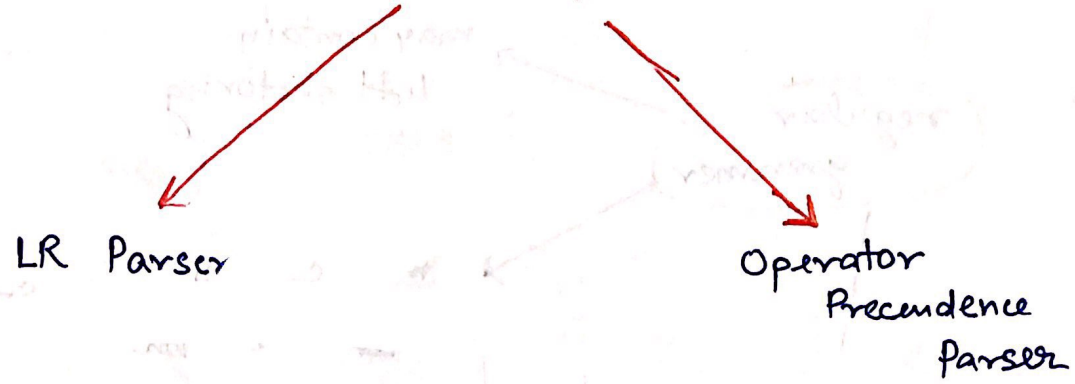
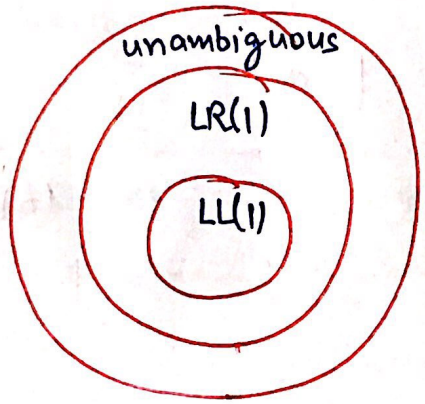
Bottom-up Parser



i) LR Parser :-

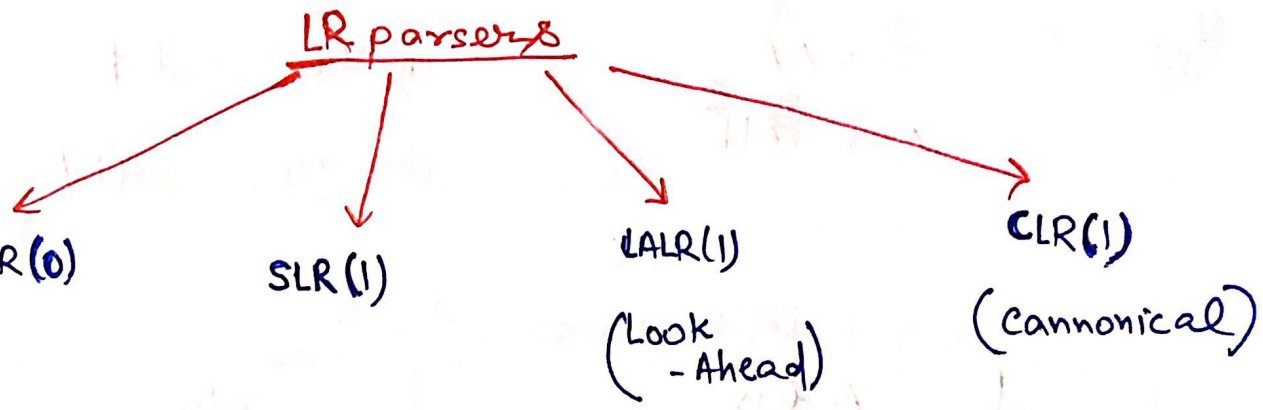
- only applicable for unambiguous grammer



"LR ⟶ unambiguous"

- Every LR is unambiguous, but not vice-versa
- Every LL is LR, but not vice versa

"LL ⟶ LR"



- For all LR parsers, parsing algorithm is same, but parsing table different

⇉ LR parsing algorithm :-

Let X be the state number on top of stack, a - look-ahead symbol :

1. if action[X, a] = S_i then shift a and i, and inc I/p pointer

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

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

3. If action[X, a] = blank then parsing error

4. If action[X, a] = acc then successful completion of parsing

eg:

$S \xrightarrow{①} AA$

$A \xrightarrow{②} aA \mid \xrightarrow{③} b$

(i) I/p: abab $

(ii) I/p: aabb $

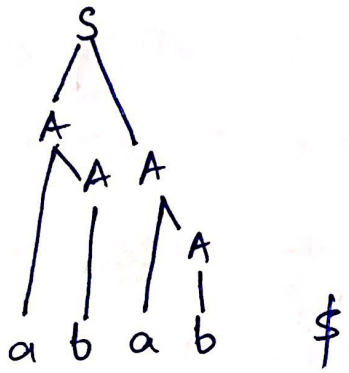
LR(0) parsing table

	Action			Goto	
	a	b	$	S	A
0	s_3	s_4		1	2
1			acc		
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		

(i)

I/p	stack	Reductions
~~a~~bab $	0 → 0a3	s_3
~~b~~ab $	0a~~3~~ → 0a3b4	s_4
~~ab~~ $	0a3b4 → 0a3A6	$A \to b$ r_3
	0a3A6 → 0A2	$A \to aA$ r_2
	0A2 → 0A2a3	

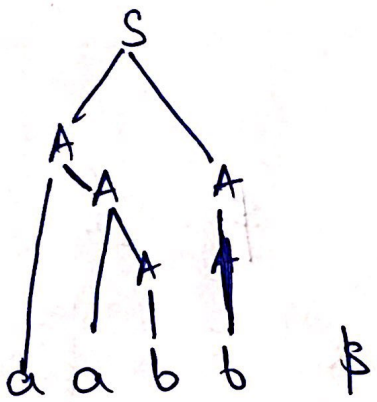
I/p	stack	Reductions
~~b~~ $	0A2a3 → 0A2a3b4	s_4
$	0A2a3b4 → 0A2a3A6	$A \to b$ r_3
	0A2a3A6 → 0A2A5	$A \to aA$ r_2
	0A2A5 → 0S1	$S \to AA$ r_1
	0S1 →	[ACCEPT]



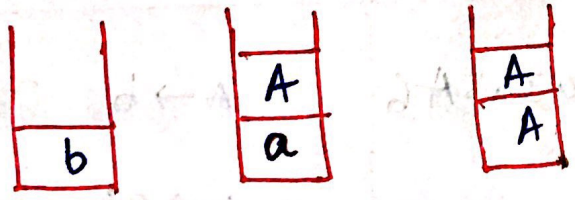
(ii)

I/p	stack	Reductions
~~a~~abb $	0 → 0a3	s_3
~~a~~bb $	0a3 → 0a3a3	s_3
~~b~~b $	0a3a3 → 0a3a3b4	s_4
~~b~~ $	0a3a3b4 → 0a3a3A6	$A \to b$ r_3
	0a3a3A6 → 0a3A6	$A \to aA$ r_2
	0a3A6 → 0A2	$A \to aA$ r_2
	0A2 → 0A2b4	s_4

I/P	stack	Reduction
$	0A2b4 → 0A2A5	A → b r_3
	0A2A5 → 0S1	S → AA r_1
	0S1	ACCEPT



- **Viable Prefix :-** The set of all Lprefix present in Bottom of Parser stack.



