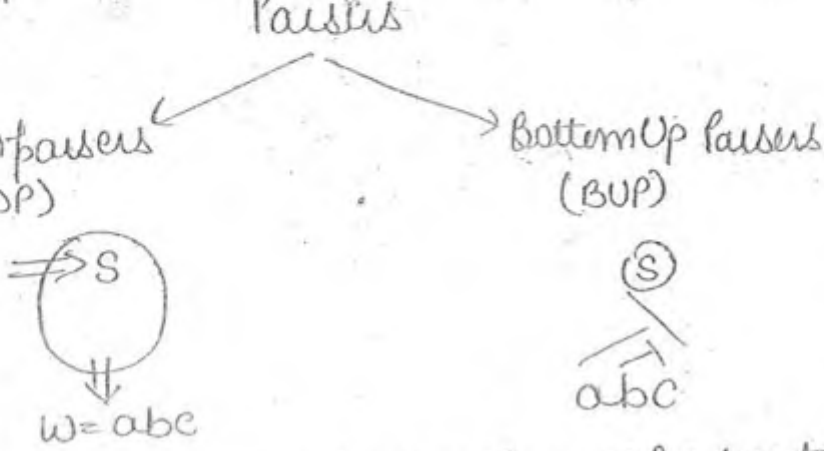
1. (char *(*)()) (*ptr[N])()
Array of N pointers to functions, that will return pointer to a function which will return pointer to a character pointer.

* Chapter No 2 *
Parsing

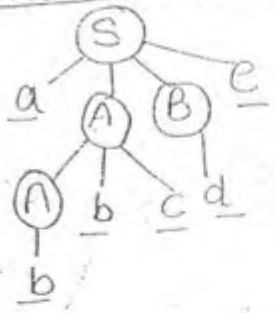
ϕ Type of parsers



TDP :- In TDP, we will derive the given string by taking the start symbol.
BUP :- In BUP, we will take the string and finally we will get the start symbol.

Working of TDP ⇒ TDP follow the left most derivation.

Ex :- S → aABe
A → Abc/b
B → d
w = abbcde
↑ look ahead symbol.

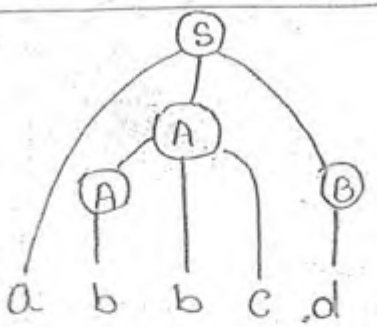


TDP, only sees one symbol at a time.
If there are two possibilities, choosing the right one is difficult.

Note 1 :- In TDP we are using left most derivation.
Note 2 :- The difficulty in TDP, is that if a variable is having more than one possibility, choosing right production.

Working of Bottom-UP parser

Exp:-



$S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

$w = abbcde$

Note-1:- In Bottom up parsing, we are using reverse of Right most derivation to derive the string.

Note2:- The difficulty in bottom up parsing finding the substring (handle), which will give our required variable, so that we will go to start symbol.

Top Down Parsers

TDP (No-LR, No-LF)

- With Backtracking → Brute force method
- Without backtracking → predictive parsers
  - Recursive Decent Parsers
  - Non-Recursive Decent Parsers (LL(1))

LL(1) Parser

L = left-right (reading the I/P symbol)

L = using left most derivation

1 = taking 1 symbol at a time

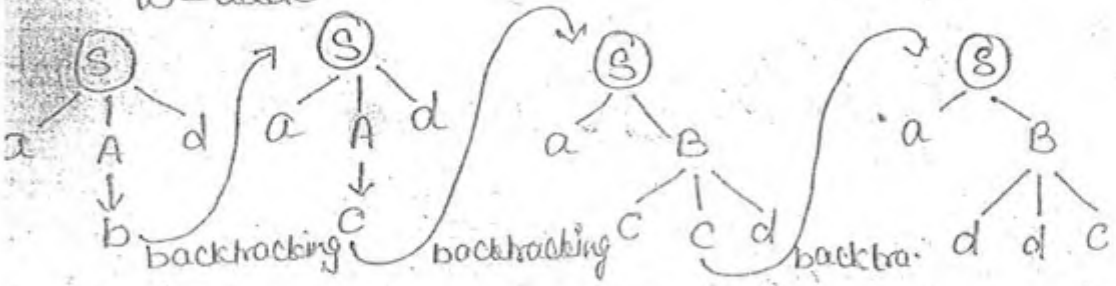
LR(1) Parser

L = left-right (reading the I/P symbol)

