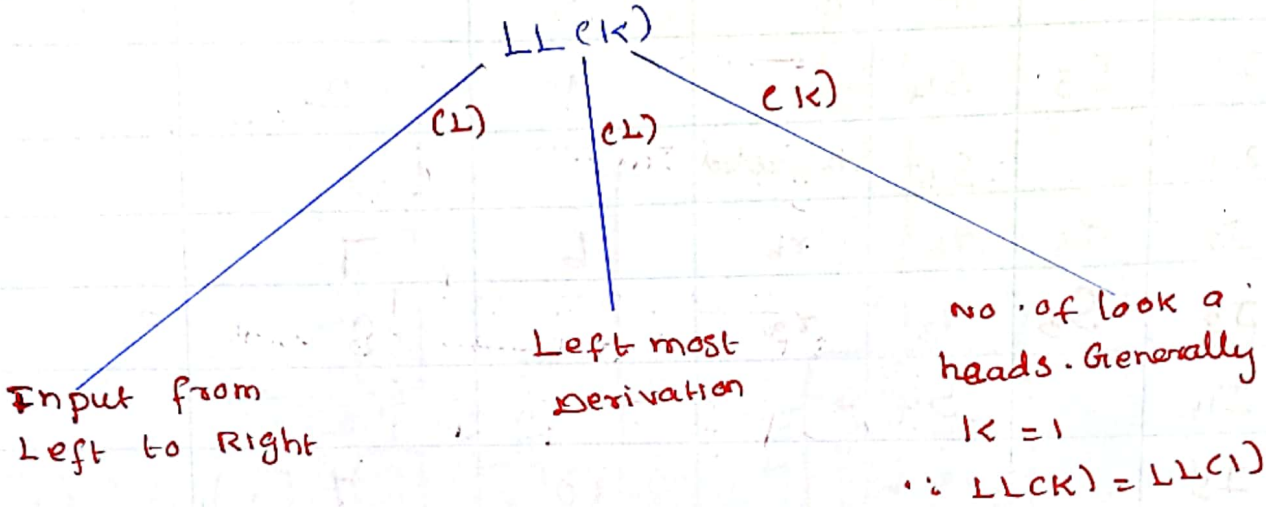
LL(1) parser (or) Predictive parser

LL parser accepts LL Grammar. It is denoted as LL(k)



A grammar G is LL(1).

-> If there are two distinct productions.

