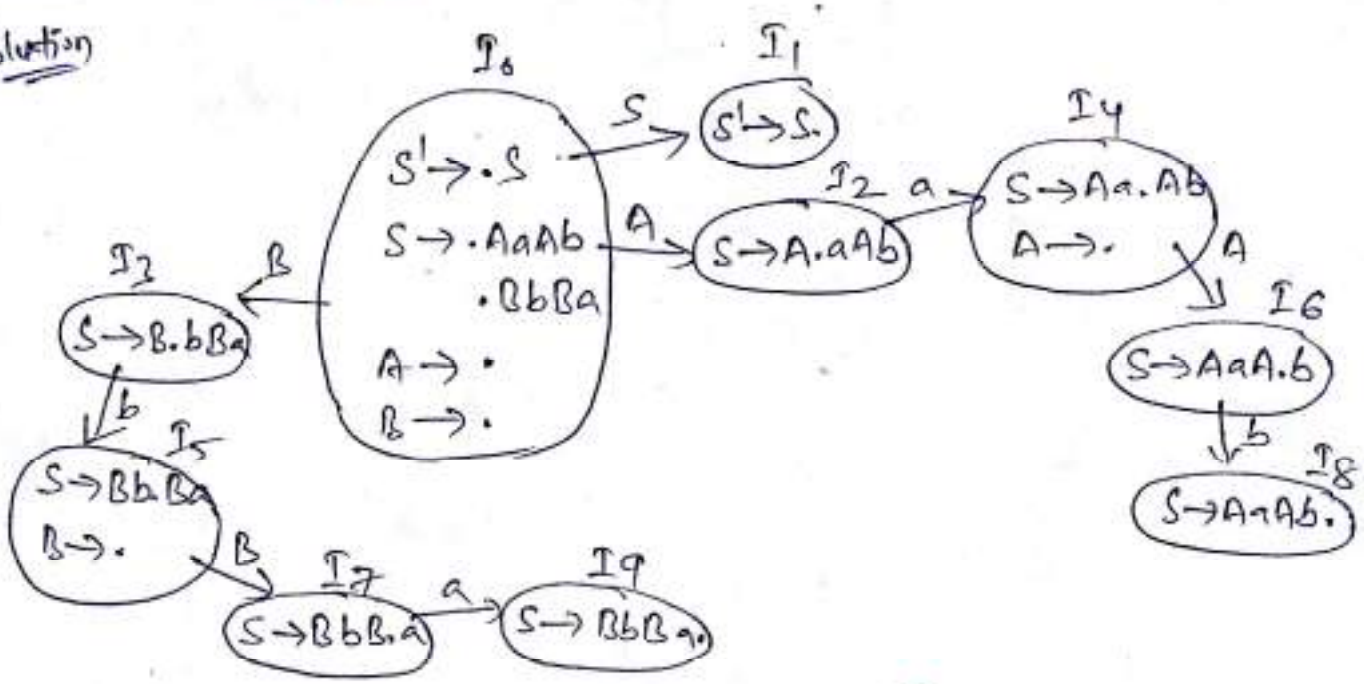
Q7 Check for LR(0) grammar:

$S \rightarrow \overset{①}{A}aAb \mid \overset{②}{B}bBa$

$A \rightarrow \overset{③}{\epsilon}$

$B \rightarrow \overset{④}{\epsilon}$

Solution



	← Action →			← Goto →		
States	a	b	$	S	A	B
0	r_3/r_4	r_3/r_4	r_3/r_4	1	2	3
1			accept			
2	S_4					
3		S_5				
4	r_3	r_3	r_3		6	
5	r_4	r_4	r_4			7
6		S_8				
7	S_9					
8	r_1	r_1	r_1			
9	r_2	r_2	r_2			

∴ RR conflict exists

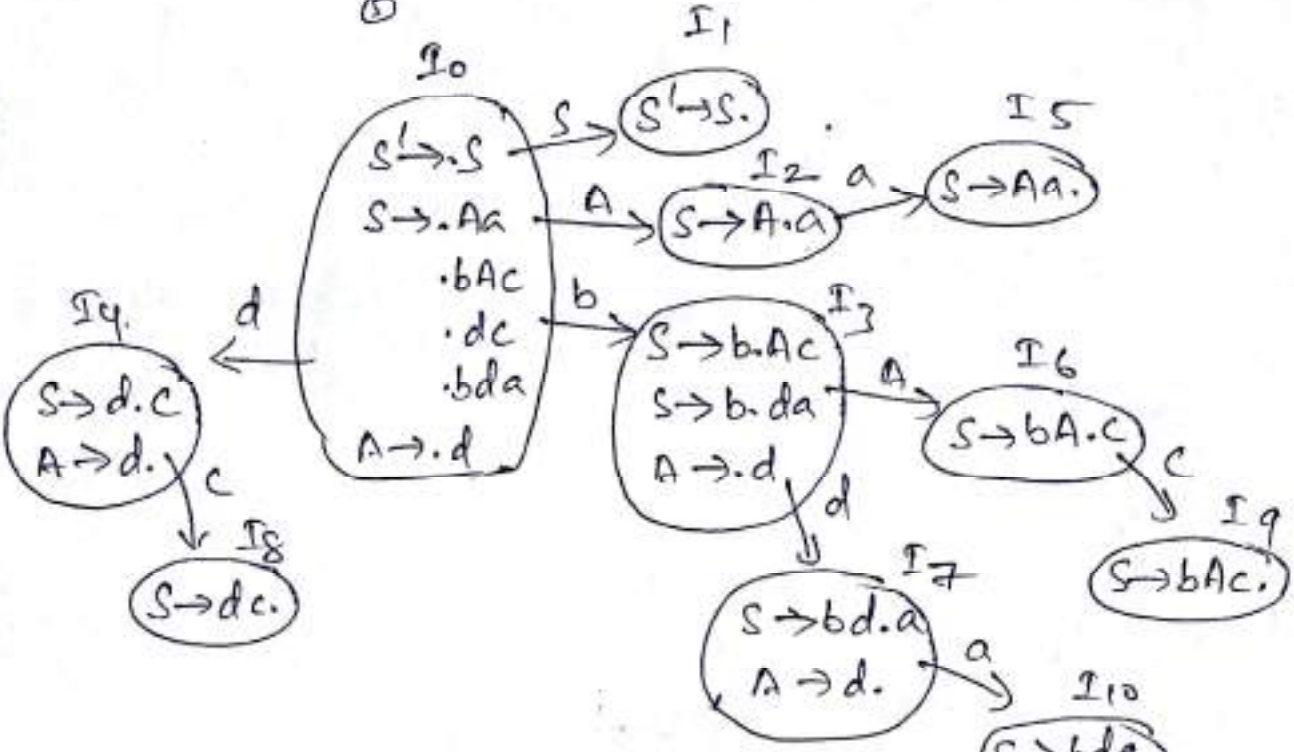
∴ Not LR(0) grammar.

Q8 Check for LR(0) grammar :→

$S \xrightarrow{(1)} Aa \mid \xrightarrow{(2)} bAc \mid \xrightarrow{(3)} dc \mid \xrightarrow{(4)} bda$

$A \xrightarrow{(5)} d$

Soln



← Action → | ← Goto →

States	a	b	c	d	$	S	A
0		S_3		S_4		1	2
1					accept		
2	S_5						
3				S_7			6
4	r_5	r_5	S_8/r_5	r_5	r_5		
5	r_1	r_1	r_1	r_1	r_1		
6			S_9				
7	S_{10}/r_5	r_5	r_5	r_5	r_5		
8	r_3	r_3	r_3	r_3	r_3		
9	r_2	r_2	r_2	r_2	r_2		
10	r_4	r_4	r_4	r_4	r_4		

∴ SR conflict exists.

∴ Not LR(0) Grammar.