R = using right most derivation, taking one symbol at a time

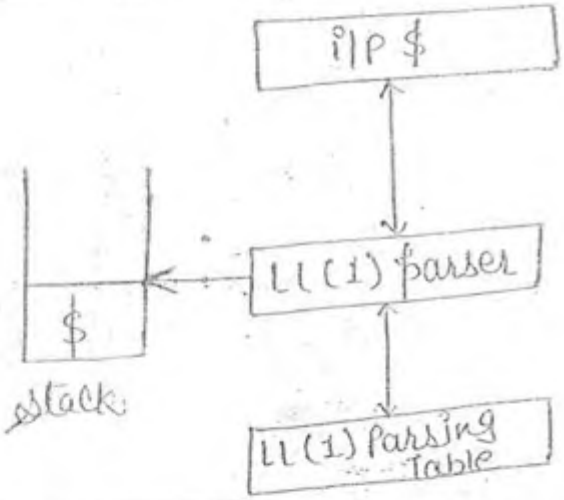
Method

S → aAd | aB
A → b | c
B → ccd | ddc
w = addc



Drawback:- It will take a lot of time in backtracking. It is not good for debugging.

Non-Recursive Decent Parsers:-



LL(1) Parsing Algo

If X is top of the stack & 'a' is a lookahead symbol-

1. If ((X == a) == $) then, successful completion.
2. If ((X == a) ≠ $), pop out from the stack, increment i/p pointe
3. If (X is non terminal) then use LL(1) parsing table entry M[X,a]
   If M[X,a], if X → UVW, replace X by UVW in the reverse order
   If M[X,a] = blank space, indicate error.

Exp:
E →
T →
F →
0 = 1

M:

E
E'
T
T'
F

Ren
E -
E' -
T -
T' -
F →

IN
dol

Exp:—

$E \to E+T \mid T$
$T \to T*F \mid F$ ⇒ Exp(1)
$F \to id \mid (E)$

$w = id + id * id \$$

LL(1) Parsing table

M	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'		T'→ϵ	T'→*FT'		T'→ϵ	T'→ϵ
F	F→id			F→(E)		

⇒ Size of LL(1) Parsing table.

$m * (n+1)$

m = no. of variables in grammar
n = no. of terminals in grammar
* always we are doing guess, because we want left most derivation.

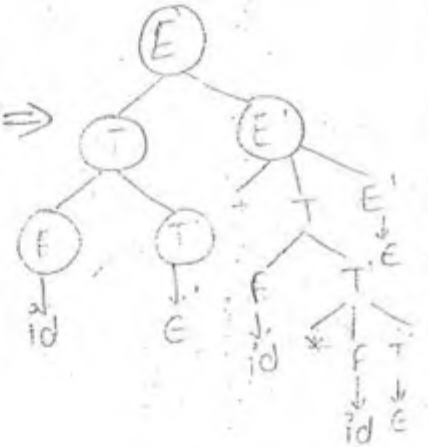
Removal of left recursion

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

⇒

E→TE'	T→FT'	T'→ϵ
T→FT'	F→id	E'→ϵ
F→id	T'→*FT'	
T'→ϵ	F→id	
E'→+TE'		

⇒



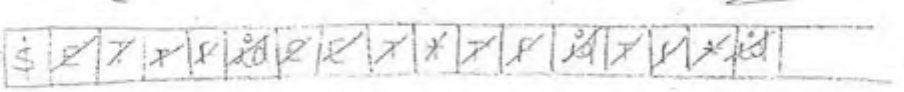
N. $w = (id + id * id) \$$ ⇒ Exp(2)

Total productions =

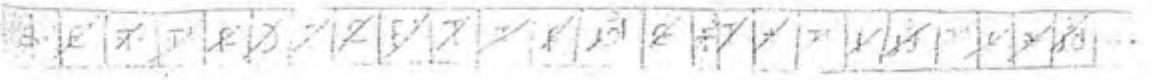
E→TE'	T'→ϵ	T'→ϵ
T→FT'	E'→+TE'	E'→ϵ
F→(E)	T→FT	T'→ϵ
E→TE'	F→id	E'→ϵ
T→FT'	T'→*FT	
F→id	F→id	

Ans

Exp-1 stack



Exp-2 stack:



N. $S \to (L) \mid a$
$L \to L,S \mid S$

$W = (a,a,a)$

	a	,	(	)	$
S	$S \to a$		$S \to (L)$		
L	$L \to SL'$		$L \to SL'$		
L'		$L' \to ,SL'$		$L' \to \epsilon$	

Parsing table

Sol^n

Q^n $S \to (L) \mid a$
$L \to SL'$
$L' \to \epsilon \mid ,SL'$

$S \to (L)$	$L' \to ,SL'$	$L' \to \epsilon$
$L \to SL'$	$S \to a$	
$S \to a$	$L' \to ,SL'$	
	$S \to a$	

Construction of LL(1) Parsing table

> First(α), gives a set of terminals, that begin in strings derived from α.