(iii)

I/p	Stack	Reduction
~~a~~aab $	0 → 0a3	s_3
~~a~~ab $	0a3 → 0a3a3	s_3
~~a~~b $	0a3a3 → 0a3a3a3	s_3
~~b~~ $	0a3a3a3 → 0a3a3a3b4	s_4
$ ↑	0a3a3a3b4 → 0a3a3a3A6	A→b r_3
	0a3a3a3A6 → 0a3a3A6	A→aA r_2
	0a3a3A6 → 0a3A6	A→aA r_2
	0a3A6 → 0A2	A→aA r_2
	0A2 ↑	Error Blank

Parsing

Error

* Language is not member of Grammer

LR(0) parsing table construction algo:-

To construct LR(0) parsing table we need 2 func^{n}:

- closure()
- goto()

Augmented grammer

G: $S \to AA$
$A \to aA|b$

G': $(S' \to S)$ — Augmented production
$S \to AA$
$A \to aA|b$

$S \to abc$

$S \to \cdot abc$
$S \to a \cdot bc$
$S \to ab \cdot c$
$S \to abc \cdot$

} LR(0) items

$(S \to abc \cdot)$ — complete LR(0) item or final or reduced

(i) **closure():-** closure (I) denotes the production we can derive by scanning $\in$ symbol.

closure (I) : (i) ADD I
(ii) If $A \to B \cdot CDE$ & $C \to EFG$ then add $C \to \cdot EFG$ to closure(I)
(iii) Repeat (ii) for every newly added LR(0) items

eg:- $S' \to S$
$S \to AA$
$A \to aA|b$

closure $(S \to \cdot AA) = \{ S \to \cdot AA,\ A \to \cdot aA,\ A \to \cdot b \}$

if $\cdot V$ (Variable) $\longrightarrow$ Expand with productions of V

if $\cdot t$ (terminal) $\longrightarrow$ stop

(ii) GOTO( ) :

GOTO(I, x) = (i) Add I by moving • after x

(ii) Add closure ((i))

eg:- GOTO(S → .AA , A) = { S → A.A, A → .aA, A → .b }

GOTO (A → .aA , a) = { A → a.A, A → .aA, A → .b }

Algorithm:-

(i) I_0 = closure (Augmented LR(0) item)

(ii) Using I_0, construct DFA

(iii) Convert DFA into , LR(0) parsing table.

Q. Construct LR(0) parsing table for the following grammer :

S → AA
A → aA | b

Augmented Grammer :-

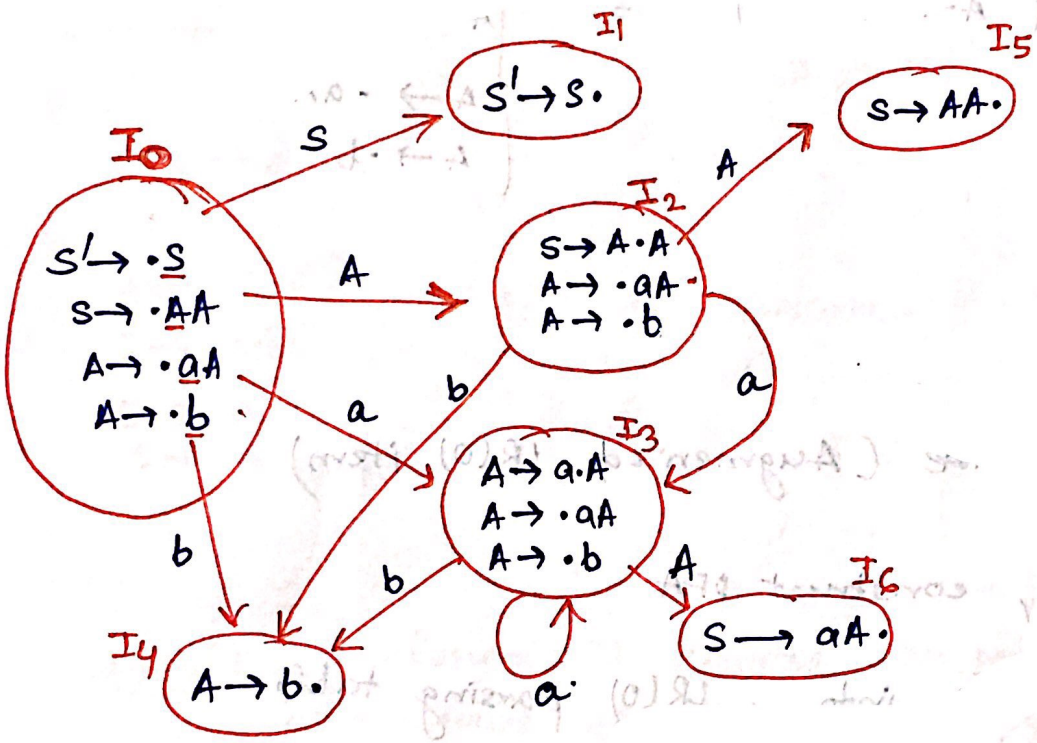
$S' \rightarrow S$

$S \rightarrow AA$ ①

$A \rightarrow aA$ ② $| \; b$ ③

$(S' \rightarrow \cdot S)$ Augmented LR(0) item

I_0 = closure (Augmented LR(0) item)



LR(0) parsing table

rows

(no. of states in DFA)

columns

(Action → terminals + $) + (Goto → variables)

	ACTION			GOTO	
	a	b	$	S	A
0	S_3	S_4		1	2
1			Acc		
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		

- Complete / final / reduced LR(0) items are blindly fill in all action columns.

r_1 $S \rightarrow AA\cdot$

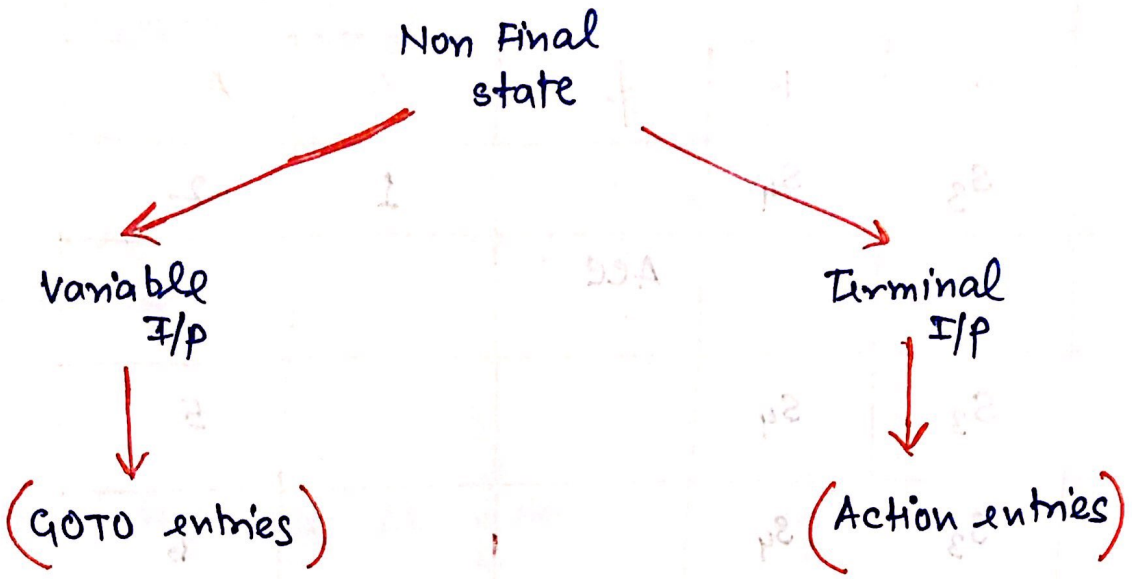
r_2 $S \rightarrow aA\cdot$

r_3 $A \rightarrow b\cdot$

$(S' \rightarrow S\cdot)$ used for "acc"

- Non-Final LR(0) items → shift op^n in table (entries)

Final LR(0) items → Reduce op^n (entries) in table



- Augmented production ($S' \rightarrow S$) is added to check whether S is over or not.

($S' \rightarrow S.$) $\longrightarrow$ it means S over i.e string over,

so if $ comes then parsing is successful & string is accepted.

Since in Grammer, there is no I/p for S to check for S so we add augmented production explicitly.