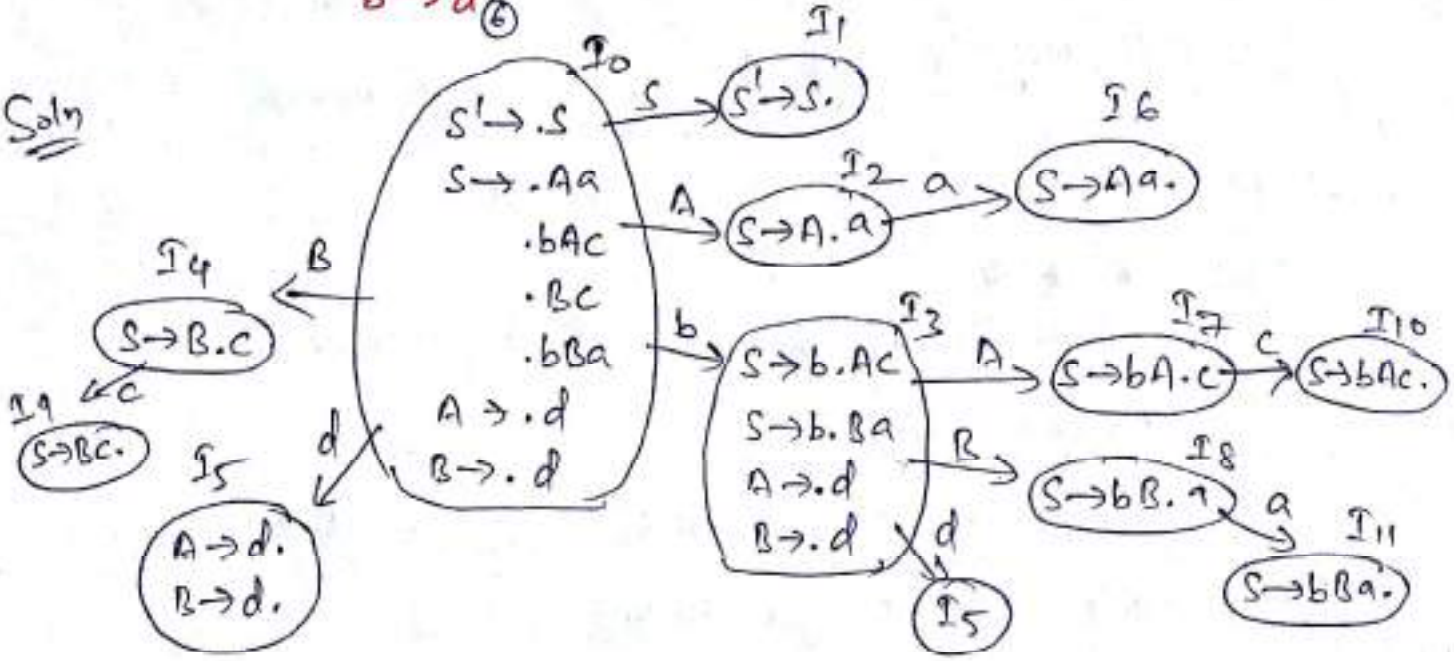
Q9 check for LR(0) grammar.

$S \to Aa^{(1)} | bAc^{(2)} | Bc^{(3)} | bBa^{(4)}$

$A \to d$ (5)

$B \to d$ (6)

Soln



	← Action →					← Goto →		
States	a	b	c	d	$	S	A	B
0		S_3		S_5		1	2	4
1					accept			
2.	S_6							
3.				S_5			7	8
4			S_9					
5	r_5/r_6	r_5/r_6	r_5/r_6	r_5/r_6	r_5/r_6			
6	r_1	r_1	r_1	r_1	r_1			
7			S_{10}					
8	S_{11}							
9	r_3	r_3	r_3	r_3	r_3			
10	r_2	r_2	r_2	r_2	r_2			
11	r_4	r_4	r_4	r_4	r_4			

∴ RR conflict exists

∴ Not LR(0) Grammar

Note:→ (1) If a Grammar is LR(0) grammar, then that grammar is SLR(1) grammar also.

(2) If a Grammar is SLR(1) grammar, then that grammar may or may not be LR(0) grammar.

SLR(1) table for Questions 1 to 9.

Question 1

Since the grammar is LR(0) grammar.

∴ The given grammar is also SLR(1).

SLR(1) parsing table will be same as that of LR(0) parsing table. Except for Reduce entries.

Follow(A) = { $ }

follow(S) = { $ }

	Action → a	b	$	← Goto → S	A
0	S_3	S_4		1	2
1			accept		
2	S_3	S_4			5
3	S_3	S_4			6
4			r_3		
5			r_1		
6			r_2		

fig SLR(1) Parsing table.

SLR(1) parsing table For Question 2

Follow(S) = {$}

Follow(A) = {$}

Follow(B) = {$}

	← Action →					← Goto →		
State	a	b	c	d	$	S	A	B
0	S_3			S_2		1		
1					accept			
2		S_5	S_6				4	
3		S_8	S_9					7
4					r_1			
5		S_5	S_6				10	
6					r_4			
7					r_2			
8		S_8	S_9					11
9					r_6			
10					r_3			
11					r_5			

fig SLR(1) parsing table.

SLR(1) parsing table For Question 3

Follow(S) = {$, ,,)}

Follow(L) = {, ,)}

	← Action →					← Goto →	
States	a	(	)	,	$	S	L
0	s_3	s_2				1	
1					accept		
2	s_3	s_2				5	4
3			r_2	r_2	r_2		
4			s_6	s_7			
5			r_4	r_4			
6			r_1	r_1	r_1		
7	s_3	s_2				8	
8			r_3	r_3			

fig SLR(1) parsing table.

SLR(1) Parsing table For Question 4

Follow (E) = { $, + }

Follow (T) = { $, + }

	← Action →			← Goto →	
States	id	+	$	E	T
0	s_3			1	2
1		s_4	accept		
2		r_2	r_2		
3		r_3	r_3		
4	s_3				5
5		r_1	r_1		

fig SLR(1) Parsing table.

SLR(1) parsing table For Question 5

Follow(E) = { $ }

Follow(T) = { +, $ }

States	id	+	$	E	T
0	S_3			1	2
1			accept		
2		S_4	r_2		
3		r_3	r_3		
4	S_3			5	2
5			r_1		

fig. SLR(1) parsing table. (No conflict)

SLR(1) parsing table For Question 6

Follow(E) = { $, + } Follow(F) = { *, $, +, a, b }

Follow(T) = { $, +, a, b }

States	a	b	+	*	$	E	T	F
0	S_4	S_5				1	2	3
1			S_6		accept			
2	S_4	S_5	r_2		r_2			7
3	r_4	r_4	r_4	S_8	r_4			
4	r_6	r_6	r_6	r_6	r_6			
5	r_7	r_7	r_7	r_7	r_7			
6	S_4	S_5					9	7
7	r_3	r_3	r_3	S_8	r_3			
8	r_5	r_5	r_5	r_5	r_5			
9	S_4	S_5						7

fig SLR(1) parsing table. (No conflict)

SLR(1) parsing table for Question 7

Follow(S) = {$}

Follow(A) = {a, b} Follow(B) = {a, b}

States	Action a	Action b	Action $	Goto S	Goto A	Goto B
0	r_3/r_4	r_3/r_4		1	2	3
1			accept			
2	S_4					
3		S_5				
4	r_3	r_3			6	
5	r_4	r_4				7
6		S_8				
7	S_9					
8			r_1			
9			r_2			

RR conflict exist. ∴ NOT SLR(1) grammar.

SLR(1) parsing table for Question 8

Follow(S) = {$} Follow(A) = {a, c}

States	Action a	Action b	Action c	Action d	Action $	Goto S	Goto A
0		S_3		S_4		1	2
1					accept		
2	S_5						
3				S_7			6
4	r_5		S_8/r_5				
5					r_1		
6			S_9				
7	S_{10}/r_5		r_5				
8					r_3		
9					r_2		
10					r_4		

SR conflict exists ∴ NOT SLR(1) Grammar

SLR(1) parsing table for Question 9

Follow(S) = {$}

Follow(A) = {a, c}

Follow(B) = {a, c}

	← Action →					← Goto →		
State	a	b	c	d	$	S	A	B
0		s_2		s_5		1	2	4
1					accept			
2	s_6							
3				s_5			7	8
4			s_9					
5	r_5/r_6		r_5/r_6					
6					r_1			
7			s_{10}					
8	s_{11}							
9					r_3			
10					r_2			
11					r_4			

RR conflict exists.

∴ NOT SLR(1) grammar.

Construction of SLR Parsing table

Input : An Augmented grammar G'

Output : SLR parsing table.

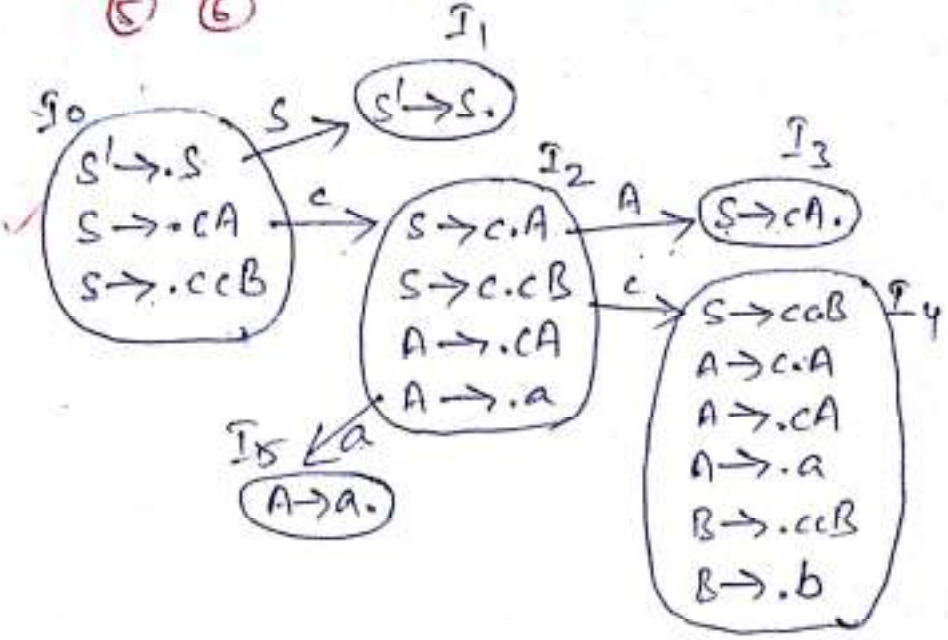
Algorithm :->

1. Initially construct set of items $C = \{I_0, I_1, I_2, \ldots I_n\}$ where C is a collection of set of LR(0) items for the input grammar G'.
2. The parsing actions are based on each item I_i. The actions are as given below.
   (a) If $A \to \alpha . a\beta$ is in I_i and goto$(I_i, a) = I_j$ then set action $[i, a]$ as "shift j". Note that 'a' must be a terminal symbol.
   (b) If there is a rule $A \to \alpha .$ is in I_i, then set action $[i, a]$ to "reduce $A \to \alpha$" for all symbols a, where $a \in$ Follow(A). Note that A must not be an augmented grammar S'.
   (c) If $S' \to S$ is in I_i, then the entry in the action table action$[i, \$]$ = "accept".
3. The goto part of the SLR table can be filled as: The goto transitions for state i is considered for non-terminals only. If goto$(I_i, A) = I_j$, then goto$[I_i, A] = j$.
4. All the entries not defined by rule 2 and 3 are considered to be "error".