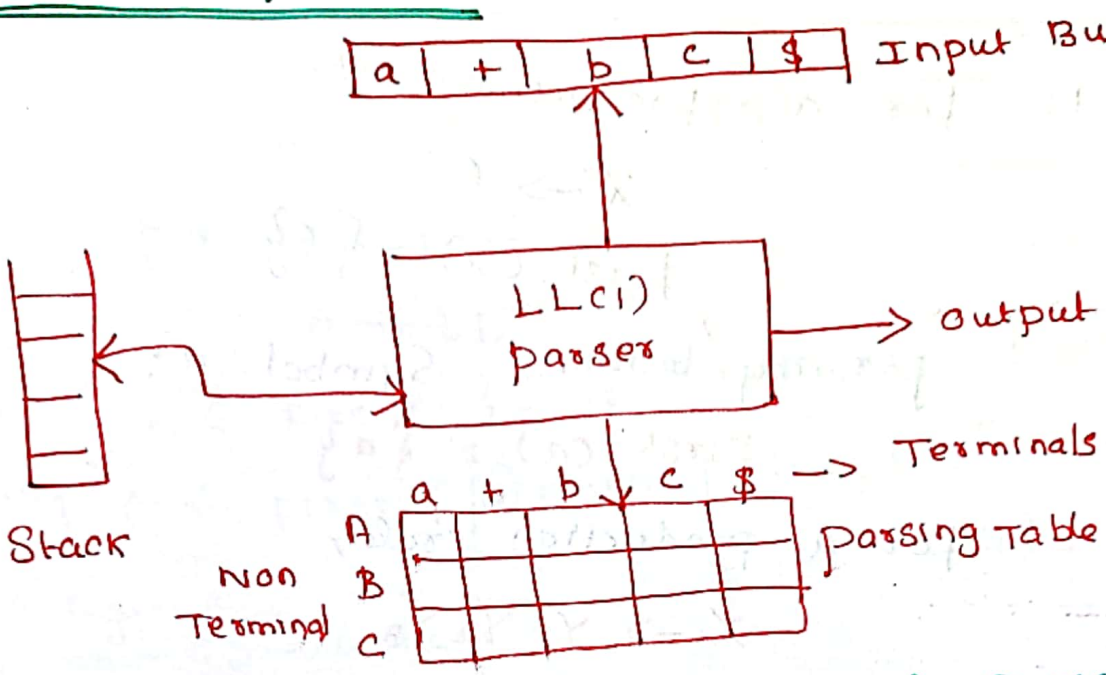
$$A \rightarrow \alpha \mid \beta$$

1) For no terminal α & β derive string beginning with 'a'.
2) At most one of α & β can derive empty string.
3. If $\beta \rightarrow \epsilon$, then 'a' does not derive any string beginning with a terminal in FOLLOW(A).

LL(1) uses data structure

1. Input Buffer
2. Stack
3. parsing Table.

Structure of LL(1):



Construction of Predective (or) LL(1) Parser:-

1) FIRST() | Leading()
   FOLLOW() | Trailing()
2) Predective parsing table by using first and follow functions.
3) Parse the input string with the help of the table.

i) Finding FIRST and FOLLOW:-

First and follow sets are needed, so that the parser can properly apply the needed production rule at the correct position.

(a) FIRST Function.

First (α) is a set of terminal symbol that begins in strings derived from α.

Eg, A → abc | def | ghi

then.

FIRST(A) = {a, d, g}

Rules for Calculating First Function:-

Rule 1: For a production rule

$$X \rightarrow \epsilon$$

$$First(X) = \{\epsilon\}$$

Rule 2: For any terminal Symbol a:

$$First(a) = \{a\}$$

Rule 3: For a production rule,

$$X \rightarrow Y_1 Y_2 Y_3$$

Calculating First(X):

→ If $\epsilon \notin First(Y_1)$, then $First(X) = First(Y_1)$

→ If $\epsilon \in First(Y_1)$, then $First(X) = \{First(Y_1) - \epsilon\} \cup First(Y_2 Y_3)$

Calculating First($Y_2 Y_3$):

→ If $\epsilon \notin First(Y_2)$, then $First(Y_2 Y_3) = First(Y_2)$

→ If $\epsilon \in First(Y_2)$, then $First(Y_2 Y_3) = \{First(Y_2) - \epsilon\} \cup First(Y_3)$.

Similarly, we can expand the rule for any production rule $X \rightarrow Y_1 Y_2 Y_3$

Follow Function:

Follow(A) is a set of terminal Symbol that appear immediately to the right of A.

Rule for Calculating Follow Function:-

Rule 1: For the Start Symbol S, place $ in Follow(S)

Rule 2: For any production rule

$A \rightarrow \alpha B$

$Follow(B) = Follow(A)$

Rule 3: For any production rule

$A \rightarrow \alpha B \beta$

-> If $\epsilon \notin First(\beta)$, then $Follow(B) = First(\beta)$

-> If $\epsilon \in First(\beta)$, then $Follow(B) = \{First(\beta) - \epsilon\} \cup Follow(A)$

Eg: $S \rightarrow aAC$

$A \rightarrow bd$.

$First(S) = a$

$First(A) = b$

$Follow(A) = C$

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

Note:-

1) ϵ may appear in the first function of a non-terminal.
2) ϵ will never appear in the follow function of a non-terminal.
3) It is recommended to eliminate left recursion from grammar, if present before calculating first and follow functions.
4) We will calculate the follow function of a non-terminal looking where it is present on Right hand side of a production rule

Example :-

Calculate the first and follow functions for the given grammar.