Conflicts in LR(0) parsing table :-

(i) RR conflict

(ii) SR conflict

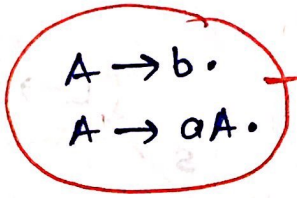
If state J having problem then, J should satisfy following condition:

(i) J should contain minimum 2 production

(ii) Atleast one Reduce in the J

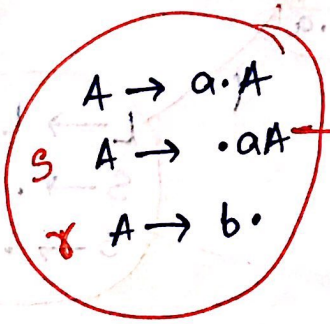
R	R
S	R

RR conflict :-



→ r_2/r_3 (multiple reduce choice in table cell)

SR conflict :-



→ S/r (contains both final & non final LR(0) items)

NOTE:- If LR(0) parsing table doesn't contains any conflicts than grammar is LR(0)

OR

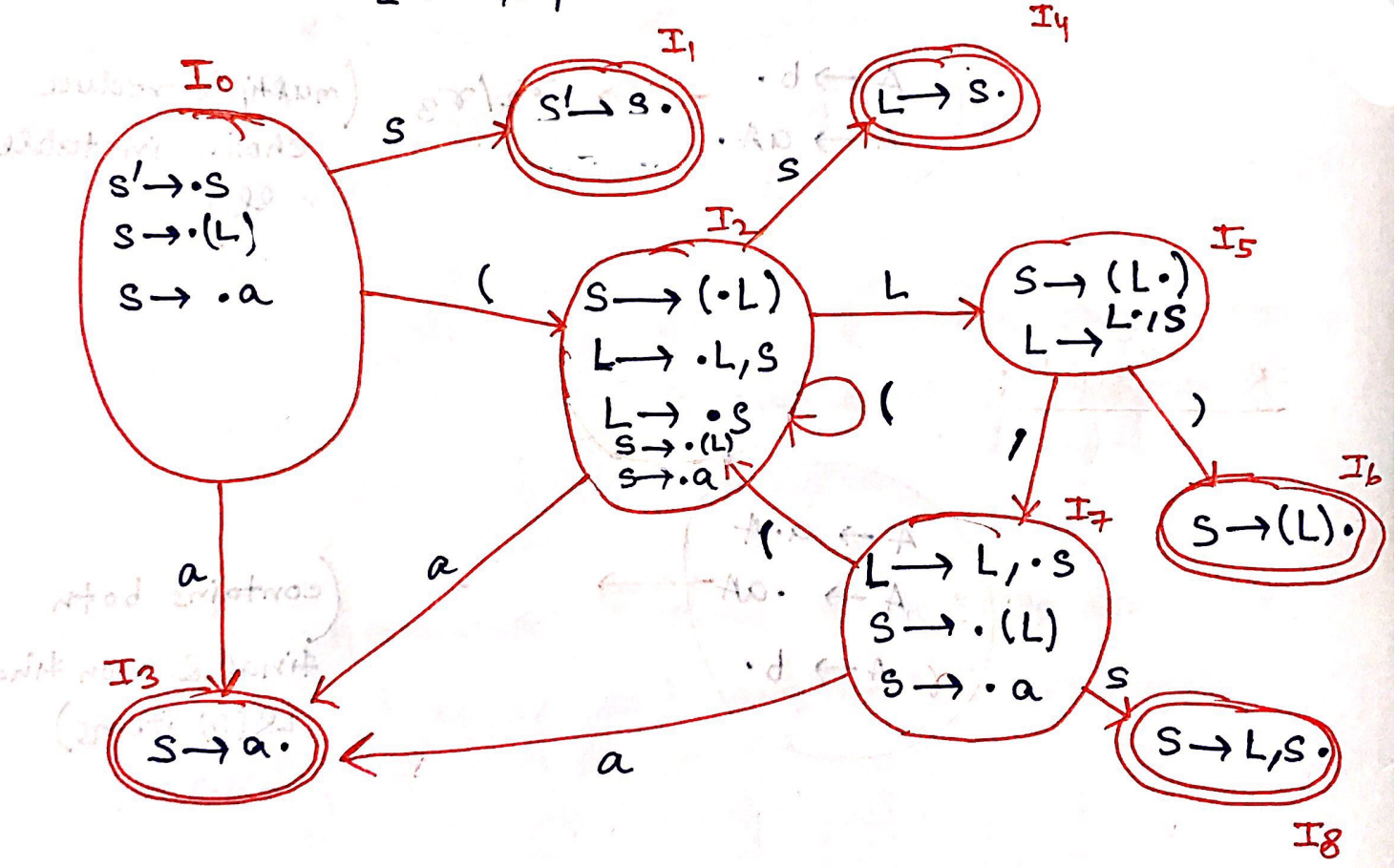
In DFA check every state if it contains both final ~~or~~ & non final LR(0) items or more than 1 LR(0) items then Grammer is not LR(0)

Q. Construct LR(0) parsing table for the following Grammer G:

$$S \rightarrow (L) \mid a$$
$$L \rightarrow L,S \mid S$$

Augmented Grammer G':

$$S' \rightarrow S$$
$$S \rightarrow (L) \overset{①}{\mid} a \overset{②}{}$$
$$L \rightarrow L,S \overset{③}{\mid} S \overset{④}{}$$



	Action					Goto	
	a	(	)	,	$	S	L
0	s_3	s_2				1	
1					acc		
2	s_3	s_2				4	5
3	r_2	r_2	r_2	r_2	r_2		
4	r_4	r_4	r_4	r_4	r_4		
5			s_6	s_7			
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		

Q: Check the following grammes is SLR(1) or not:

Q:

$E \rightarrow T+E \mid T$

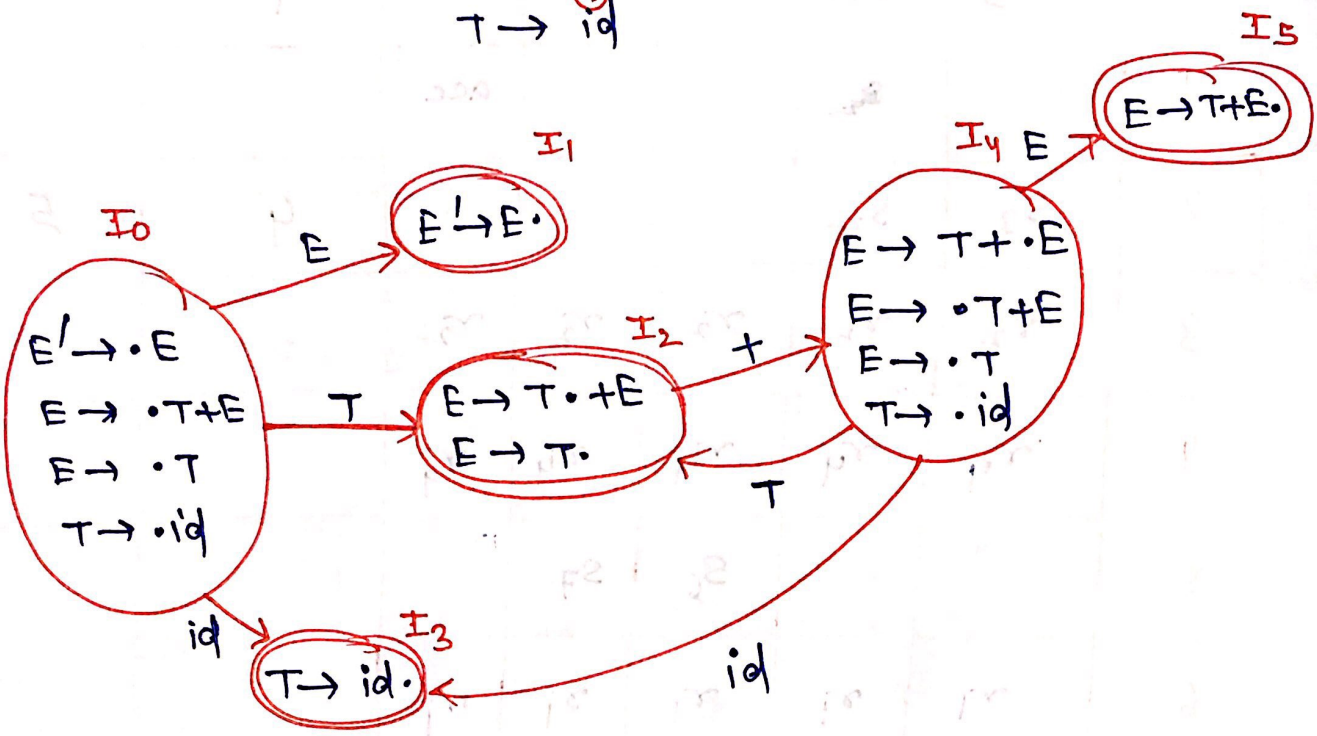
$T \rightarrow id$

Augmentated Grammer:

$E' \rightarrow E$

$E \rightarrow T + E$ ① | T ②

$T \rightarrow id$ ③



As we can see in I_2 : $E \rightarrow T \cdot +E$ (S_4)

$E \rightarrow T \cdot$ (r_2)

SR conflict

	Action			Goto	
	+	id	$	E	T
I_2	S_4 / r_2	r_2	r_2		

Inadequatic state (state with conflict)