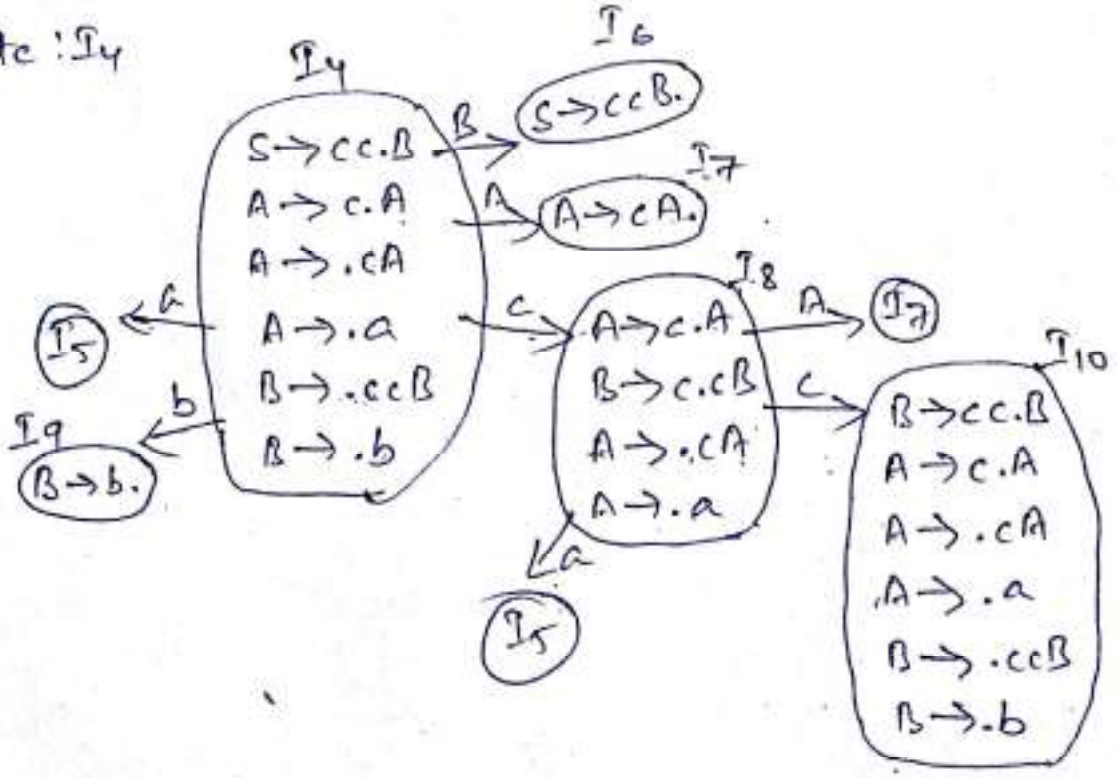
Q Construct SLR(1) parsing table for the following grammar :->

① ②
S → cA | ccB
③ ④
A → cA | a
B → ccB | b
⑤ ⑥

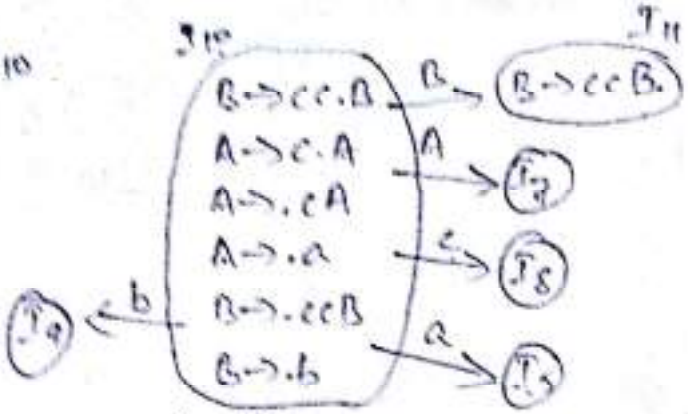
Solution :



State : I4



State: I_{10}



States	Action a	b	c	$	Goto S	A	B
0			S_2		1		
1				accept			
2	S_5		S_4			3	
3				r_1			
4	S_5	S_9	S_8			7	6
5				r_4			
6				r_2			
7				r_3			
8	S_5		S_{10}			7	
9				r_6			
10	S_5	S_9	S_8			7	11
11				r_5			