$S \rightarrow A$

$A \rightarrow aB \mid dA$

$B \rightarrow b$

$C \rightarrow g$

Solution :

The given grammar is left recursive, so first remove left recursion from the given grammar

-> After eliminating left recursion.

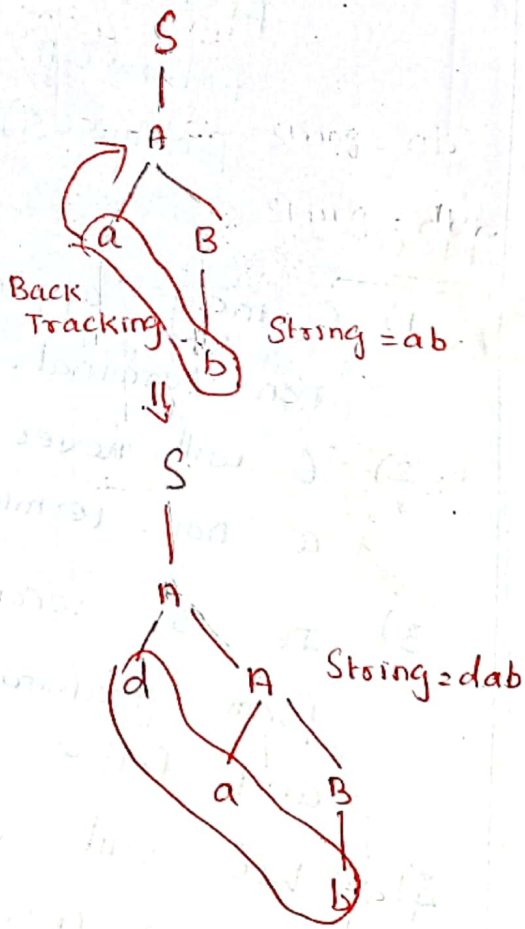
$S \rightarrow A$

$A \rightarrow aBA'$

$A' \rightarrow dA' \mid \epsilon$

$B \rightarrow b$

$C \rightarrow g$



First Function :-

First(S) = First(A) = {a}

First(A) = {a}

First(A') = {d, ϵ}

First(B) = {b}

First(C) = {g}

Follow Function:

Follow(A)
S → A) (no)

- Follow(S) = { $ }
- Follow(A) = Follow(S) ∪ { $ } ~~follow(A') = { $, d, ε }~~
- Follow(A') = Follow(A) = { $, d, ε }
- Follow(B) = First(A') − { ε } ∪ Follow(A)
  = { d, $ }
- Follow(C) = NA

Step 1:

Production	First	Follow
S → A	{a}	{ $ }
A → aBA'	{a}	{ $ }
A' → dA' | ε	{ d, ε }	{ $ }
B → b	{ b }	{ d, $ }
C → g	{ g }	NA

Step 2: Construct parse table using first and follow functions.

	a	d	b	g	$
S	S → A				
A	A → aBA'				
A'		A' → dA'			A' → ε
B			B → b		
C				C → g	

eg
First of S is 'a'

ε means go to follow up

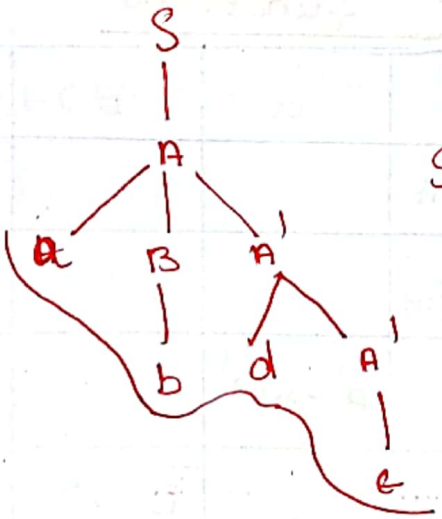
Stack Implementation Using parsing Table:-

Input String abd$

Stack	Input	Production
S$	abd$	S → A
A$	abd$	A → aBA'
aBA'$	abd$	pop a
BA'$	bd$	B → b
bA'$	bd$	pop b
A'$	d$	A' → dA'
dA'$	d$	pop d
A'$	$	A' → ε
$	$	Accepted [The input is properly passed

Generate the parse Tree:-

Generate the parse tree Stack implementation following Top down approach.



