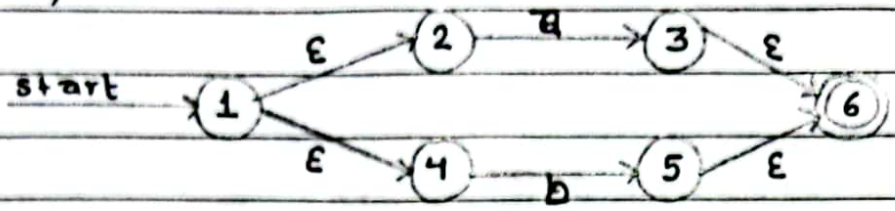
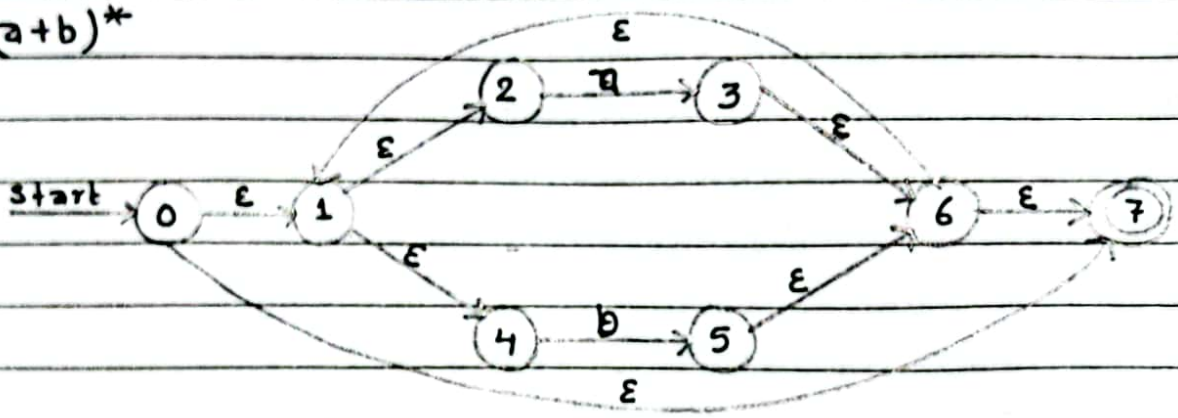
Assignment 1.

1. At first construct NFA of RE $(a+b)^*abb$ then convert resulting NFA to DFA

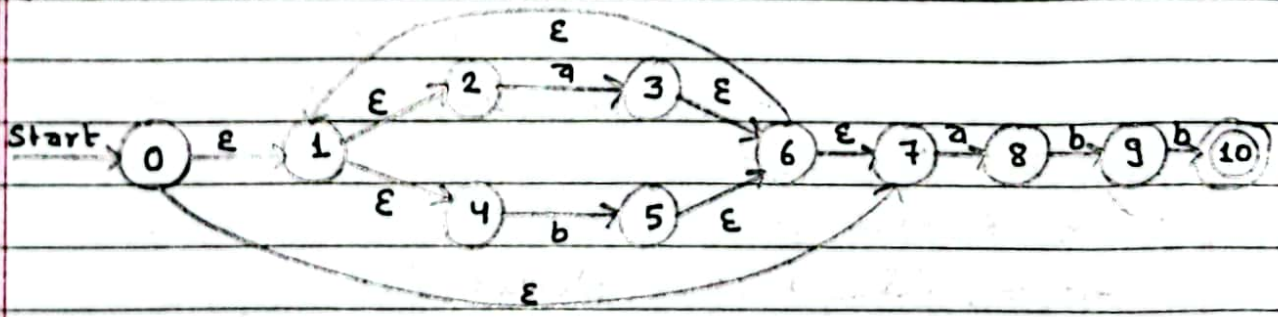
$(a+b)$



$(a+b)^*$



$(a+b)^*abb$



Starting state of DFA = S_0 = ε-closure ({0})
= {0, 1, 2, 4, 7}

$\Rightarrow S_0 = \{0, 1, 2, 4, 7\}$

Mark S_0:

For a: ε-closure (move (S_0, a)) = ε-closure (3, 8)
= {1, 2, 3, 4, 6, 7, 8} $\Rightarrow S_1$

For b (move (S_0, b) = ε-closure (5)
= {1, 2, 4, 5, 6, 7} ⇒ S_2

Mark S_1
For a: (S_1, a) = ε-closure (3, 8)
= {1, 2, 3, 4, 6, 7, 8} ⇒ S_1
For b: (S_1, b) = ε-closure (5, 9)
= {1, 2, 4, 5, 6, 7, 9} ⇒ S_3

Mark S_2
For a: (S_2, a) = ε-closure (3, 8)
= {1, 2, 3, 4, 6, 7, 8} ⇒ S_1
For b: (S_2, b) = ε-closure (5, 10)
= {1, 2, 4, 5, 6, 7, 10} ⇒ S_4

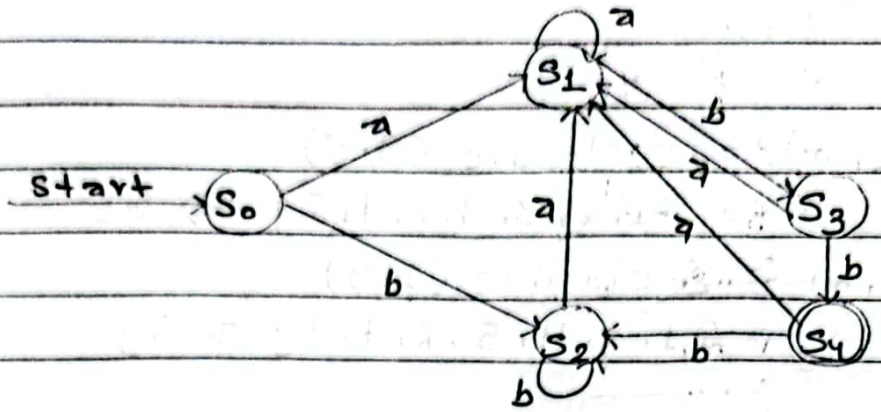
Mark S_3
For a: (S_3, a) = ε-closure (3, 8)
= {1, 2, 3, 4, 6, 7, 8} ⇒ S_1
For b: (S_3, b) = ε-closure (5, 10)
= {1, 2, 4, 5, 6, 7, 10} ⇒ S_4

Mark S_4
For a: (S_4, a) = ε-closure (3, 8)
= {1, 2, 3, 4, 6, 7, 8} ⇒ S_1
For b: (S_4, b) = ε-closure (5)
= {1, 2, 4, 5, 6, 7} ⇒ S_2

S_0 is the start state of DFA since 0 is a member of S_0 {0, 1, 2, 4, 7}.
S_4 is the accepting state of DFA since final state of NFA is i.e. 10 is a member of S_4

{1, 2, 4, 5, 6, 7, 10}.