⇒ SLR(1) parsing table :-

- Shift entries are same
- Reduce entries are changed (fill reduce entries only place of Follow of handler)

eg:- $(E \rightarrow T\cdot)$

reduce entries at Follow(E) i.e $

	Action			Goto	
	+	id	$	E	T
I_2	S_4		r_4		

No conflict so Grammer is SLR(1)

LL(1) ✗

LR(0) ✗

SLR(1) ✓

LALR(1) ✓

CLR(1) ✓

Q. $E \rightarrow E+T \mid T$ check SLR(1) or not?

$T \rightarrow TF \mid F$

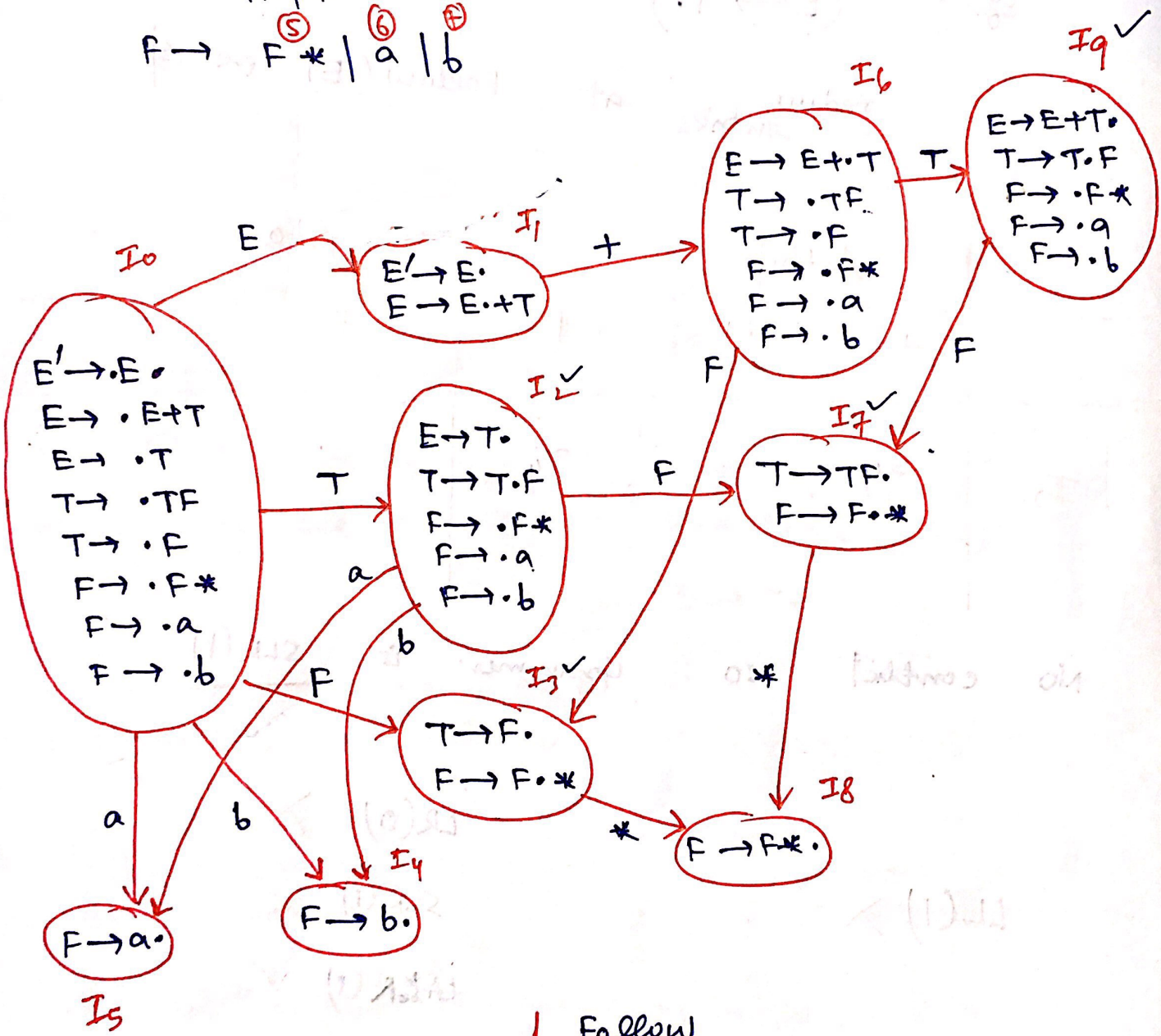
$F \rightarrow F* \mid a \mid b$

$E' \rightarrow E$

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

$T \rightarrow TF$ (3) $\mid F$ (4)

$F \rightarrow F*$ (5) $\mid a$ (6) $\mid b$ (7)



	Follow
E	$, +
T	$, +, a, b
F	$, +, a, b, *

states which may have (Inadequatic in LR(0)) conflict = ~~I_1~~, I_2, I_3, I_7, I_9

As I_1 contains augmentation production which is used for accepting string not for reducing so it has no conflict.

SLR(1)	Action					goto		
	+	*	a	b	$	E	T	F
2	r_2		s_5	s_4	r_2			
3	r_4	S8	r_4	r_4	r_4			
7	r_3	S8	r_3	r_3	r_3			
9	r_1		s_5	s_4	r_1			

[0 Inadequate states in SLR(1)]

As we can see there are no conflicts so, grammer is SLR(1)

NOTE:- Only check for Action (terminals i/p) , as they can only produce conflicts.

LL(1) ✗

LR(0) ✗

SLR(1) ✓

LALR(1) ✓

CLR(1) ✓

Q: Check the following grammer is SLR(1) or not?

$S \rightarrow \overset{①}{AaAb} \mid \overset{②}{BbBa}$

$A \rightarrow \overset{③}{\epsilon}$

$B \rightarrow \overset{④}{\epsilon}$

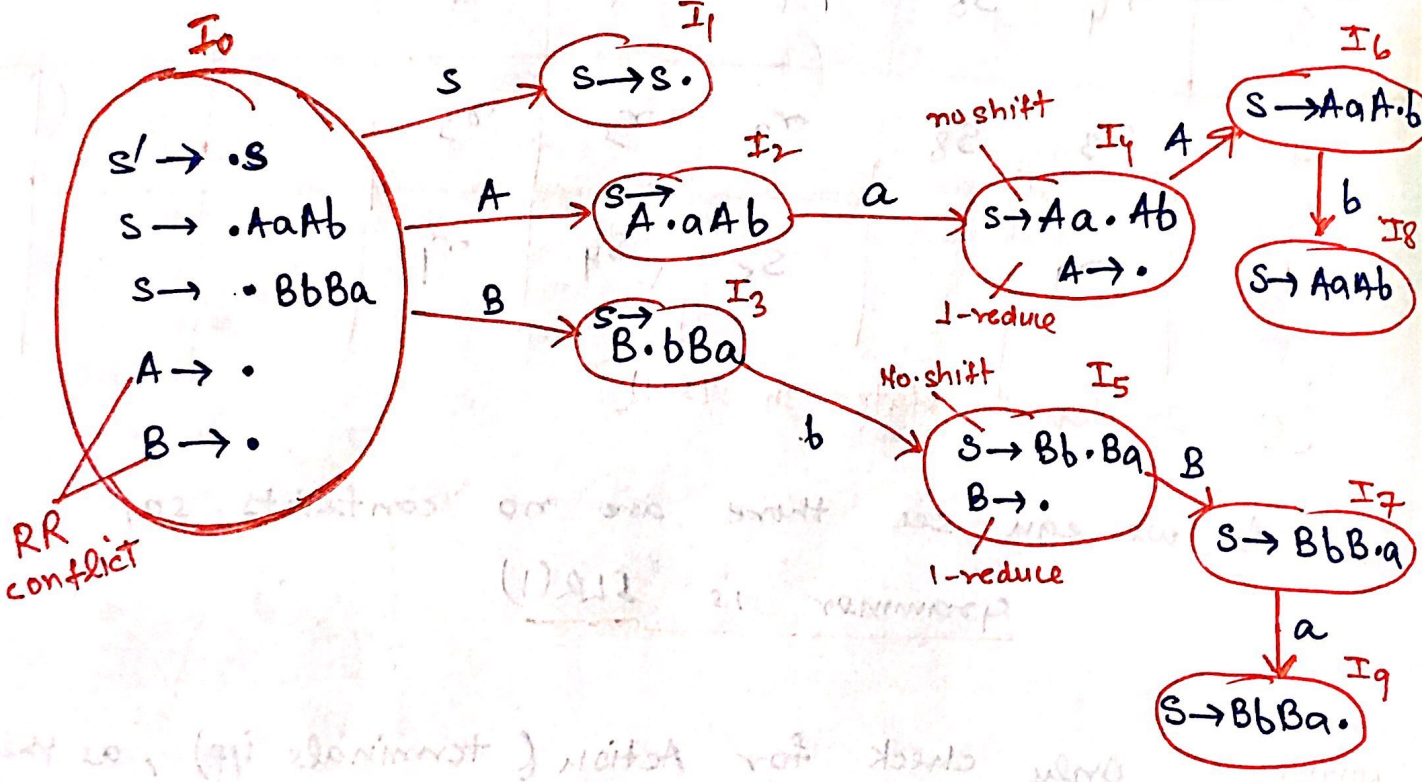
	FOLLOW
S	$
A	a, b
B	a, b

$S' \rightarrow S$

$A \rightarrow \cdot\epsilon \equiv A \rightarrow \cdot$

$B \rightarrow \cdot\epsilon \equiv B \rightarrow \cdot$

reduce/Final items



SLR(1)