String = abdε

Construction of predective parsor LL(1) for the given grammar.

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

Solution:

Step 1: First Function

$First(F) = \{ (, id \}$

$First(T') = \{ *, \epsilon \}$

$First(T) = \{ (, id \}$

$First(E') = \{ +, \epsilon \}$

$First(E) = \{ (, id \}$

Step 2: Follow Function:

1. $Follow(E) = \{ \$,) \}$
2. $Follow(E') = \{ \$,) \}$ ie Follow $E' \rightarrow$ Follow E'
3. $\begin{cases} Follow(T) = \{ First(E') - \epsilon \} \cup Follow(E) \\ Follow(T') = \{ +, \$,) \} \end{cases}$
4. $Follow(T') = \{ +, \$,) \}$
5. $Follow(F) = \{ First(T') - \epsilon \} \cup (Follow(T)$

$\therefore Follow(F) = \{ *, +, \$,) \}$

Production	First	Follow
$E \rightarrow TE'$	$\{ (, id \}$	$\{ \$,) \}$
$E' \rightarrow +TE' \mid \epsilon$	$\{ +, \epsilon \}$	$\{ \$,) \}$
$T \rightarrow FT'$	$\{ (, id \}$	$\{ +, \$,) \}$
$T' \rightarrow *FT' \mid \epsilon$	$\{ *, \epsilon \}$	$\{ +, \$,) \}$
$F \rightarrow (E) \mid id$	$\{ (, id \}$	$\{ *, +, \$,) \}$

Step 3: Construct the passing Table:

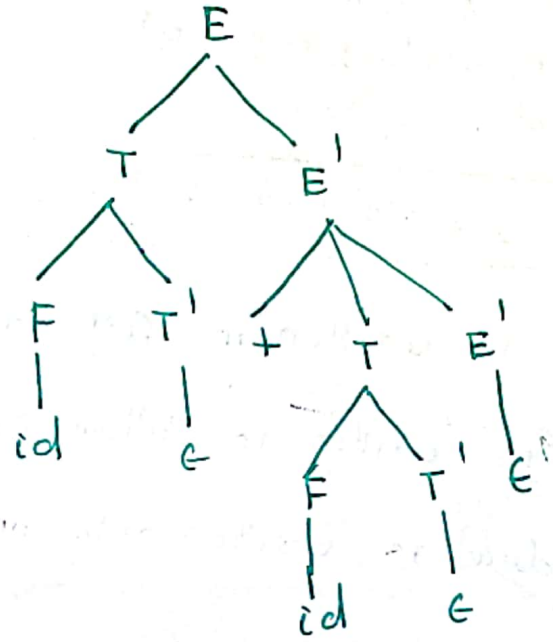
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$			$F \to (E)$		

↓ ↓ ↓ G means take follow fn

Step 4: Stack Implementation: take id + id

Step	Stack	Input	ACTION
1	$E\$$	$id + id\$$	$E \to TE'$
2	$TE'\$$	$id + id\$$	$T \to FT'$
3	$FT'E'\$$	$id + id\$$	$F \to id$
4	$idT'E'\$$	$id + id\$$	pop id
5	$T'E'\$$	$+id\$$	$T' \to \epsilon$
6	$E'\$$	$+id\$$	$E' \to +TE'$
7	$+TE'\$$	$+id\$$	pop +
8	$TE'\$$	$id\$$	$T \to FT'$
9	$FT'E'\$$	$id\$$	$F \to id$
10	$idT'E'\$$	$id\$$	pop id
11	$T'E'\$$	$\$$	$T' \to \epsilon$
12	$E'\$$	$\$$	$E' \to \epsilon$
13	$\$$	$\$$	Accepted

Step 5: parse Tree.