Now construct DFA from above information, as.



2. Convert the regular expression b.a(a+b)* ab into equivalent DFA by direct method.

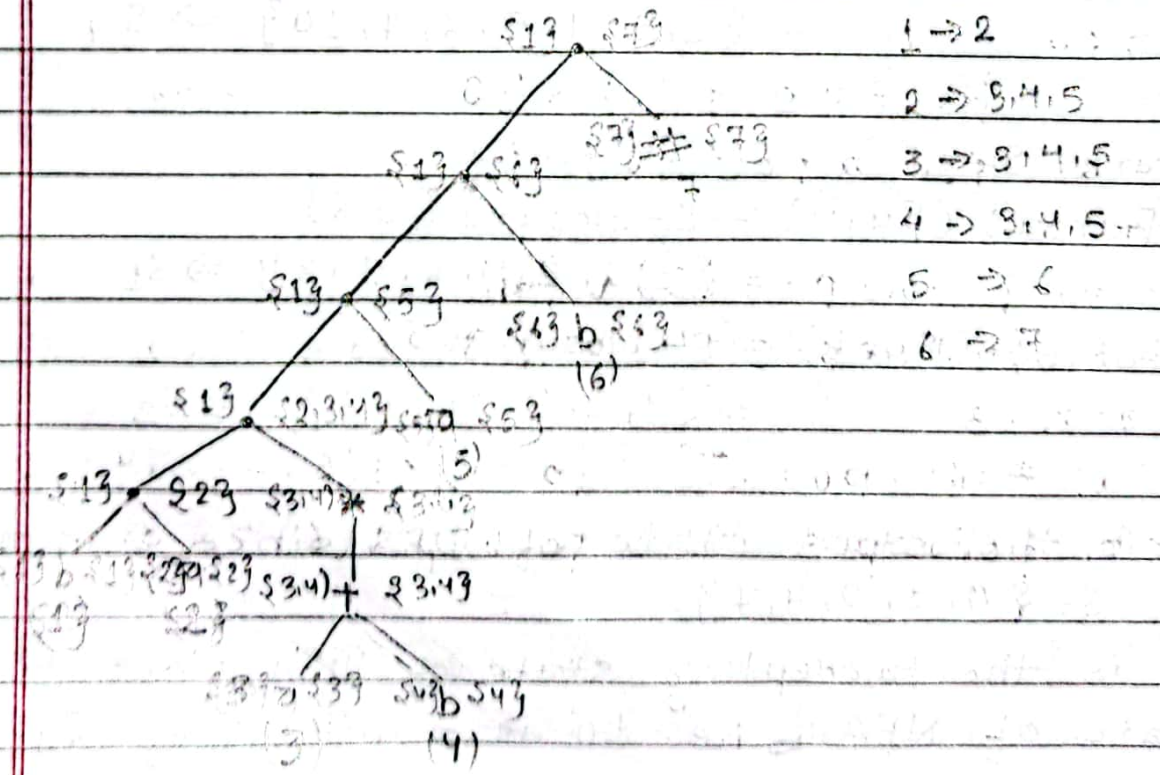
Here,

b.a.(a+b)*.ab

Step 1: Augment the given regular expression as,

b.a.(a+b)*.a.b.#

Step 2: Construct syntax tree of augmented regular expression.



Step3: Compute follow as

Node	Followpos
1	{2}
2	{3,4,5}
3	{3,4,5}
4	{3,4,5}
5	{6}
6	{7}
7	{ϕ}

step4: After we calculate followpos, we are ready to create DFA for the regular expressions as.

Starting state of DFA = S_0

= First pos (root node of syntax tree)

= 1

Mark S_0

For a, followpos {ϕ} = {ϕ}

For b, followpos {1} = {2} → S_1

Mark S_1

For a, followpos (2) = {3,4,5} → S_2

For b, followpos {ϕ} = {ϕ}

Mark S_2

For a, followpos (3) ∪ followpos (5) = {3,4,5,6} → S_3

For b, followpos (4) = {3,4,5} → S_2

Mark S_3

For a, followpos (3) ∪ followpos (5) = {3,4,5,6} → S_3

For b, followpos (4) ∪ followpos (6) = {3,4,5,7} → S_4

Mark S_4

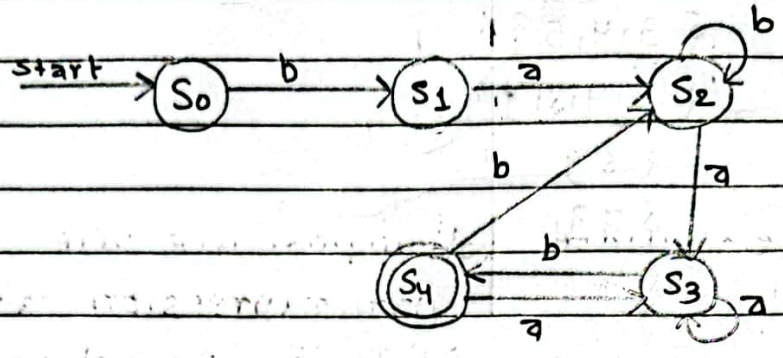
For a, followpos (3) ∪ followpos (5) = {3,4,5,6} → S_3