Follow(S) = {$}

Follow(A) = Follow(S) = {$}

Follow(B) = {$}

∴ SLR(1) ✓

Q. Construct SLR(1) parsing table for

$E \to E+T$ ①

$E \to T$ ②

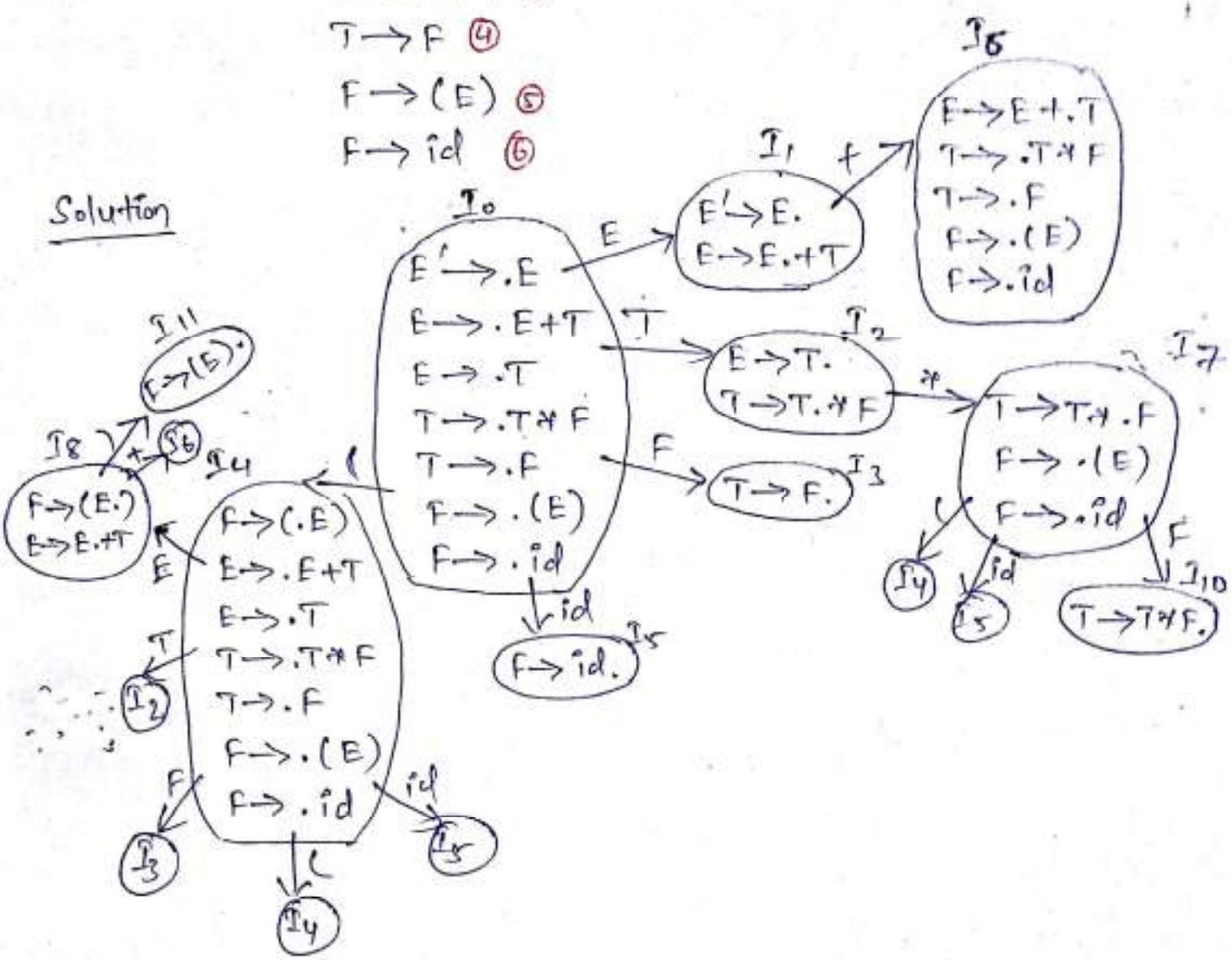
$T \to T*F$ ③

$T \to F$ ④

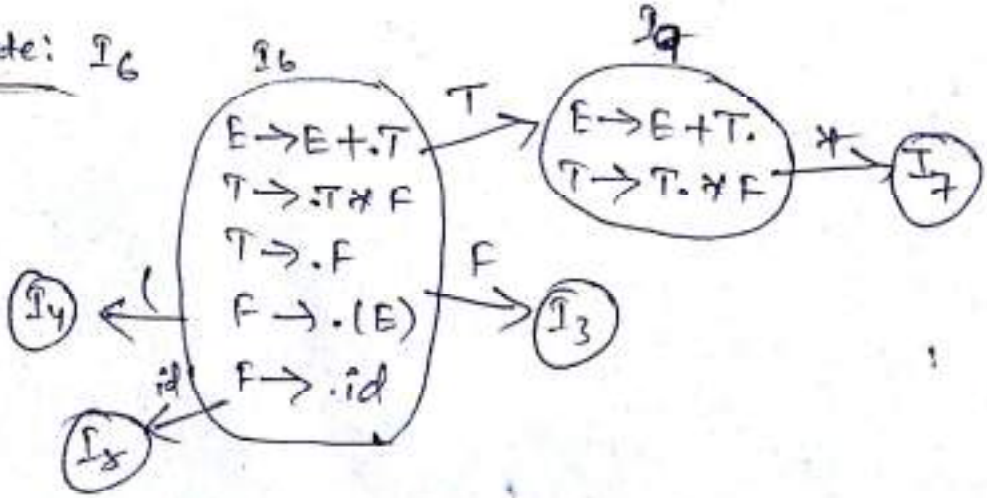
$F \to (E)$ ⑤

$F \to id$ ⑥

Solution



State: I_6



	action						goto		
	+	*	(	)	id	$	E	T	F
0			S_4		S_5		1	2	3
1	S_6					accept			
2	r_2	S_7		r_2		r_2			
3	r_4	r_4		r_4		r_4			
4			S_4		S_5		8	2	3
5	r_6	r_6		r_6		r_6			
6			S_4		S_5			9	3
7			S_4		S_5				10
8	S_6			S_{11}					
9	r_1	S_7		r_1		r_1			
10	r_3	r_3		r_3		r_3			
11	r_5	r_5		r_5		r_5			

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

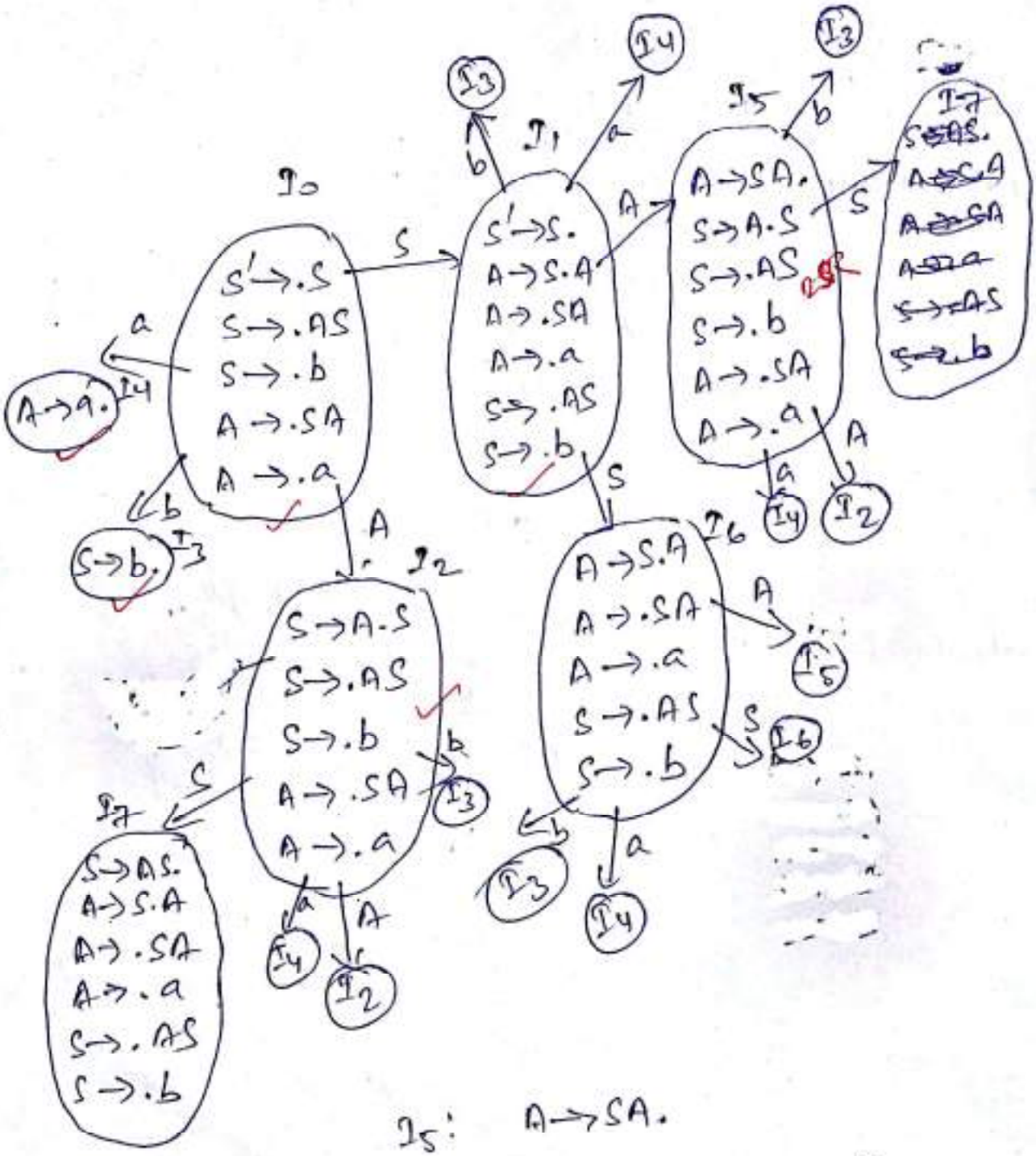
Follow(T) = {*, +,), $}

Follow(F) = Follow(T) = {*, +,), $}

Table contains unique entries. ∴ no conflict exists.

∴ grammar is SLR(1).

① ②
S → AS | b
A → SA | a
③ ④



I_5: $A \to SA.$

Follow (A) = First (S)

= {$, b, a}

Shift b is by 'a' and 'b'

∴ NOT SLR

Q Show that the following grammar is CLR(1) but not LALR.