	Action			Goto		
	a	b	$	S	A	B
I_0	r_3/r_4	r_3/r_4				
I_4	r_3	r_3			6	
I_5	r_4	r_4				7

So, grammer is not SLR(1)

no. of Inadequate states = { I_0 } = one

no. of conflicts → RR = 2

→ SR = 0

LR(0) item → LR(0) parser, SLR(1) parser

LR(1) item → LALR(1) parser, CLR(1) parser

(LR(1) item = LR(0) item + Look ahead symbol)

eg:-

$S \rightarrow \cdot abc$, d/e

($S \rightarrow \cdot abc$: LR(0) item; d/e : Look Ahead)

Closure & Goto func^n for LR(1) items :-

$S \rightarrow AA$

$A \rightarrow aA \mid b$

$S' \rightarrow S$

(i) closure ($S' \rightarrow \cdot S$, $\$$)

$$= \begin{cases} S' \rightarrow \cdot S \;,\; \$ \\ S \rightarrow \cdot AA \;,\; \$ \\ A \rightarrow \cdot aA \;,\; a/b \\ \quad\quad \cdot b \;,\; a/b \end{cases}$$

LA = first($\$$)

LA = first(A, $\$$)
= first(A) = a/b

, is used as seperator

(ii) closure ($S \rightarrow \cdot AA$, c/d)

$S \rightarrow \cdot AA$, c/d
$A \rightarrow \cdot aA$, a/b
$\quad\quad \cdot b$, a/b

LA = First(A , c/d)

So,

closure (I):

(i) Add I

(ii) $A \rightarrow B \cdot CDE$, a/b is I and $C \rightarrow EFG$ is in G then add $C \rightarrow \cdot EFG$, FIRST(DE, a/b) to closure

(iii) Repeat (ii) step.

→ Goto (I , i/p) :

Goto (I, X) = (i) Add I by moving . after X

(ii) closure (i))

eg:-

Goto (S → · aA , c|d, a)

=

$$\begin{cases} S \to a \cdot A \,(\, c|d) \\ A \to \cdot aA \,,\, c|d \\ \quad\quad \cdot a \,,\, c|d \end{cases}$$

Q. Construct CLR(1) , LALR(1) parsing table for follow-ing grammer .

G:

S → AA

A → aA|b

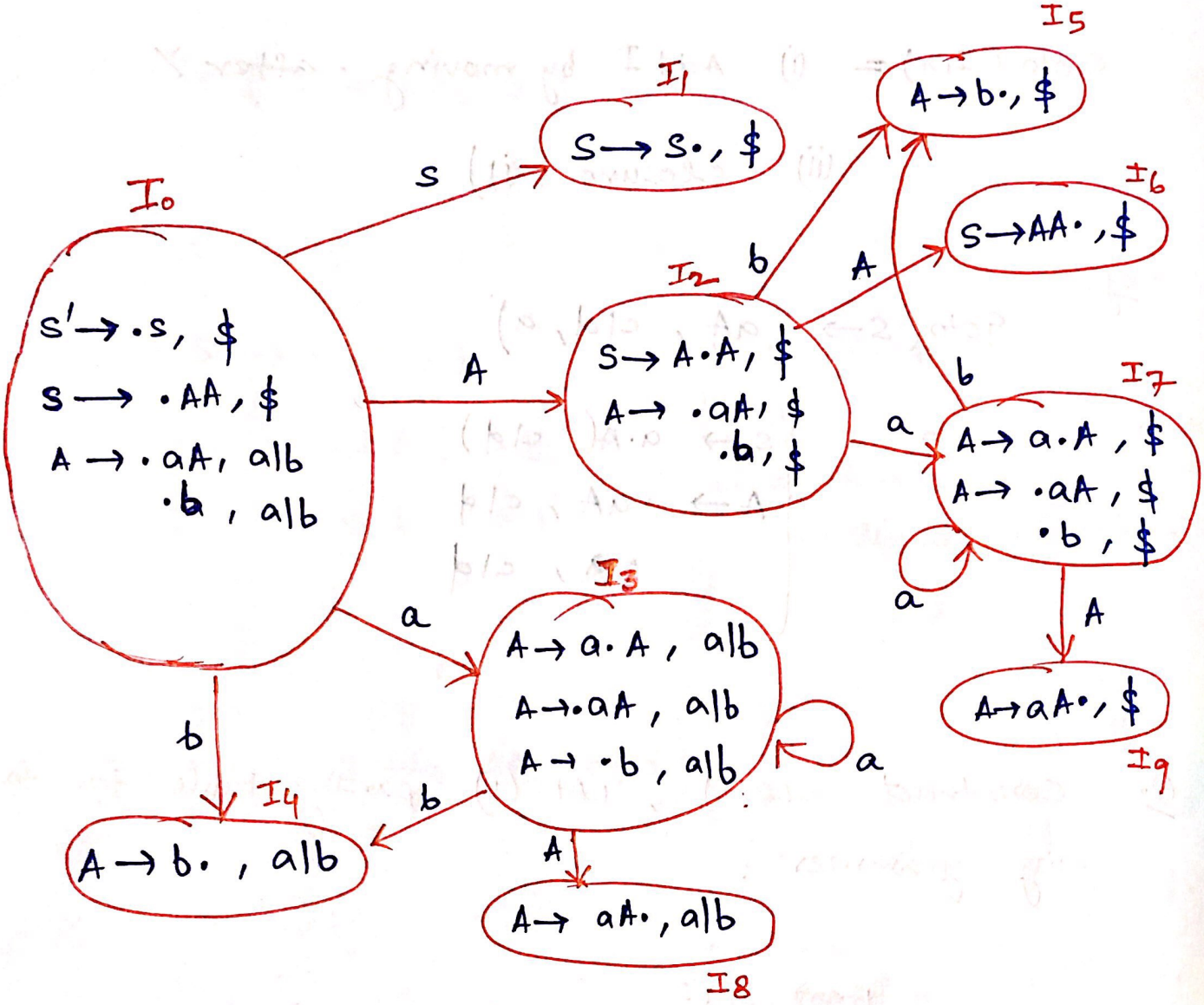
Augmented G' :

S' → S

S → AA ①

A → aA ② | b ③

$$\begin{cases} S' \to \cdot S \,,\, \$ \\ S \to \cdot AA \,,\, \$ \\ A \to \cdot aA \,,\, a|b \\ \quad\quad \cdot b \,,\, a|b \end{cases}$$



- We can see in I_3 and I_7 LR(0) items are same but LA is different that's why LR(1) items differ and hence they belongs to different states.

- No. of states increased ⟶ power also increased.

$$\left[\left(N_{LR(0)} \overset{\text{no. of states}}{=} N_{BLR(1)}\right) \leq N_{CLR(1)}\right]$$

$$= N_{LALR(1)}$$

CLR(1)	ACTION			GOTO	
	a	b	$	S	A
0	s_3	s_4		1	2
1			Acc.		
2	s_7	s_5			6
3	s_3	s_4			8
4	r_3	r_3			
5			r_3		
6			r_1		
7	s_7	s_5			9
8	r_2	r_2			
9			r_2		

- SLR(1) → follow (global neighbours)

- CLR(1) → Look Ahead (Local neighbours)

Since CLR(1) focused on Local neighbours so it can differentiate reduce entries more accurately, hence less conflicts.