For b, followpos (4) = {3,4,5} → S_2

Since, no new state occur,

Starting state of DFA = {S_0}

Accepting state of DFA = {S_4}



3. Convert the regular expression (a|ε) b c* into equivalent DFA by direct method.

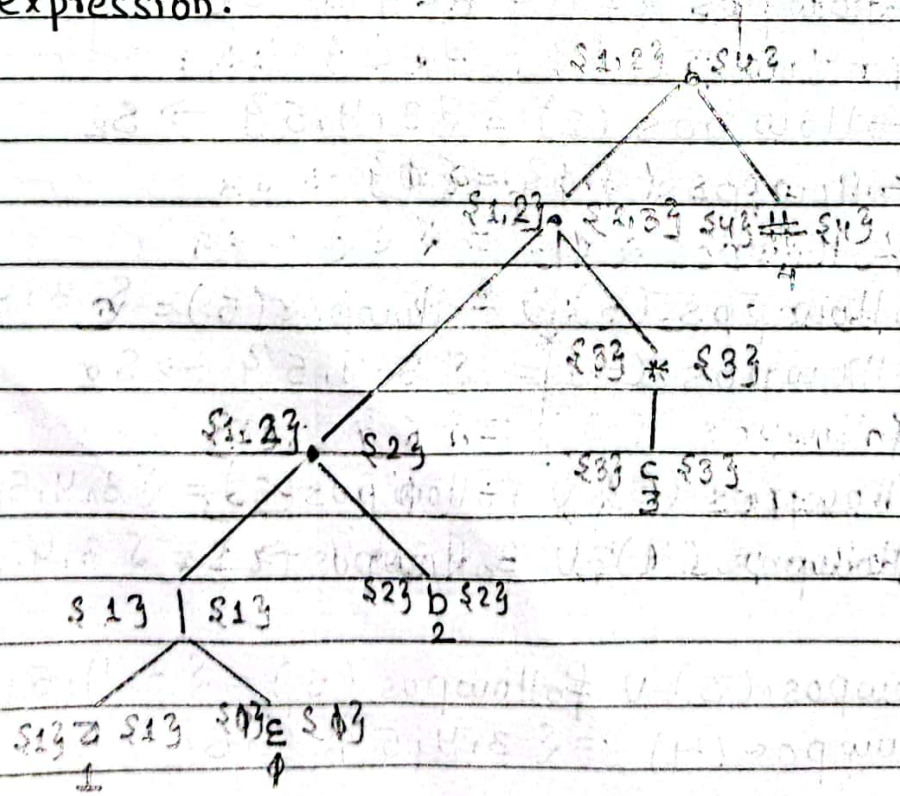
Here,

(a|ε) b c*

step 1: Augment the given regular expression as.

⇒ (a|ε)•b•c*•#

step 2: Construct syntax tree of augmented regular expression.



Step3: Compute followpos as,

Node	Followpos
1	{2}
2	{3,4}
3	{3,4}
4	{ϕ}

Step4: After we calculate followpos, we are ready to create DFA for regular expression as,

starting state of DFA = S_0 = Firstpos (root node of syntax tree)

= {1,2}

Mark S_0,

For a, followpos (1) = {2} → S_1

For b, followpos (2) = {3,4} → S_2

For c, followpos (ϕ) = {ϕ} → S_3

Mark S_1

For a, followpos (ϕ) = {ϕ} → S_3

For b, followpos (2) = {3,4} → S_2

For c, followpos {ϕ} = {ϕ} → S_3

Mark S_2

For a, followpos (3) = {ϕ} → S_3

For b, followpos {ϕ} = {ϕ} → S_3

For c, followpos (3) = {3,4} → S_2

Mark S_3

For a, followpos (ϕ) = {ϕ} → S_3

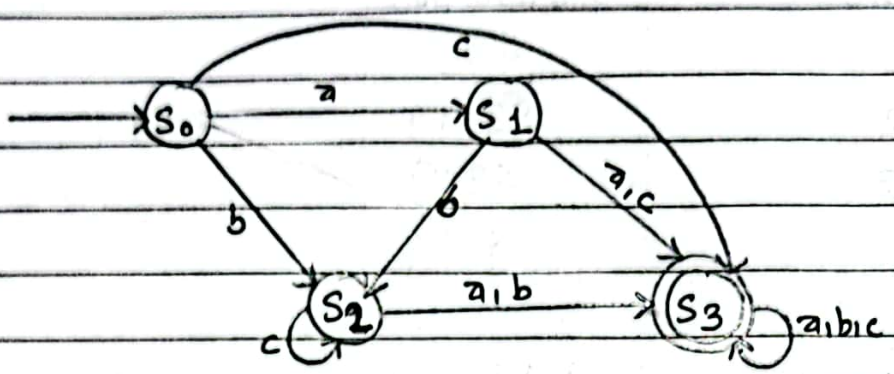
For b, followpos (ϕ) = {ϕ} → S_3

For c, followpos (ϕ) = {ϕ} → S_3

Since, no new state occur

starting state of DFA = {S_0}

Accepting State of DFA = $\{S_2\}$



4. Consider the grammar,

$S \rightarrow aBc \mid aaB$

$B \rightarrow bc \mid a$

Input string: aaa

Parse using Recursive-Descent Parsing.

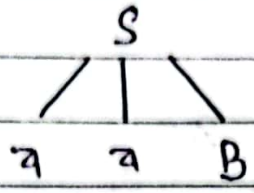
Solution,

Input	Output	Rule used
aaa	S	S → ~~aBc~~ aBc
aaa	~~aBc~~ aBc	Match symbol a
aa	Bc	Use B → a
aa	ac	Match symbol a

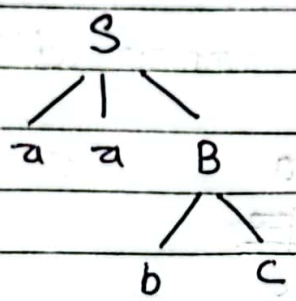
Input	Output	Rule used
aaa	S	Use S → aaB
aaa	aaB	Match Symbol a
aa	aB	Match Symbol a
a	B	Use B → bc
a	bc	Dead end, Backtrack
a	B	Use B → a
a	a	Match symbol a
ϕ	ϕ	Accepted

Graphically,

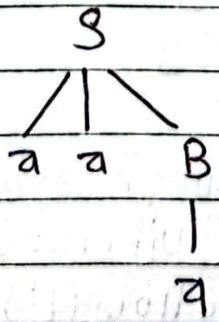
Step 1: The first rule $S \rightarrow aaB$



Step 2: The next non-terminal is B and is parsed using production $B \rightarrow bc$ as,



Step 3: Which is false and now use backtrack and use production $B \rightarrow a$ to parse for B.