$S \to Aa \,(1) \mid bAc \,(2) \mid Bc \,(3) \mid bBa \,(4)$

$A \to d \,(5)$

$B \to d \,(6)$

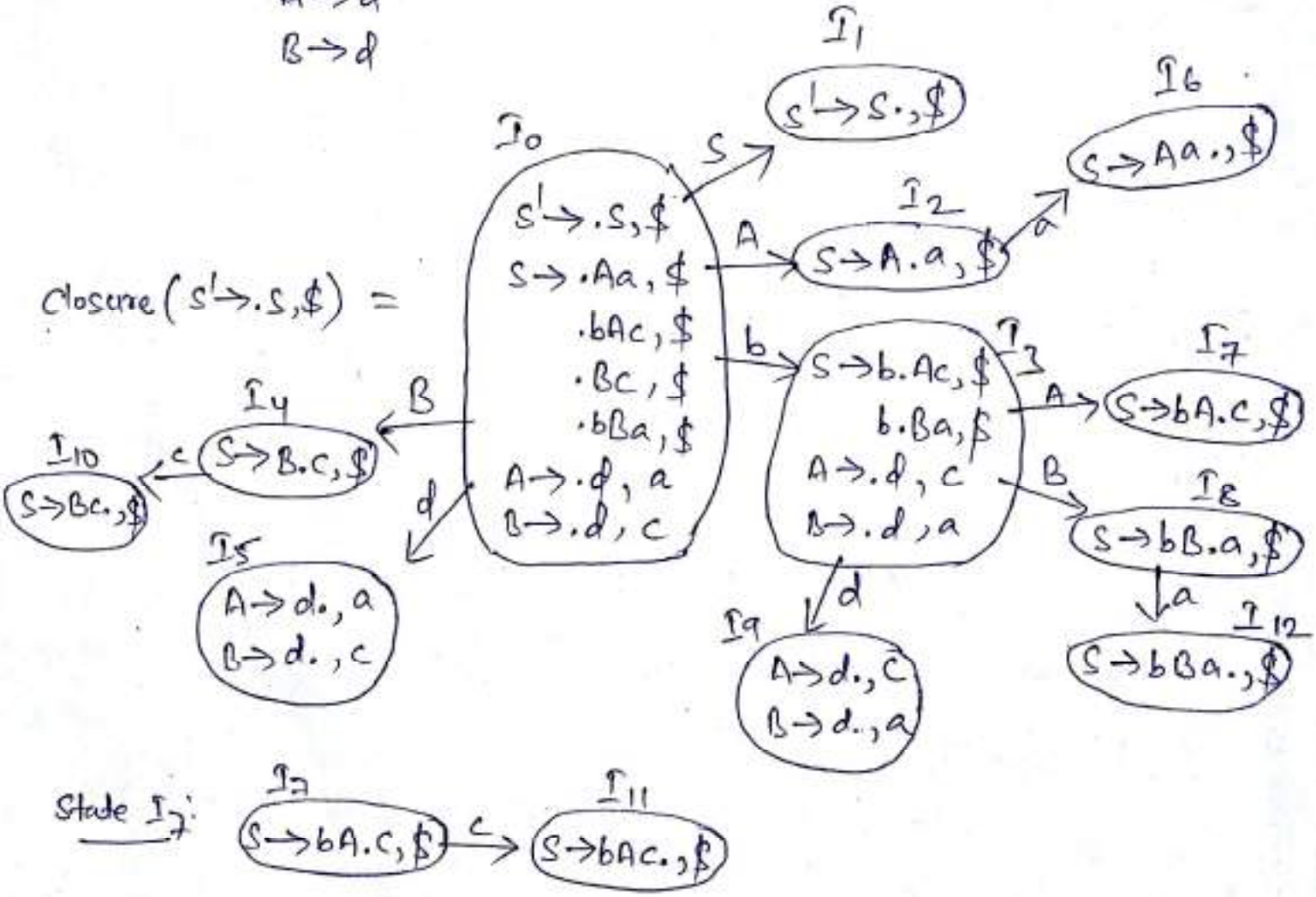
Solution:→ The Augmented Grammar is

$S' \to S$

$S \to Aa \mid bAc \mid Bc \mid bBa$

$A \to d$

$B \to d$



State I_7:

	Action					Goto		
States	a	b	c	d	$	S	A	B
0		S_3		S_5		1	2	4
1					accept			
2	S_6							
3				S_9			7	8
4			S_{10}					
5	r_5		r_6					
6					r_1			
7			S_{11}					
8	S_{12}							
9	r_6		r_5					
10					r_3			
11					r_2			
12					r_4			

fig CLR(1) parsing table.

No cell of the table contain multiple entries. ∴ the grammar is CLR(1) grammar.

Now for LALR(1) items, the states 5 and 9 can be merged.

∴ I_{59} : $A \to d.\ ,\ a/c$
$B \to d.\ ,\ c/a$

Remaining states are same as before.

	Action					Goto		
States	a	b	c	d	$	S	A	B
0		S_3		S_5		1	2	4
1					accept			
2	S_6							
3				S_9			7	8
4			S_{10}					
59	$r_5 \mid r_6$		$r_5 \mid r_6$					
6					r_1			
7			S_{11}					
8	S_{12}							
10					r_3			
11					r_2			
12					r_4			

fig. LALR(1) parsing table.

Parsing table shows multiple entries in Action[59, a] and Action[59, c]. The conflict is RR conflict i.e. reduce-reduce conflict.

∴ The given grammar is not LALR(1).

Here, no two states are same.

∴ The Grammar is CLR(1) and LALR(1).

Q Construct CLR(1) and LALR(1) parsing table for the following grammar.

$S \rightarrow AaAb$ ① | $BbBa$ ②

$A \rightarrow \in$ ③

$B \rightarrow \in$ ④

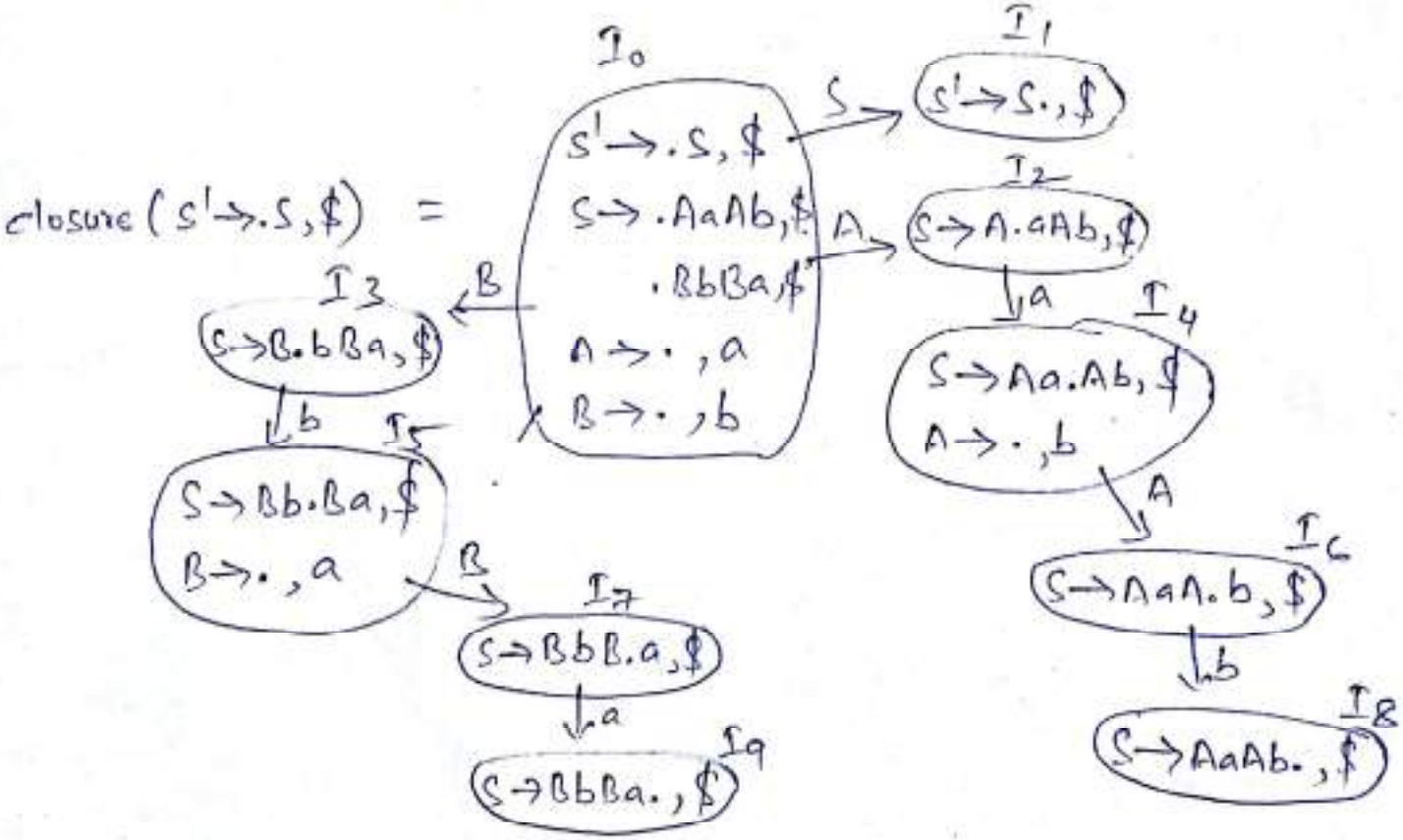
Solution The Augmented Grammar is

$S' \rightarrow S$

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$



Since, No two states are same, ∴ the states $I_0, \ldots I_9$ will also be the states in LALR(1).

	Action			Goto		
	a	b	$	S	A	B
0	r_3	r_4		1	2	3
1			accept			
2	s_4					
3		s_5				
4		r_3			6	
5	r_4					7
6		s_8				
7	s_9					
8			r_1			
9			r_2			

fig CLR(1) parsing table.

Q Construct LALR(1) parsing table for the following Grammar:

$S \rightarrow CC$ ①

$C \rightarrow aC$ ② | d ③

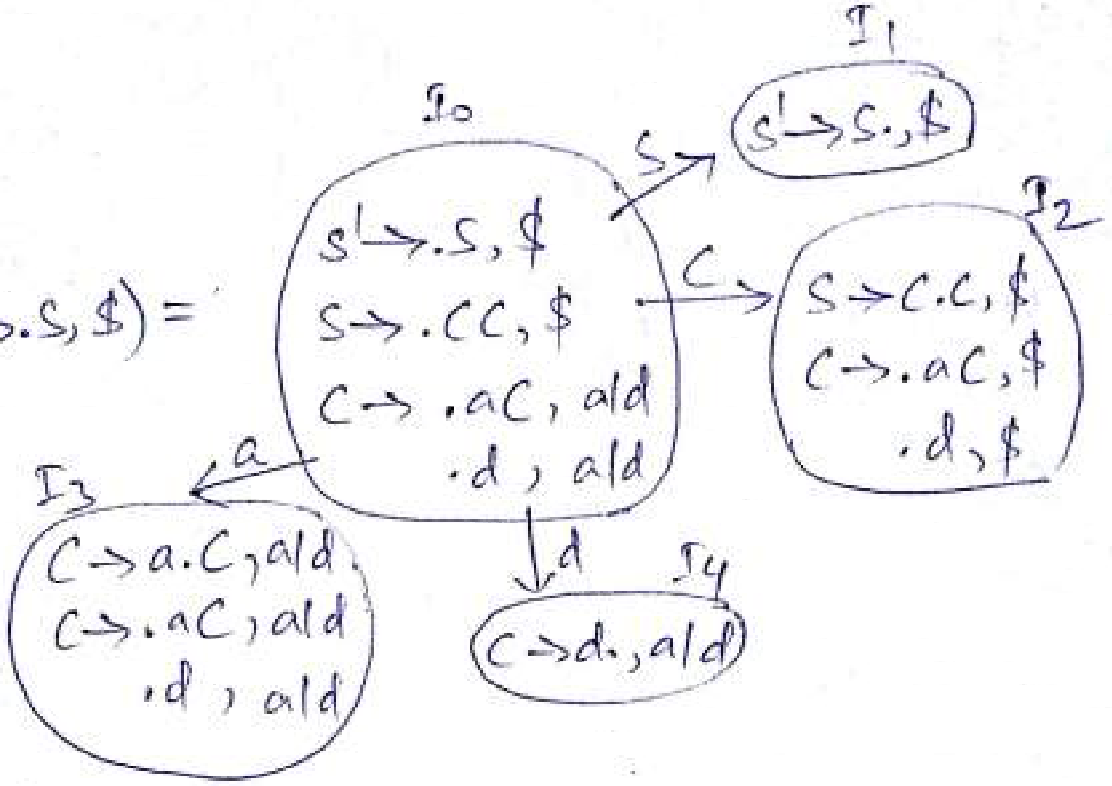
Soln The Augmented Grammar is

$S' \rightarrow S$

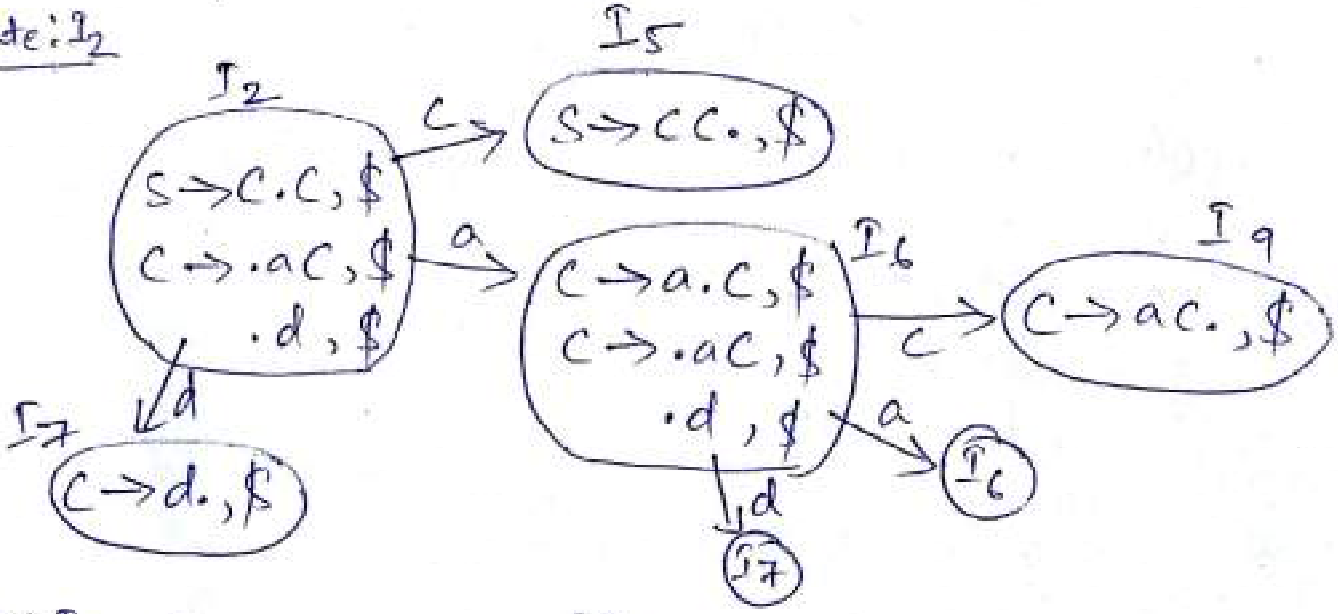
$S \rightarrow CC$

$C \rightarrow aC \mid d$

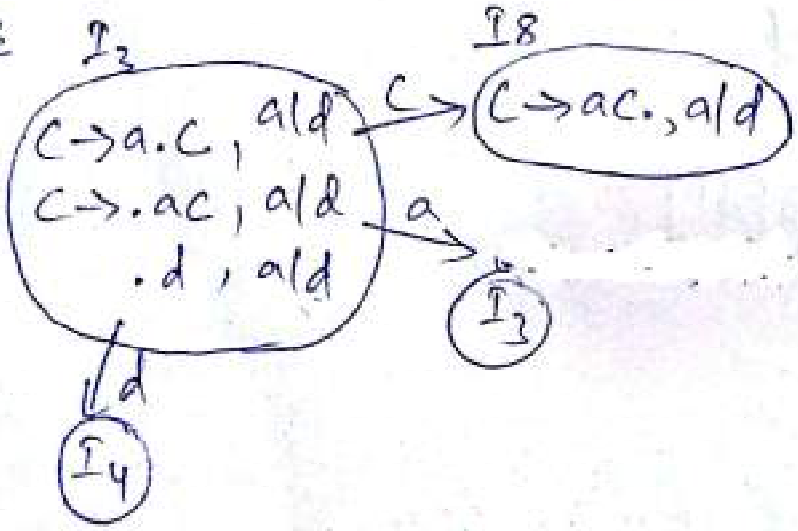
Closure $(S' \to .S, \$)$ =



State: I_2



State: I_3



	← Action →			← Goto →	
States	a	d	$	S	C
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		

fig. CLR(1) parsing table.

For LALR(1), we can merge the following states

I_{36} : C → a.C, a|d|$
C → .aC, a|d|$
.d, a|d|$

I_{47} : C → d., a|d|$

I_{89} : C → aC., a|d|$

	← Action →			← Goto →	
States	a	d	$	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		

fig LALR(1) parsing table.

CLR(1) Parsing Table Construction

Input:→ An Augmented grammar G'.

Output:→ The canonical LR parsing table functions Action and Goto for G'.

Method:→

1. Construct $C = \{I_0, I_1, \ldots I_n\}$ collection of sets of LR(1) items for G'.
2. State i of the parser is constructed from I_i. The parsing action for state i is determined as follows:→
   (a) If $[A \to \alpha . a\beta, b]$ is in I_i, and Goto $(I_i, a) = I_j$, then set Action $[i, a]$ to "shift j". Here a must be a terminal.
   (b) If $[A \to \alpha ., a]$ is in I_i, $A \neq S'$, then set Action $[i, a]$ to "reduce $A \to \alpha$".
   (c) If $[S' \to S., \$]$ is in I_i, then set Action $[i, \$]$ to "accept".
3. The goto transition for state i are constructed for all non-terminals A using the rule: If Goto$(I_i, A) = I_j$, then Goto $[i, A] = j$.
4. All entries not defined by rule (2) and (3) are made "error".
5. The initial state of the parser is the one constructed from the set of items containing $[S' \to .S, \$]$.

LALR(1) Parsing Table Construction

Step(1) :⇒ Obtain the canonical collection of sets of LR(1) items.

Step(2) :⇒ If more than one set of LR(1) items exists in the canonical collection obtained that have identical LR(1) items but which have different lookahead then combine these sets of LR(1) items to obtain a reduced collection of set of LR(1) items.

Step(3) :⇒ Construct the parsing table by using this reduced collection as follows :⇒

For action table :⇒ (C: canonical sets $(I_0, I_1, \ldots I_n)$)

(i) For each state I_i in C do for every terminal symbol 'a' do if goto$(I_i, a) = I_j$, then make action $[I_i, a] = S_j$

(ii) For every state I_i in C whose underlying sets of items contains an items of the form $[A \to \alpha., a]$ do make action $[I_i, a] = r_k$

where K is the number of production $A \to \alpha$ standing for reduce by $A \to \alpha$

(iii) Make action$[I_i, \$]$ = accept, if I_i containing an item $S' \to S., \$$.

For goto table :⇒

For every I_i in C_i do

for every nonterminal A do

if goto$(I_i, A) = I_j$, then

make goto$[I_i, A] = j$.