- So above table has no conflicts so grammer is CLR(1)

- CLR(1) parser is more powerful comparing other parser because it is taking care of upcoming symbol every where.

- Only drawback with CLR(1) is more no. of state so cost also more.

- To decrease cost of CLR(1) we apply minimization algorithm (If two states differ by only look-ahead make it as single state)

- Minimized CLR(1) also known as LALR(1)

LALR parsing table :-

- DO Grouping of L states whose LR(1) items differed only in Look ahead.

row of

so, (I_3, I_7) (I_8, I_9) (I_4, I_5)

I_{37} I_{89} I_{45}

LALR(1)	ACTION			GOTO	
	a	b	$	S	A
0	S_{37}	S_{45}		1	2
1			ACC		
2	S_{37}	S_{45}			6
37	S_{37}	S_{45}			89
45	r_3	r_3	r_3		
6			r_1		
89	r_2	r_2	r_2		

- LALR may contains conflicts during grouping.
- Since no. of states are less in LALR than CLR so

$$(\text{Power}(\text{LALR}) < \text{Power}(\text{CLR}))$$

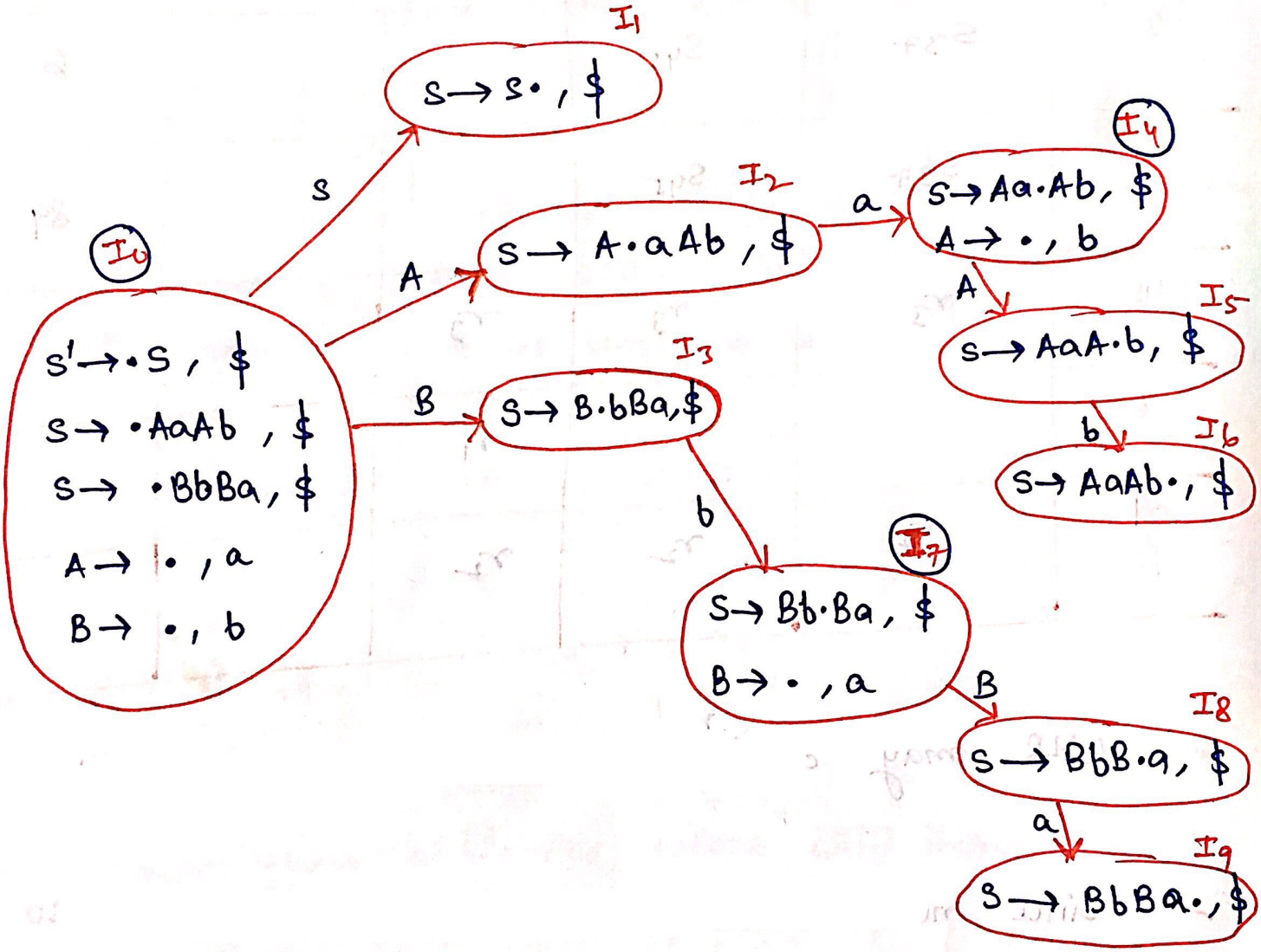
Q. Check the following grammer is LALR(1) or not ?

$S \rightarrow \underset{①}{AaAb} \mid \underset{②}{BbBa}$

$A \rightarrow \underset{③}{\epsilon}$

$B \rightarrow \underset{④}{\epsilon}$

$S' \rightarrow \cdot S \ , \ \$$



- Conflicts can only happen in those states only which has:

(i) more than one LR(1) items.

(ii) Atleast one reduce LR(1) item

So, I_0, I_4, I_7 are prone for conflicts

CLR(1)	Action		
	a	b	$
I_0	r_3	r_4	
I_4		r_3	
I_7	r_4		

No conflicts so it is CLR(1)

SLR(1)	Action		
	a	b	$
I_0	r_3/r_4	r_3/r_4	
I_4	r_3	r_3	
I_7	r_4	r_4	

conflicts so it is not SLR(1)

Since as we can see there are no common LR(1) items in pairs of states so it is already a minimized CLR(1) i.e LALR(1)

LL(1) ✓

LR(0) ✗
SLR(1) ✗
CLR(1) ✓
LALR(1) ✓

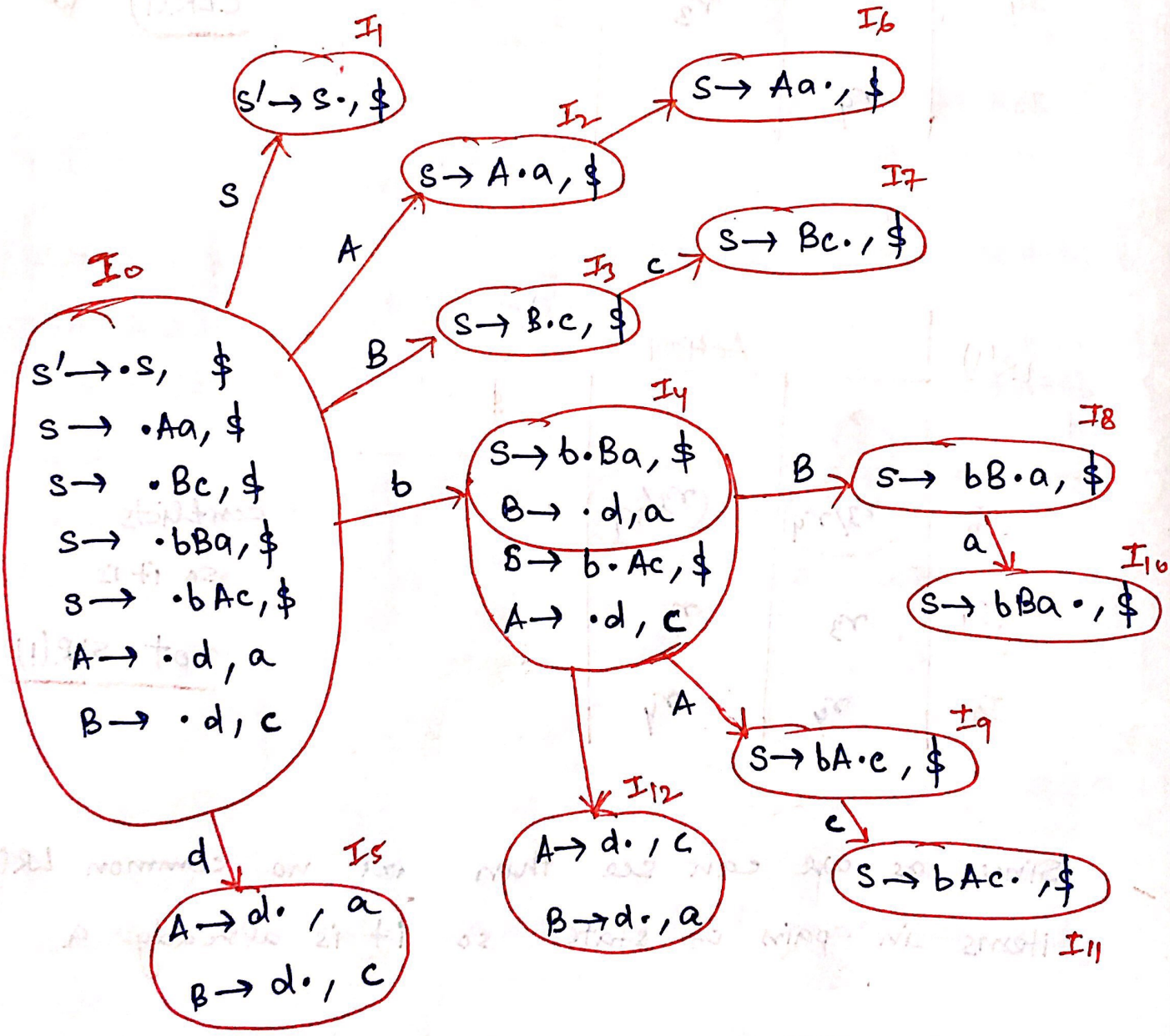
Q. Check the following grammer is LALR(1) or not?