5. Eliminate left recursion from the following grammar and also find first and follow;

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

Solution,

Eliminate left recursion as;

$S \rightarrow (L) \mid a \qquad L \rightarrow SL'$

$L' \rightarrow ,SL' \mid \varepsilon$

Now,

~~First~~ of Grammar without left recursion is,

$S \rightarrow (L) \mid a$

$L \rightarrow SL'$

$L' \rightarrow ,SL' \mid \varepsilon$

Now,

First of grammar are:

i) First (S) = { (, a }

ii) First (L) = { (, a }

iii) First (L') = { , , ε }

Then,

Follow of grammar are

i) Follow of (S) = { $ } ∪ First (L') − { ε } ∪ Follow (L) ∪ Follow (L') =

= { $, , ,) }

ii) Follow of (L) = {) }

iii) Follow of (L') = Follow of (L) = {) }

6 Eliminate left recursion from following grammar.

$A \rightarrow Ad \mid Ae \mid aB \mid ac$

$B \rightarrow bBc \mid f$

Solution

Eliminate left recursion from the grammar.

$A' \rightarrow dA' \mid eA' \mid \varepsilon$

$A \rightarrow aBA' \mid aCA'$

$B \rightarrow bBc \mid f$

So the grammar without left recursion is,

$A \rightarrow aBA' \mid aCA'$

$A' \rightarrow dA' \mid eA' \mid \varepsilon$

$B \rightarrow bBc \mid f$

---

7. Calculate first and follow functions for the given grammar

$S \rightarrow A$

$A \rightarrow aB \mid Ad$

$B \rightarrow b$

$C \rightarrow g$

Solution,

Given grammar is left-recursive.

Eliminating Left Recursion,

$S \rightarrow A$ $\quad A \rightarrow aBA'$

$A' \rightarrow dA' \mid \varepsilon$

$B \rightarrow b$

$C \rightarrow g$

Now,

Computing First function as,

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

First (A) = {a}
First (A') = {d, ε}
First (B) = {b}
First (C) = {g}

Then,
Computing Follow function as,

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

8. Calculate the first and follow functions for the given grammar,

S → ACB | CbB | Ba
A → da | BC
B → g | ε
C → h | ε

Solution
The given grammar is free of left-recursion
Now, Computing first function as,

i) First (S) = {First (A) - ε} ∪ {First (C) - ε} ∪ First (B) ∪ First (b) ∪ First (a)
= {d, g, h, ε, b, a}

ii) First (A) = First (d) ∪ {First (B) - ε} ∪ first (C)
= {d, g, h, ε}

iii) First (B) = {g, ε}

iv) First (C) = {h, ε}

Then,
Computing Follow Function as,

Follow (S) = { $ }
Follow (A) = { First (t) - ε } ∪ { First (B) - ε } ∪ Follow (S)
= { $, h, g }
Follow (B) = Follow (S) ∪ First (a) ∪ { First (C) - ε } ∪ Follow (A)
= { $ a, h, g }
Follow (E) = Follow(S) ∪ First (b) ∪ { First (f) - ε } ∪ Follow (A)
= { $, b, h, g }

9. Construct LL(1) Parsing Tables for following grammar:

S → i E t S S' | a
S' → e s | ε
E → b

Solution,
Computing First of Grammar as,
i) First (S) = { i, a }
ii) First (S') = { e, ε }
iii) First (E) = { b }

Now, computing follow as,
i) Follow (S) = { $, e }
ii) Follow (S') = { Follow (S) } = { $, e }
iii) Follow (E) = First (t S S') = { t }
Then
Constructing LL(1) Parsing Table.

Non Terminals	Terminal Symbol					
	i	t	a	e	b	$
S	S→iEtSS'		S→a			
S'				S'→eS		
				S→ε		S'→ε
E					E→b	

The given grammar is not LL(1) because there are two productions into the same cell.

10. Consider the following grammar,

S → L = R
S → R
L → *R
L → id
R → L

Construct LL(1) parsing tables for the given grammar.

Computing first

i) First (S) = First (L) = {*, id}
ii) First (L) = {*, id}
iii) First (R) = First (L) = {*, id}
i.e First (S) = First (L) = First (R) = {*, id}

Computing Follow
i) Follow (S) = {$}

ii) Follow (L) = { $, = }

iii) Follow (R) = Follow(S) ∪ Follow (L)
= { $, = }

Now,

LL(1) Parsing Table

Non Terminals	Terminal Symbol			
	*	id	$	=
S	S → L=R	S → L= R		
L	L → *R	L → id		
R	R → L	R → L		

Given grammar is not LL(1) since there are more than one ~~ie S produc~~ production into the same cell, which is; S.

	*	id
S	S → L=R	S → L=R
	S → R	S → R

11. Consider the following grammar

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

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

Solution

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

Computing Shift-Reduce parsing for the input $(a,(a,a))$.

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)

$ stack	Input Buffer	Parsing Action
$ S	$	Accept

12. Consider the following 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.

Solution

Performing shift-Reduce Parsing for "int id, id;"

Stack	Input Buffer	Parsing Action
$	int id, id; $	Shift
$ int	id, id; $	Reduce $T \rightarrow int$
$ T	id, id; $	Shift
$ T id	, id; $	Reduce $L \rightarrow id$
$ T L	, id; $	Shift Reduce $S \rightarrow TL$
$ T L,	id; $	Shift
$ T L, id	; $	Reduce $L \rightarrow L, id$
$ T L	; $	Shift
$ T L;	$	Reduce $S \rightarrow TL;$
$ S	$	Accept

13. Construct LR(0), SLR(1), CLR and LALR parsing for the grammar.

$E \rightarrow B\ B$

$B \rightarrow cB \mid d$

And parse the input string : ccdd.

Solution.