Q Construct CLR(1) and LALR(1) parsing table for the following grammar:

(1) $S \rightarrow AA$

$A \rightarrow aA \mid b$

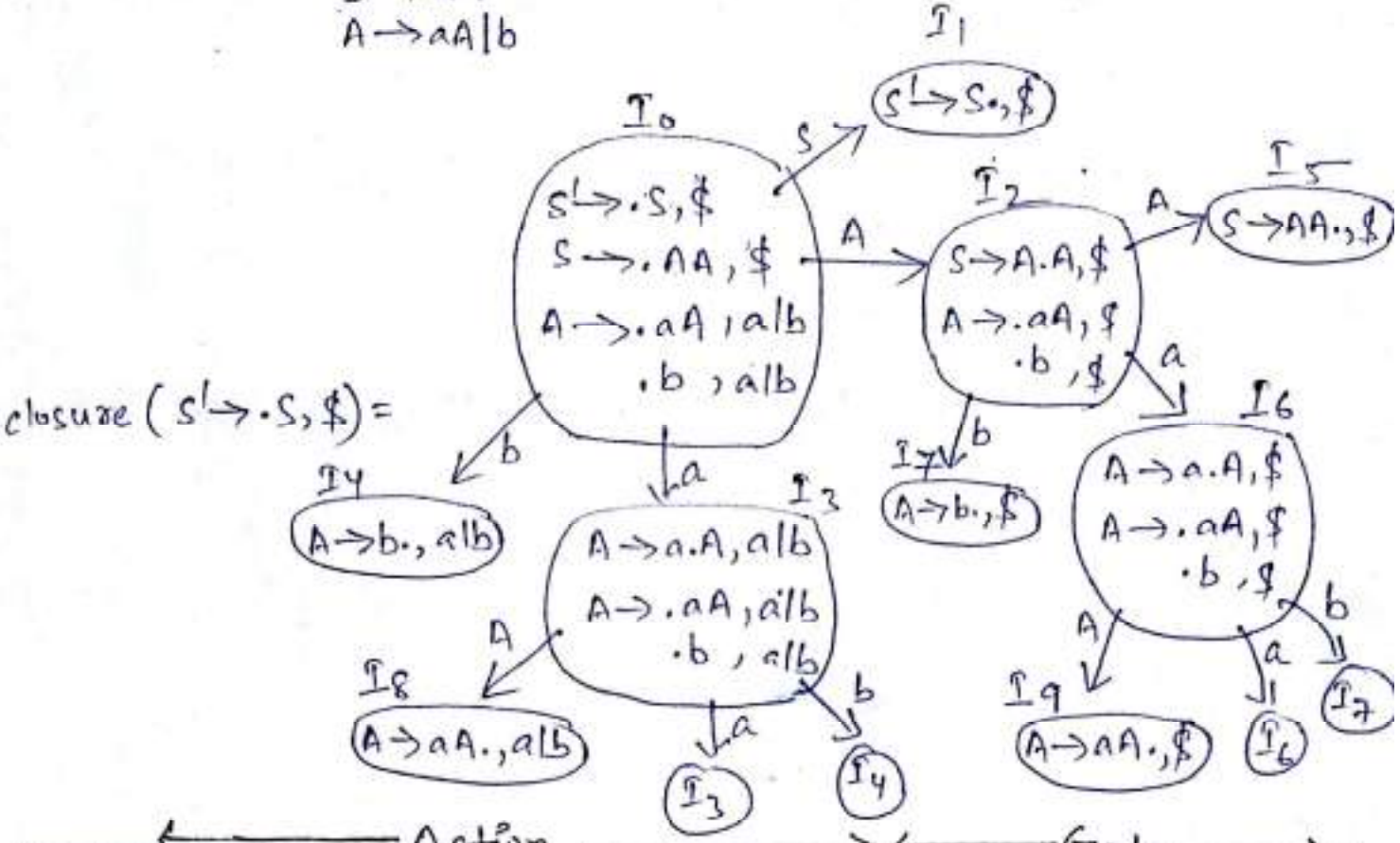
(2) (3)

Solution: The Augmented Grammar is

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA \mid b$



closure ($S' \rightarrow .S, \$$) =

	Action			Goto	
states	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		

Now, for LALR(1), we can merge the following states:

I_{36}: A → a.A, a|b|$
A → .aA, a|b|$
.b, a|b|$

I_{47} : A → b., a|b|$

I_{89}: A → aA., a|b|$

and all remaining states are same.

	← Action →			← Goto →	
states	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		

Q. Check the following Grammar is LALR(1) or not.

$S \xrightarrow{1} dA \mid \xrightarrow{2} aB$

$A \xrightarrow{3} bA \mid \xrightarrow{4} c$

$B \xrightarrow{5} bB \mid \xrightarrow{6} c$

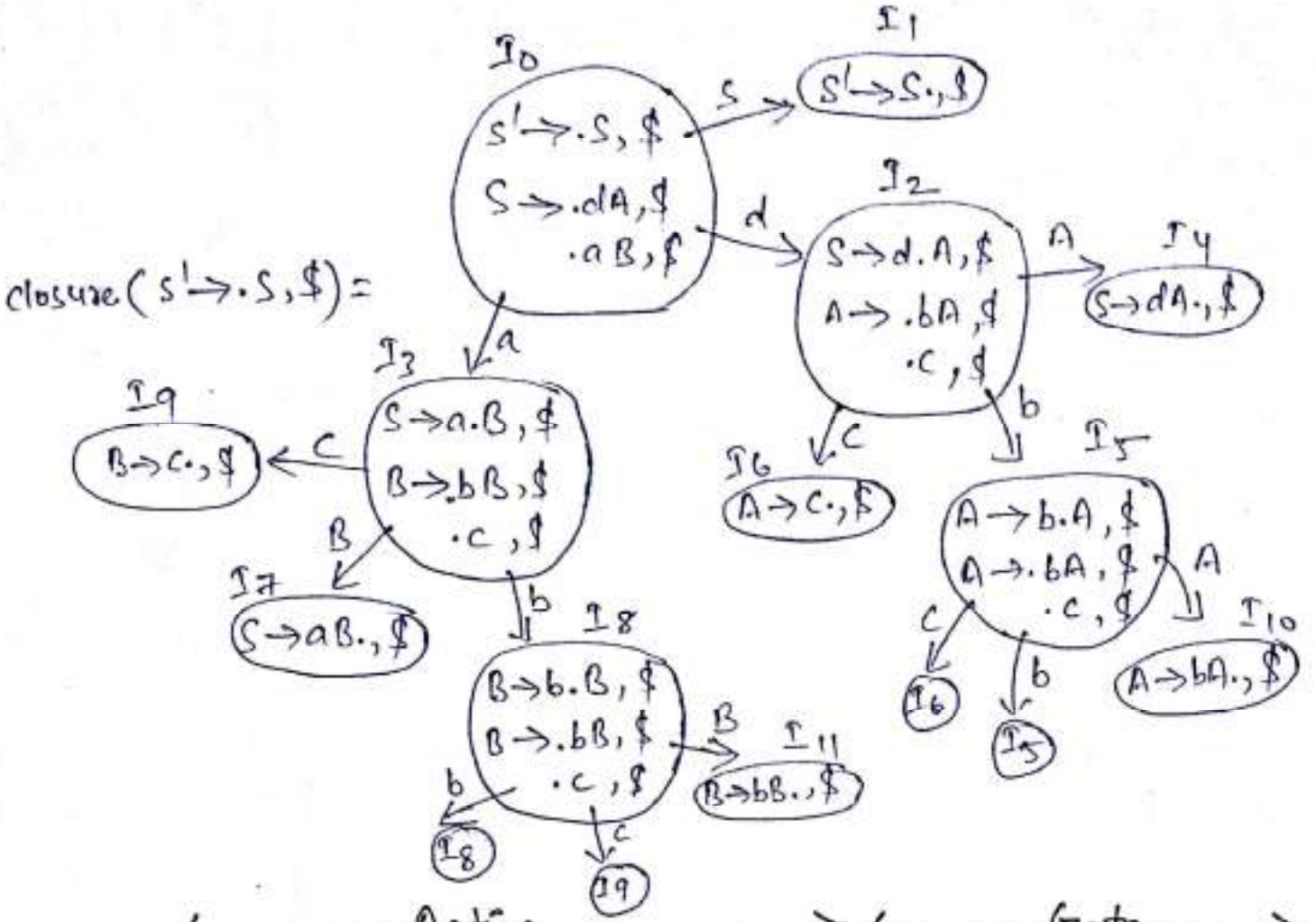
Solution : The Augmented Grammar is :

$S' \rightarrow S$

$S \rightarrow dA \mid aB$

$A \rightarrow bA \mid c$

$B \rightarrow bB \mid c$



	← Action →					← Goto →		
states	a	b	c	d	$	S	A	B
0	S_3			S_2		1		
1					accept		4	
2		S_5	S_6					
3		S_8	S_9					7
4					r_1			
5		S_5	S_6				10	
6					r_4		•	
7					r_2			
8		S_8	S_9					11
9					r_6			
10					r_3			
11					r_5			

No conflict exists in the table. ∴ Grammar is CLR(1).
It is also minimized CLR(1) also.