CLR(1) & LALR(1) parsing

Canonical LR(CLR)

Canonical collection of

> LR(1) items => LR(0) item + look a head

LR(0) => reduce is written in full row

SLR(1) = Reduce is written in follow of production

CLR & LALR => reduce is written only on look a head

Eg of Look a head:

E -> BB

B -> cB | d

Augment grammar & LR(1) Items

$E' \to \cdot E, \$$ -> look a head item

$E \to \cdot BB, \$$ -> First of B is c & d. -> Look a head item

$B \to \cdot cB \mid \cdot d, c/d$ -> Look a head item

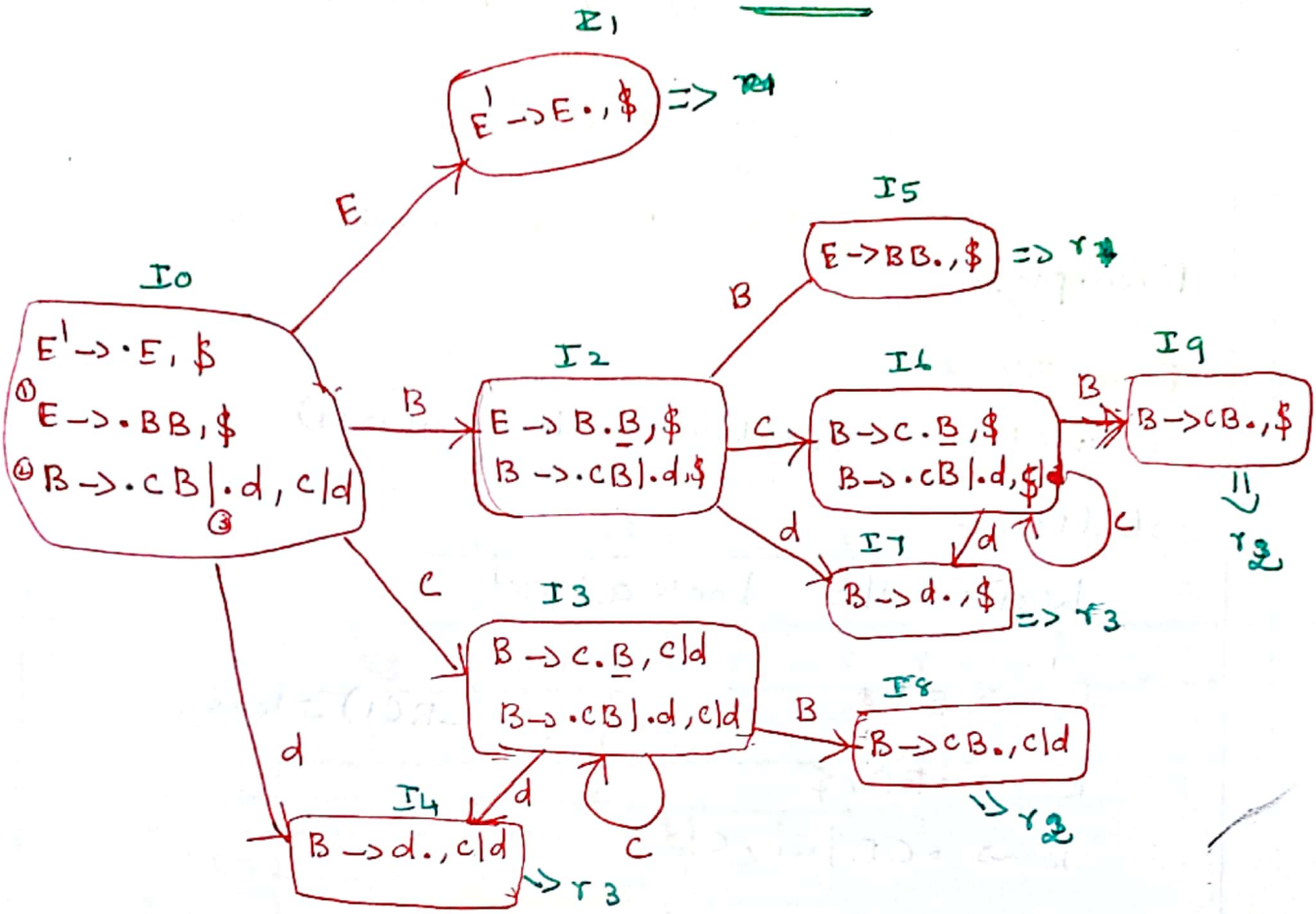
Write First of c & d

$E' \to \cdot E, \$$

$E \to \cdot BB, \$$

$B \to \cdot cB \mid \cdot d, c/d$

CLRC1)



States	ACTION			GOTO	
	c	d	$	E	B
I0	S_3	S_4		1	2
I1			Accept		
I2	S_6	S_7			5
I3	S_3	S_4			8
I4	r_3	r_3			
I5			r_1		
I6	S_6	S_7			9
I7			r_3		
I8	r_2	r_2			
I9			r_2		

LALR(1) Look a head LR

⇓

LR(1) Items

⇓

LR(0) items + look a head value.

Example:-

E -> BB

B -> cB|d Construct the LALR(1)

Solution:-

Write the Look a head

$E' \to .E, \$$

$E \to .BB, \$$

$B \to .cB|.d, c|d$ } ⇒ LR(1) Items.

Check the CLR Items.

∴ I_3 & I_6 is Similar production but the look a head value is different.