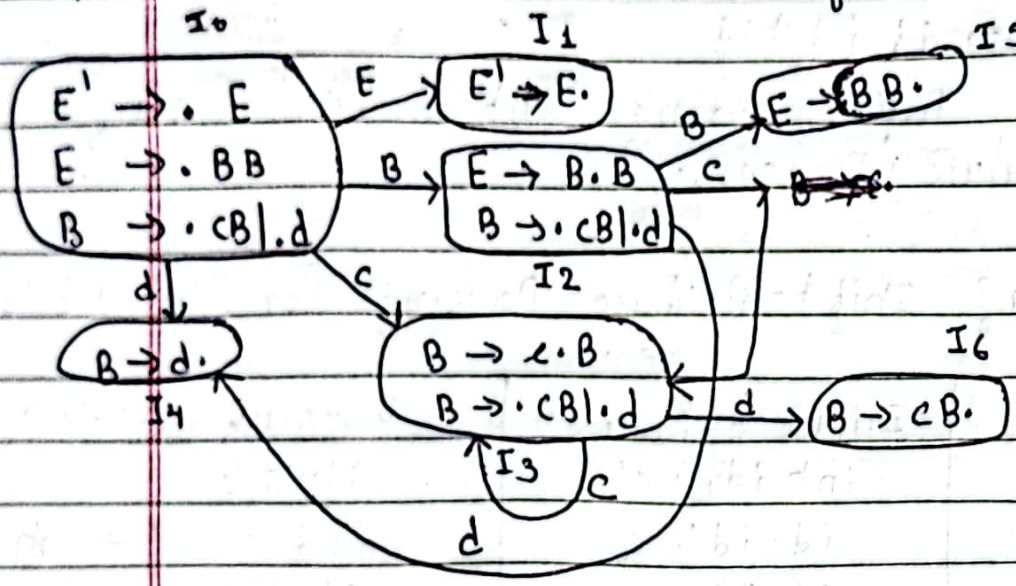
Step 1: Augment the given grammar

$E' \to E$

$E \to BB$

$B \to cB \mid d$

Step 2: Draw canonical collection of LR(0) item



Step 3: Number the production,

$E' \to E$

$E \to BB$ — (1)

$B \to cB$ — (2)

$B \to d$ — (3)

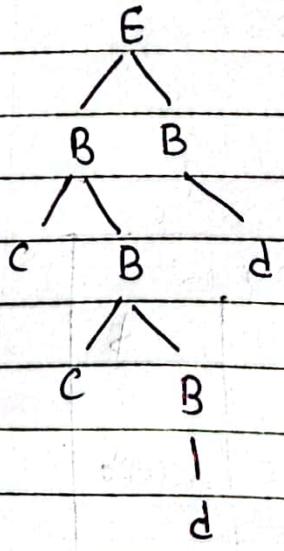
Step 4: Parsing table

State	Action			Goto	
	c	d	$	E	B
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
I_4	R_3	R_3	R_3		
I_5	R_1	R_1	R_1		
I_6	R_2	R_2	R_2		

Step5: Stack implementation ccdd$

Stack	Input	Action
$0	ccdd$	Shift C → S3
$0C3	cdd$	shift C → S3
$0C3C3	dd$	shift d → S4
$0C3C3d4	d$	Reduce R3 B→d
$0C3C3B6	d$	Reduce R2 B→cB
$0C3B6	d$	Reduce R2 B→cB
$0B2	d$	shift d → S4
$0B2d4	$	Reduce R3 B→d
$0B2B5	$	Reduce R1 E→BB
$0E1	$	Accept

Step6: Parse Tree



SLR(1)

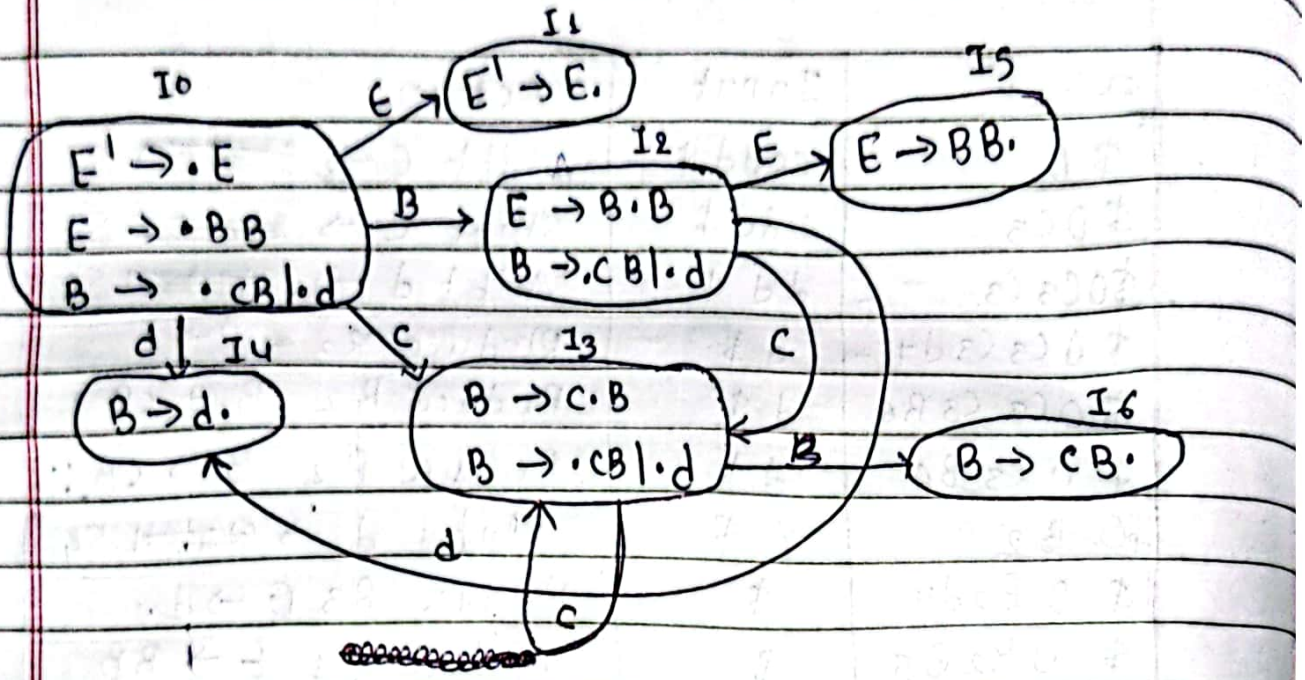
Step 1: Augment the given grammar

E' → E

E → BB

B → CB|d

Step2: Draw canonical collection LR(0) items



Step 3: Number the production,

$E' \rightarrow E$

$E \rightarrow BB$ — (1)

$B \rightarrow cB$ — (2)

$B \rightarrow d$ — (3)

Step4: Parsing Table

State	Action			Goto	
	c	d	$	E	B
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
I_4	R_3	R_3	R_3		
I_5			R_1		
I_6	R_2	R_2	R_2		