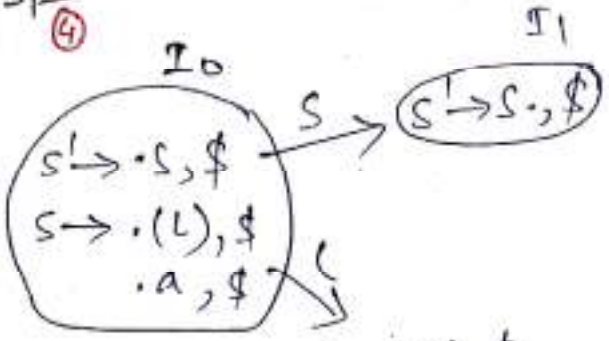
∴ It is LALR(1) grammar

Q. Construct the grammar is LALR(1) or not.

① ②
$S \to (L) \mid a$

$L \to L, S \mid S$
③ ④

Soln



Closure $(S' \to \cdot S, \$) =$

$S \to (\cdot L), \$$

$L \to \cdot L, S,)$

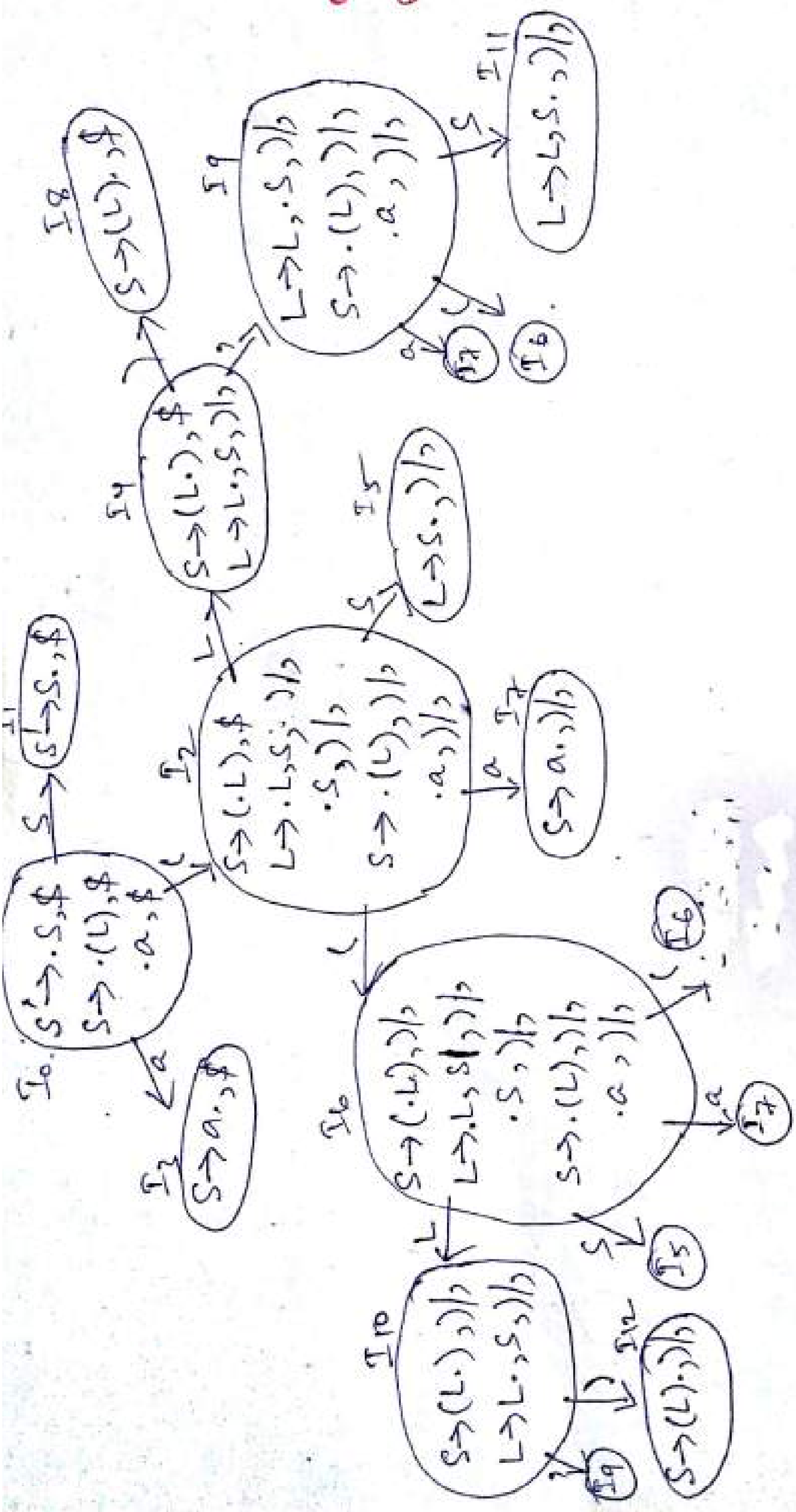
$\cdot S,)$

$S \to \cdot (L),)$

$\cdot a,)$

Q $S \to (L) \mid a$

$L \to L,S \mid S$



Q Consider the following Grammar:

(1) (2)
$E \to E+T \mid T$

(3) (4)
$T \to T*F \mid F$

(5)
$F \to id$

Solution:→ The Augmented Grammar is

$E' \to E$
$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to id$

closure($E' \to .E, \$$) =

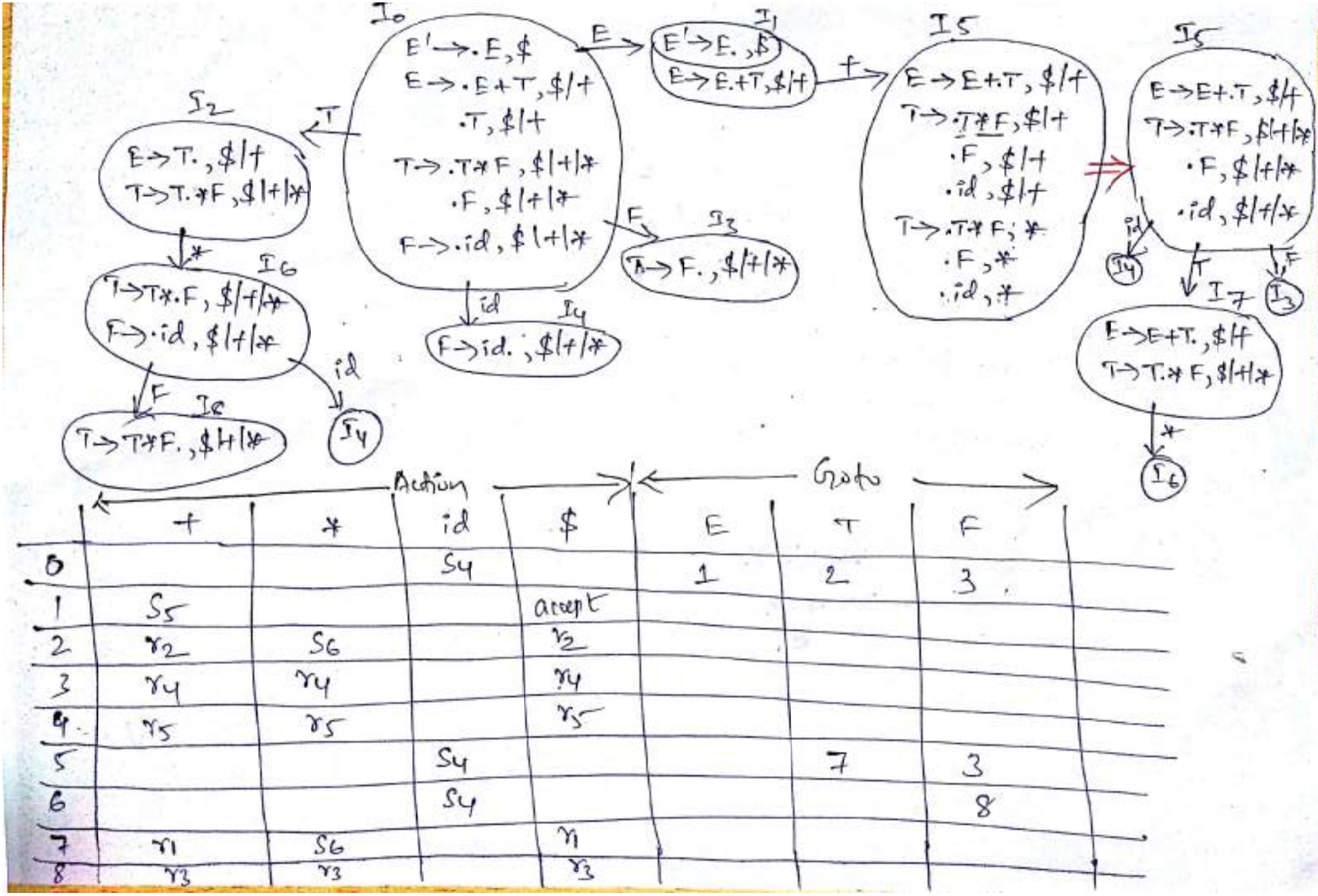
I0

$E' \to .E, \$$
$E \to .E+T, \$$
$\quad .T, \$$
$E \to .E+T, +$
$\quad .T, +$
$T \to .T*F, \$$
$\quad .F, \$$
$T \to .T*F, +$
$\quad .F, +$
$T \to .T*F, *$
$\quad .F, *$
$F \to .id, \$$
$F \to .id, +$
$F \to .id, *$

can be written as

I0

$E' \to .E, \$$
$E \to .E+T, \$/+$
$\quad .T, \$/+$
$T \to .T*F, \$/+/*$
$T \to .F, \$/+/*$
$F \to .id, \$/+/*$



	Action				Goto		
	+	*	id	$	E	T	F
0			S_4		1	2	3
1	S_5			accept			
2	r_2	S_6		r_2			
3	r_4	r_4		r_4			
4	r_5	r_5		r_5			
5			S_4			7	3
6			S_4				8
7	r_1	S_6		r_1			
8	r_3	r_3		r_3			