→ I_4, I_7 is Similar but look a head is different

→ I_8, I_9 also Similar product but look a head value is different.

∴ I_3, I_6

I_4, I_7

I_8, I_9 } Same production & Look a head value is different

Step 1: First construct the CLR Diagram.

Step 2: Construct the CLR Table.

Step 3: Reduce the no. of States.

States	ACTION			GOTO	
	c	d	$	E	B
I_0	S_{36}	S_{47}		1	2
I_1			Accept		
I_2	S_{36}	S_{47}			5
I_{36}	S_{36}	S_{47}			8,9
I_{47}	r_3	r_3	r_3		
I_5			r_1		
~~I_{36}~~	~~S_{36}~~	~~S_{47}~~			~~89~~
~~I_{47}~~			r_3		
I_{89}	r_2	r_2	r_2		
~~I_{89}~~			r_2		

⇓ Reconstruct the above table. ie LALR Table

States	ACTION			GOTO	
	c	d	$	E	B
I_0	S_{36}	S_{47}		1	2
I_1			Accept		
I_2	S_{36}	S_{47}			5
I_{36}	S_{36}	S_{47}			89
I_{47}	r_3	r_3	r_3		
I_5			r_1		
I_{89}	r_2	r_2	r_2		

Reduce from 10 States to 7 States

Example: Construct the Canonical LR parsing(CLR) table for the given grammar.

$S \to CC$

$C \to aC$

$C \to b$

Solution:

Write the augmented Grammar.

$S' \to .S, \$ \to$ look a head

$S \to .CC$, First($\$$) = $\$ \to$ look a head

$C \to .aC$, First($C\$$) => ab

$C \to .b$, First($C\$$) => ab.