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

$A \rightarrow d$ ①

$B \rightarrow d$ ②

since, $S \rightarrow bBa \mid bAc$ (left factoring) so not LL(1)



States prone for conflicts : (I_5 , I_{12})

LR(0)	a	b	c	d	$
I_5	r_1/r_2	r_1/r_2	r_1/r_2	r_1/r_2	r_1/r_2
I_{12}	r_1/r_2	r_1/r_2	r_1/r_2	r_1/r_2	r_1/r_2

Not LR(0)

SLR(1)	a	b	c	d	$
I_5	r_1/r_2		r_1/r_2		
I_{12}	r_1/r_2		r_1/r_2		

Not SLR(1)

CLR(1)	a	b	c	d	$
I_5	r_1		r_2		
I_{12}	r_2		r_1		

CLR(1)

LALR(1)	a	b	c	d	$
$I_{5,12}$	r_1/r_2		r_1/r_2		

Not LALR(1)

NOTE:-

- If CLR(1) don't have R-R conflict then LALR(1) may or not contain R-R conflict

- If CLR(1) don't have S-R conflict then LALR(1) don't have SR conflict.

CLR(1) $\longleftrightarrow$ LALR(1)

SR - conflict SR - conflict

Q. Consider the following grammer :

G: $S \rightarrow (S) \mid a$

LR(0) — n_1 states

SLR(1) — n_2 "

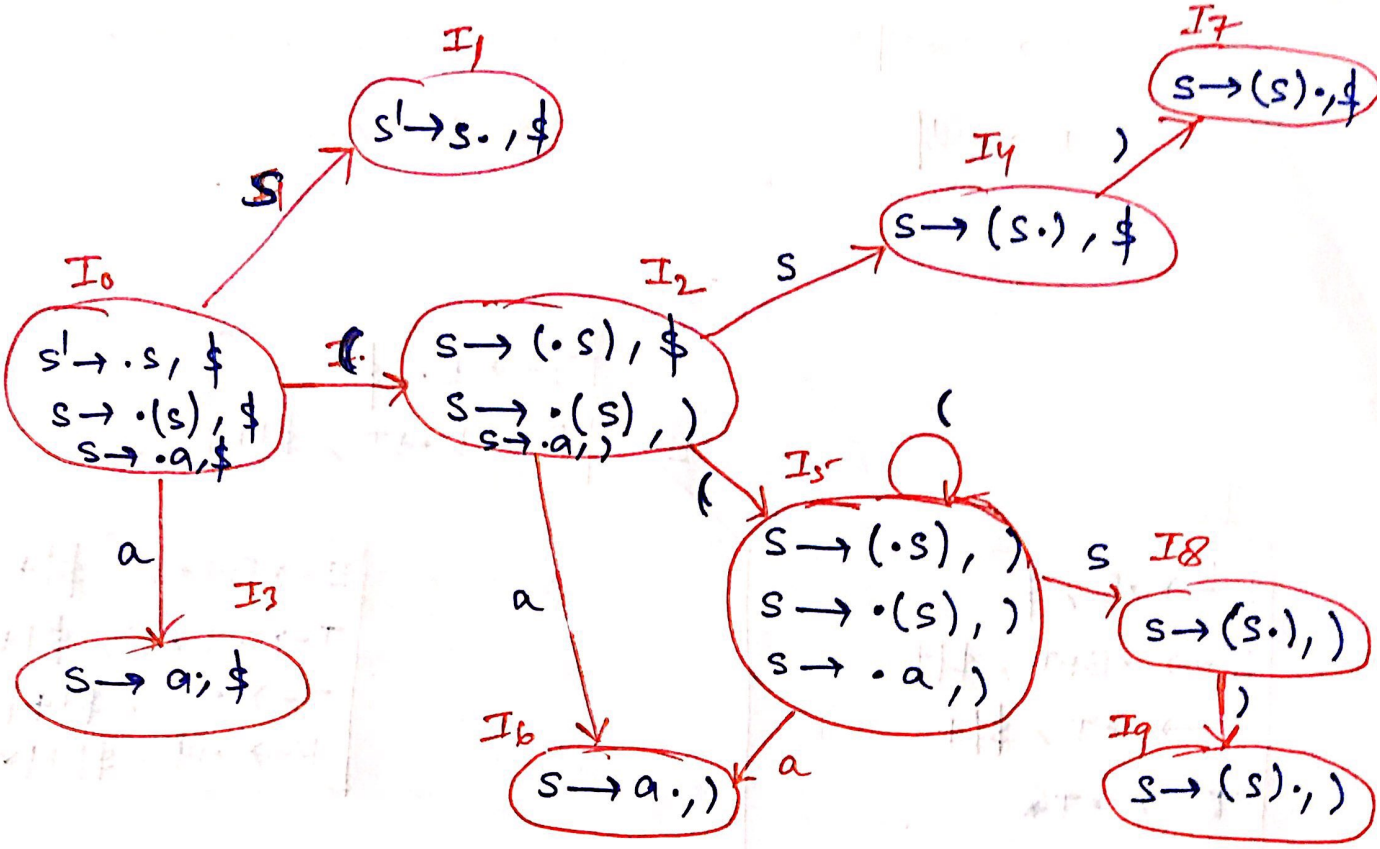
LALR(1) — n_3 "

CLR(1) — n_4 "

Find rel^n b/w n_1, n_2, n_3, n_4

$S' \rightarrow \cdot S \;,\; \$$

$S \rightarrow \cdot (S) \;,\; \$$



Since we know rel^n:

$$(n_1 = n_2 = n_3) \leq n_4$$

↑ dilemma

so, if n_4 has some state extra then $n_4 > n_3$

and in DFA I_3, I_6 are same but LAS is different

so,

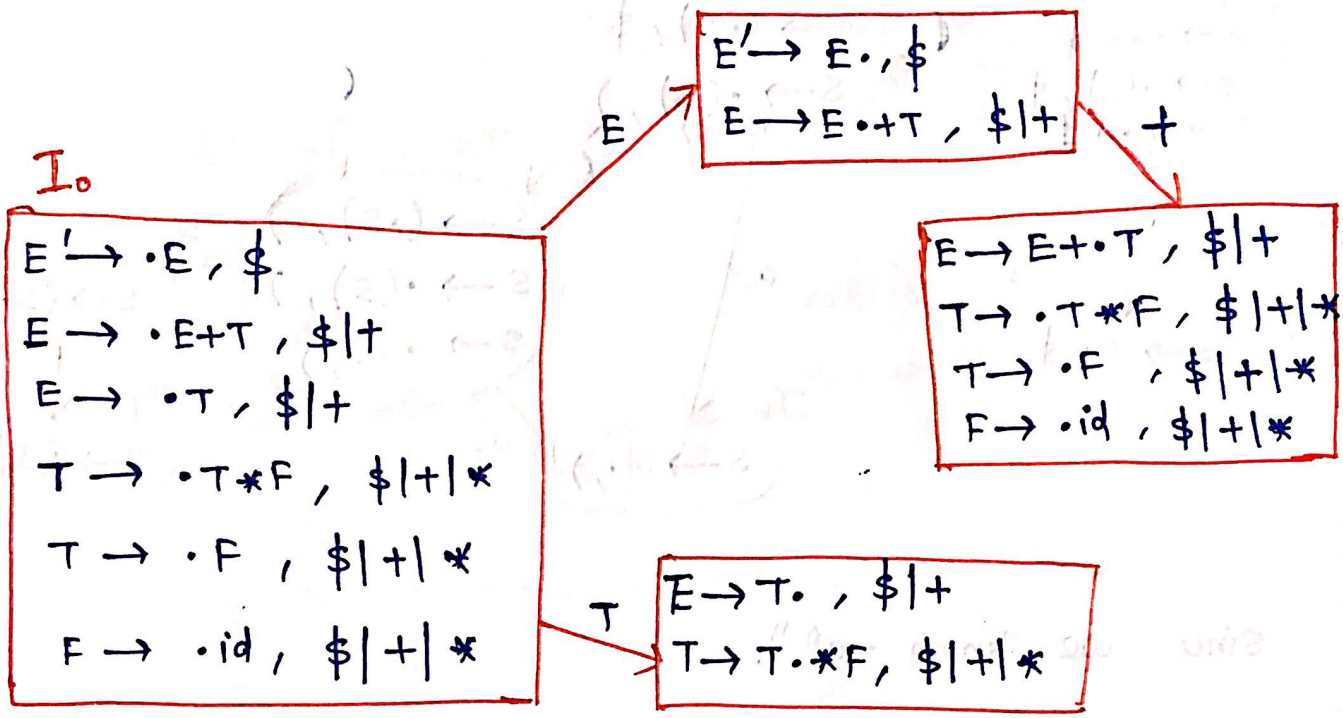
$$\left[(n_1 = n_2 = n_3) < n_4\right]$$

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

$T \to T*F \mid F$

$F \to id$

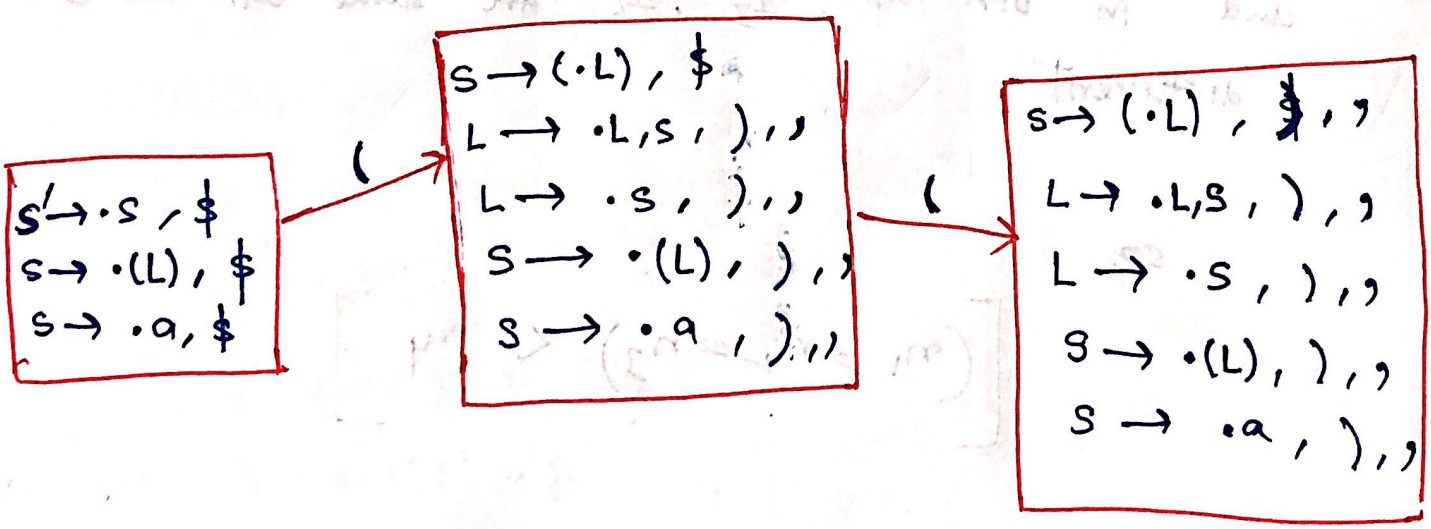
Construct DFA - 1st 4 states which contains more than one production.



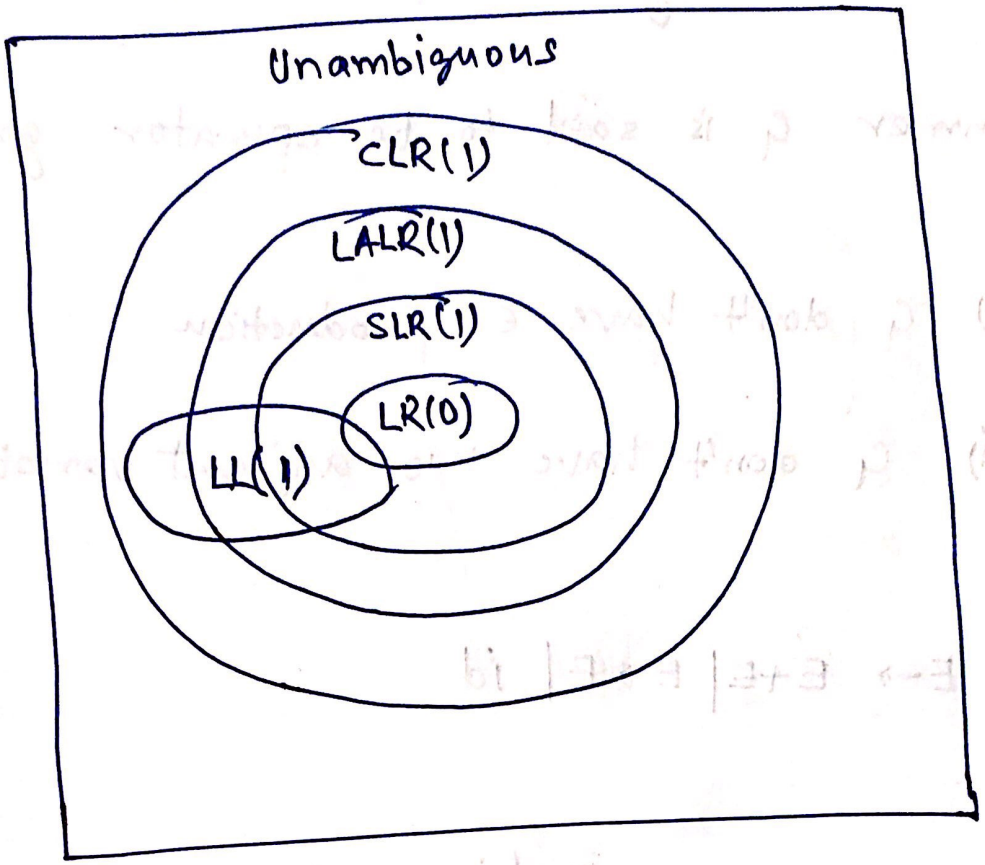
Q. $S \to (L) \mid a$

$L \to L,S \mid S$

Construct CLR(1) parser DFA 4 states.



Graphical representation of parsir grammers :-



NOTE :-

$$LL(1) \subseteq LR(1)$$
$$LL(2) \subseteq LR(2)$$
$$LL(K) \subseteq LR(K)$$

Operator Precedence Parser :-

- It is applicable only for operator grammer
- A Grammer G is said to be operator grammer iff
  - (i) G don't have ϵ production
  - (ii) G don't have two adjacent variable on RHS

eg:- $E \rightarrow E+E \mid E*E \mid id$

- It can works on ambiguous grammer.

$E \rightarrow E+E \mid E*E \mid id$	$E \rightarrow E+T \mid T$ $T \rightarrow T*F \mid F$ $F \rightarrow id$	$S \rightarrow AB$ $A \rightarrow a$ $B \rightarrow b$
Ambiguous G ✓	Ambiguous G ✗	Ambiguous G ✓
operator G ✓	operator G ✓	operator G ✗