Look for follow,

$B \rightarrow d$ Follow(B) = {c, d, $} $B \rightarrow cB$ Follow(B) = {c, d, $}

$E \rightarrow BB$ Follow(E) = {$}

Step 1: Augment of the grammar for look ahead

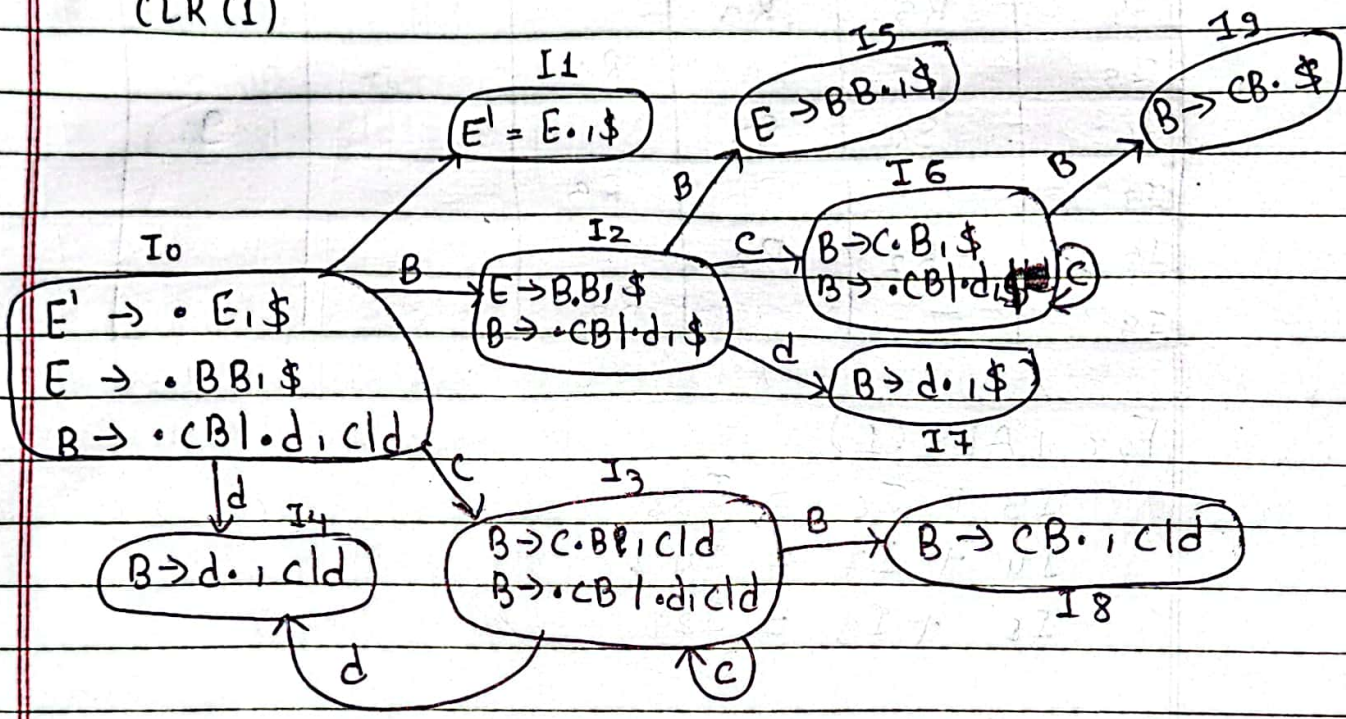
$E' \to \cdot E, \$$

$E \to \cdot BB, \$$

$B \to \cdot CB / \cdot d, \$ / cld$

Step 2: Draw Canonical collection of LR(1)

(CLR(1))



Step 3: Number the production

$E' \to E$

$E \to BB$ — (1)

$B \to CB$ — (2)

$B \to d$ — (3)

Step 4: Parsing Table

State	Action			Go to	
	c	d	$	E	B
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		

For LALR(1)

$I_3 + I_6 = I_{36}$

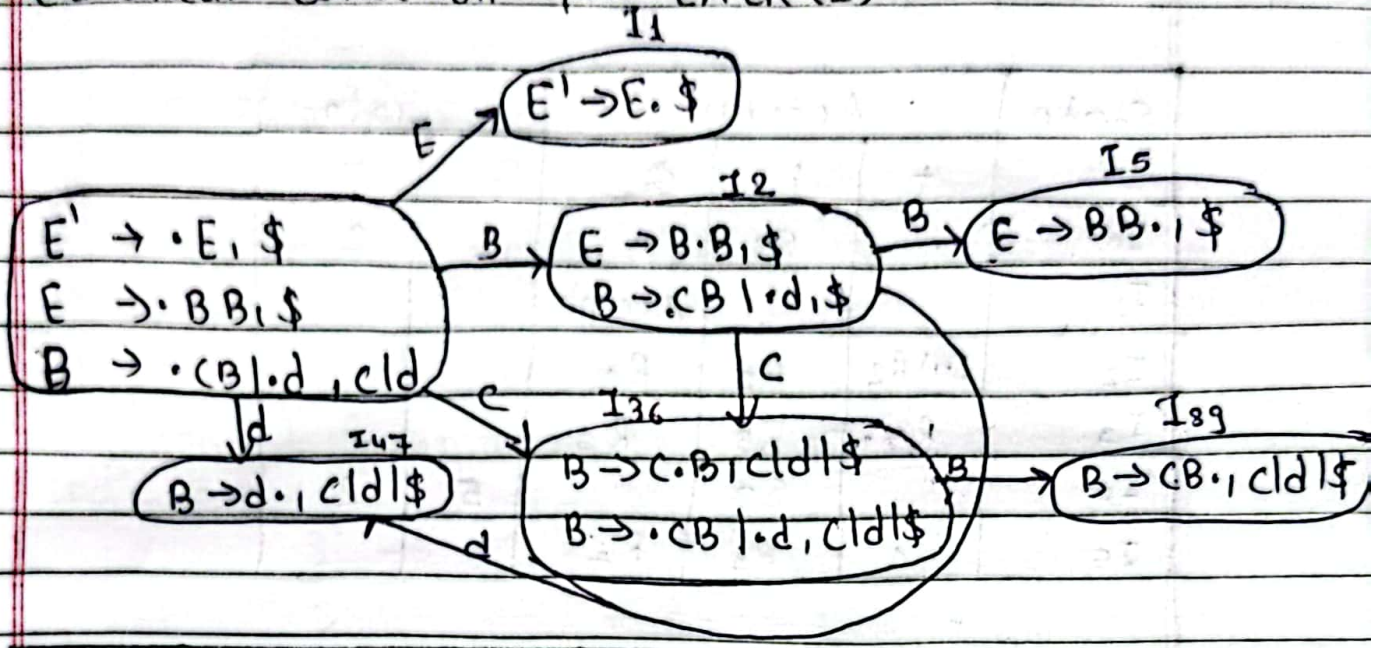
$I_4 + I_7 = I_{47}$

$I_8 + I_9 = I_{89}$

Parsing Table

State	Action			Go to	
	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		

Canonical collection for LALR(1):