⇓

$S' \to .S, \$$

$S \to .CC, \$$

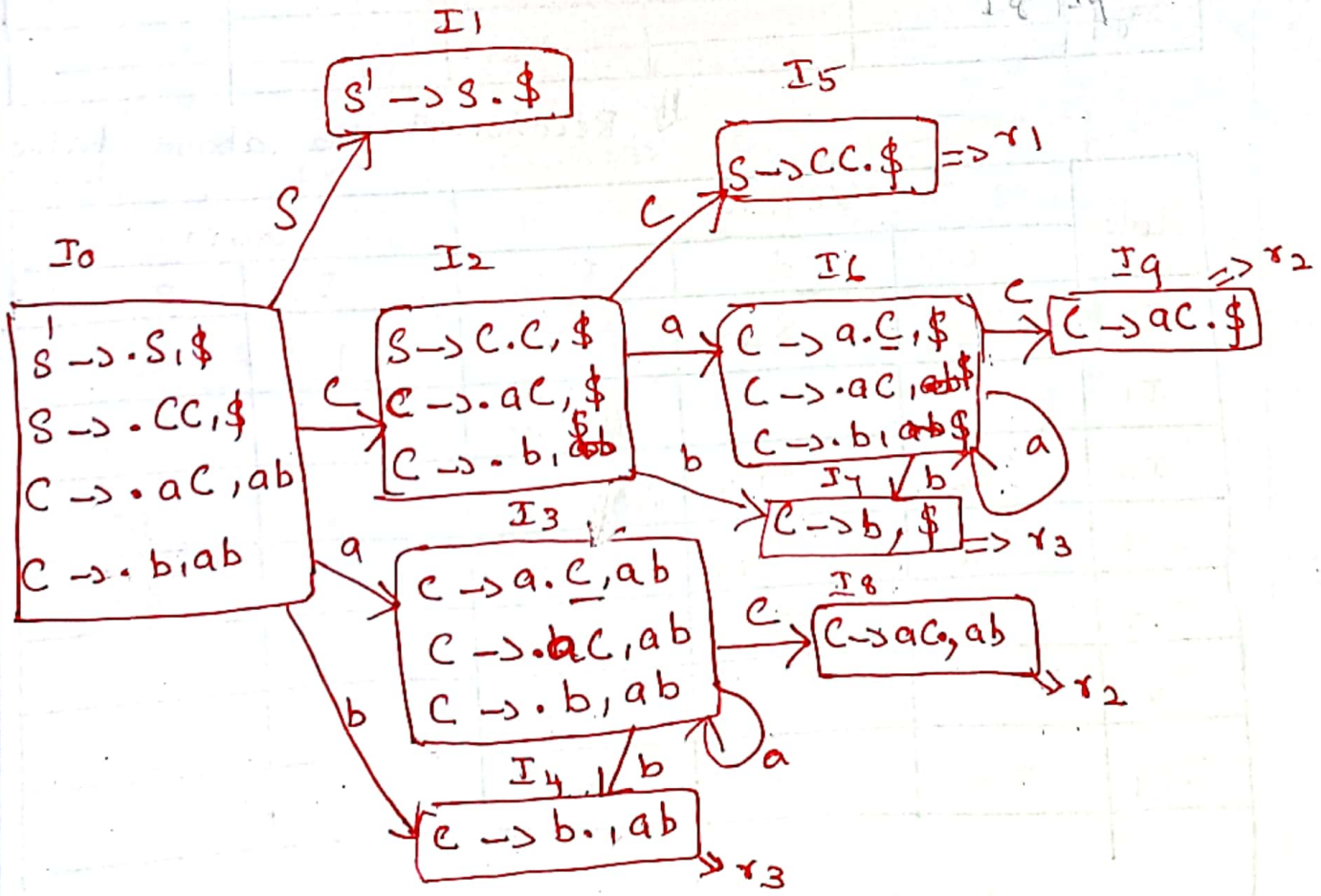
$C \to .aC, ab$

$C \to .b, ab$

I3 I6

I4 I7

I8 I9



States	ACTION			GOTO	
	a	b	$	S	C
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_6	S_7			5
I_3	S_3	S_4			8
I_4	r_3	r_3			
I_5			r_1		
I_6	S_6	S_7			9
I_7			r_3		
I_8	r_2	r_2			
I_9			r_2		

H/W Construct the CLR parsing table for the given grammar.

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

$A \rightarrow c$

$B \rightarrow c$