14. Check whether the following grammar is LR(0) or not. If not LR(0), then parse it using SLR(1).

$E \to T+E \mid T$

$T \to i$

Solution;

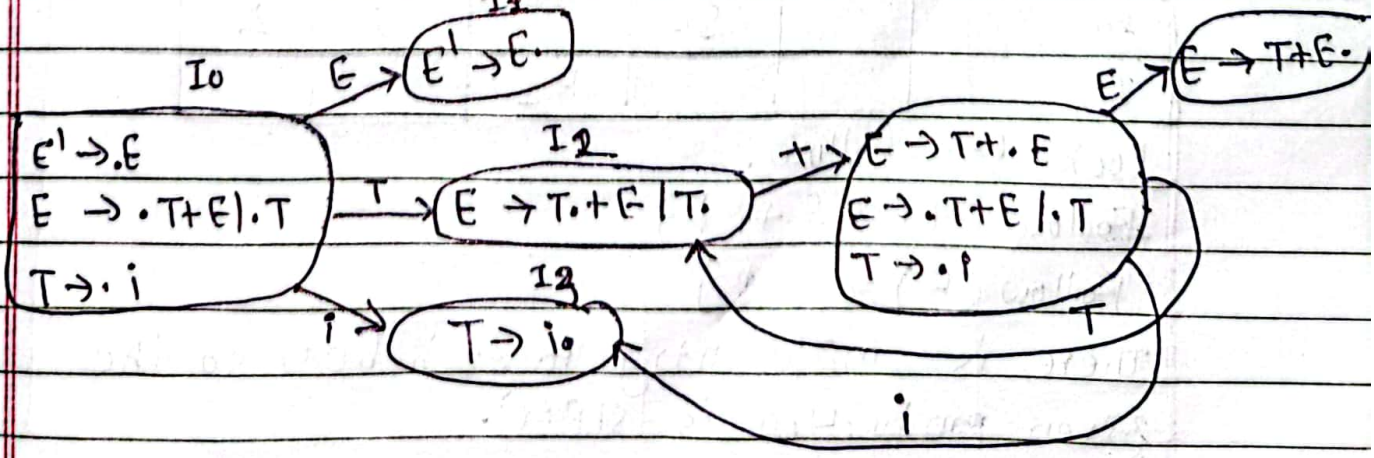
Step1: Augment of given grammar

$E' \to .E$

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

$T \to .i$

Step 2: Draw canonical collection of LR(0) item:



Step 3: Draw parsing Table for LR(0)

State	Action			Goto	
	+	i	$	E	T
I_0		S_3		1	2
I_1			Accept		
I_2	S_4/R_2	R_2	R_2		
I_3	R_3	R_3	R_3		
I_4		S_3		5	2
I_5	R_1	R_1	R_1		

It is not LR(0) as two entries (S_4 & R_2) are in the same box.

Now, Parsing it using SLR(1)

Parsing Table [SLR(1)]

State	Action			Goto	
	+	i	$	E	T
I_0		S_3		1	2
I_1			Accept		
I_2	S_4		R_2		
I_3	R_3		R_3		
I_4		S_3		5	2
I_5			R_1		

Look for follow:

Follow(T) = { +, $ }

Follow(E) = { $ }

There is single entry in each box, so the given production is SLR(1).

15. Consider the following grammar:

$S \to CC$

$C \to cC$

$C \to d$

Construct CLR and LALR parsing table for the grammar.

Augmented Grammar of given grammar with 1 look ahead

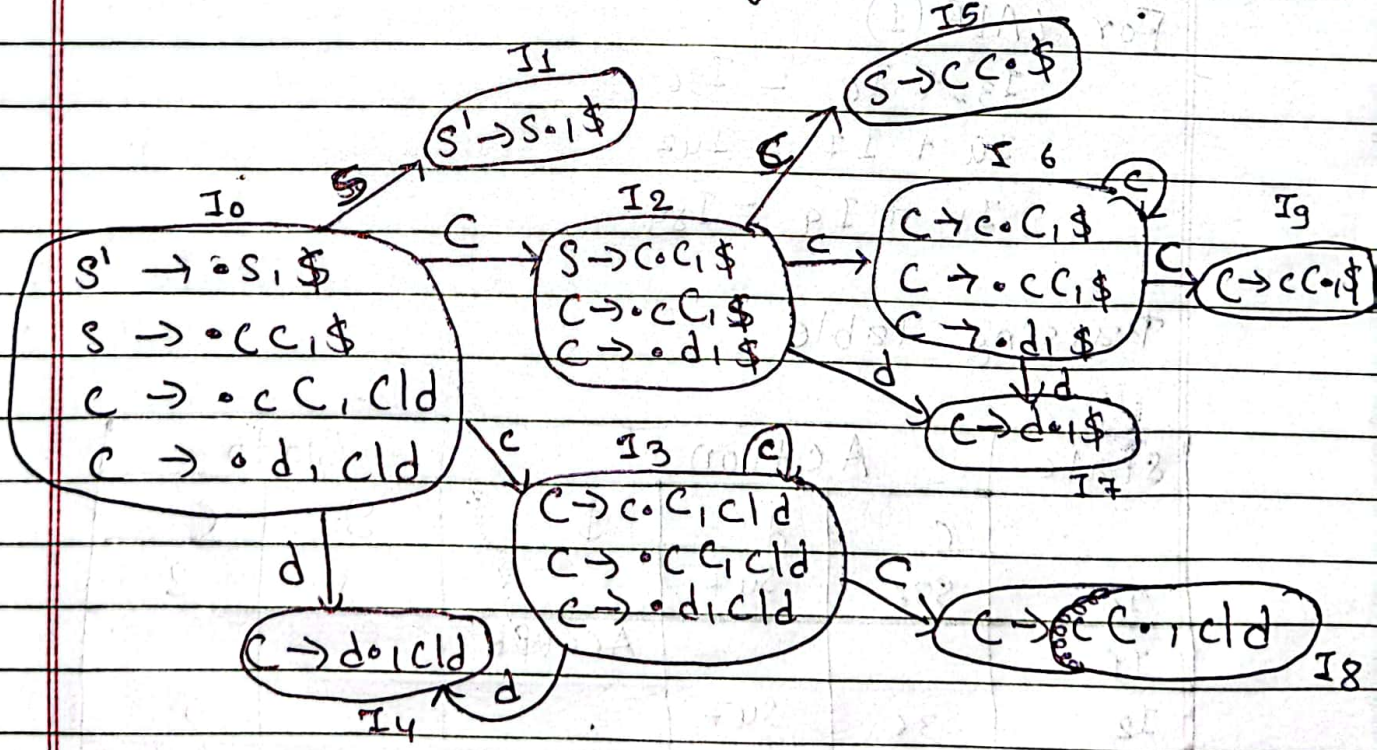
$S' \to \cdot S, \$$

$S \to \cdot CC, \$$

$C \to \cdot cC, c/d$

$C \to \cdot d, c/d$

Construct the canonical form for CLR(1) items



Parsing Table

State	Action			Go to	
	c	d	$	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			
I_7					
I_8	R_2	R_2			
I_9			R_2		

For LALR(1)

$I_3 + I_6 = I_{36}$

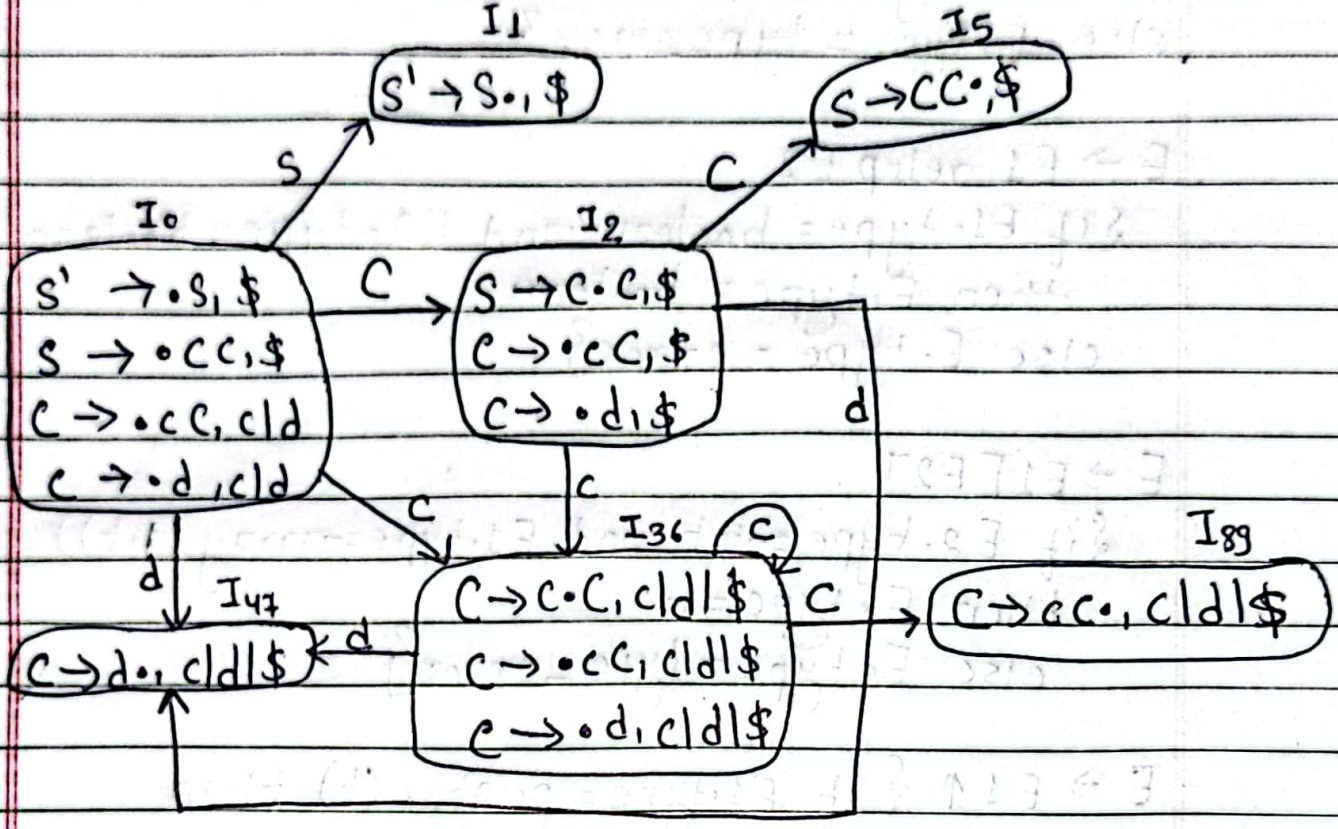
$I_4 + I_7 = I_{47}$

$I_8 + I_9 = I_{89}$

Parsing Table

State	Action			Go to	
	c	d	$	S	C
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		

Draw canonical collection for LALR (1) items:



16. Write SDD to carry out type checking for the following expression.

E → id | E1 op E2 | E1 relop E2 | E1 [E2] | E1↑

Solution,

E → id {E.type = lookup (id.entry)}

~~E → E1 op E2 {E.type = (E1.type == E2.type) ? E1.type : type-error)}~~

E → E1 op E2 {if (E1.type = integer and E2.type = integer) then E.type = integer
else if (E1.type = integer and E2.type = real) then E.type = real
else if (E1.type = real and E2.type = integer) then E.type = real
else if (E1.type = real and E2.type = real)

then E.type = real
else E.type = type_error }

E → E1 relop E2
{ if E1.type = boolean and E2.type = boolean
then E.type = boolean
else E.type = error }

E → E1 [E2]
{ if E2.type == int and E1.type == array (1, t))
then E.type = t
else E.type = type_error }

E → E1 ↑ { if E1.type = pointer (t) then
E.type = t
else E.type = type_error }