Exp:- $\alpha \to abc \mid def \mid Ab$
$A \to e$

gives by $\alpha' \Leftarrow \begin{bmatrix} a \Leftarrow abc \\ d \Leftarrow def \\ e \Leftarrow eb \end{bmatrix}$

Rule-1:- First(α) = α, if α is terminal

Rule-2:- First(ϵ) = ϵ

Rule-3:- $\alpha \to x_1 x_2 x_3$

First(α) = First($x_1 x_2 x_3$)
= First(x_1)
= First(x_1) - ϵ $\cup$ First(x_2, x_3)

only if, $x_1 \to \epsilon$

Exp:- $x_1 \to a \mid \epsilon$
$x_2 \to b \mid \epsilon$
$x_3 \to c \mid \epsilon$

First(x_1, x_2, x_3)
⇓
First(x_1)
a $\cup$ First(x_2, x_3)
⇓
First(x_2)
⇓
b $\cup$ First(x_3)
⇓
c, ϵ

N Find the first for the following grammar-
$E \to TE'$
$E' \to \epsilon \mid +TE'$
$T \to FT'$
$T' \to \epsilon \mid *FT'$
$F \to id \mid (E)$

DN.
S →
B →
C →
D →
E →

DN.
S → A
A → d
B →
C →

DN.
S → A
A → c
B → d

Foll
Follo
o th
rule1:-
rule2:-

ution	First()
E	id, C
E'	ϵ, +
T	id, C
T'	ϵ, *
F	id, C

IN. Find the first for the following grammar:-

S → aBDh First(S) = {a}
B → cC First(B) = {c}
C → bC | ϵ First(C) = {b, ϵ}
D → EF First(D) = {g, f, ϵ}
E → g | ϵ First(E) = {g, ϵ}
F → f | ϵ First(F) = {f, ϵ}

IN. Find the first for the following grammar:-

S → ACB | CbB | Ba First(S) = d, g, h, ϵ, b, a
A → da | BC First(A) = d, g, h, ϵ
B → g | ϵ First(B) = {g, ϵ}
C → h | ϵ First(C) = {h, ϵ}

IN. Find the first for the following grammar:-

S → AaAb | BbBa First(S) = c, a, d, b
A → c | ϵ First(A) = c, ϵ
B → d | ϵ First(B) = d, ϵ

Follow():→ Follow(A) ← variable.

Follow(A) gives set of all terminals, that may follow immediately to the right of A.

Rule 1:- If A a start symbol, then

Follow(A) = $

Rule 2:- If X → αAβ is in G:-

then, Follow(A) = First(β)

Rule-3: If $X \to \alpha A$ (or) $X \to \alpha A\beta$, $\beta \to \epsilon$

Follow(A) = Follow(X)

Q1. Find first and follow for the following grammar:-

$X \to aABe$

$B \to c|d$

$A \to a$

	First()	Follow()
X	a	$
B	c, d	e
A	a	c, d

Q2. Find first and follow for the following grammar:-

$E \to TE'$

$E' \to \epsilon | +TE'$

$T \to FT'$

$T' \to \epsilon | *FT'$

$F \to id | (E)$

	First	Follow
E	id, (	$,)
E'	ε, +	$,)
T	id, (	$,), +
T'	ε, *	+, $,)
F	id, (	*, +, $,)

Q3. Find the first and follow of the following grammar-

$S \to aBDh$

$B \to cC$

$C \to bC | \epsilon$

$D \to EF$

$E \to g | \epsilon$

$F \to f | \epsilon$

	First	follow
S	a	$
B	c	g, f, h
C	b, ε	g, f, h
D	g, f, ε	h
E	g, ε	f, h
F	f, ε	h

Q4. $S \to (L) | a$

$L \to SL'$

$L' \to \epsilon | ,SL'$

	First	Follow
S	(, a	$, ,,)
L	(, a	),
L'	ε, ,	)

LL1 Table Construction Algo.

For each production, $A \to \alpha$
repeat following two steps:-

1) Add $A \to \alpha$, under $M[A, b]$
   where, $b \in First(\alpha)$
2) If First(α) contain ϵ, then
   add $A \to \alpha$, under $M[A, c]$
   where, $c \in Follow(A)$.

IN Construct LL(1) parsing table for the following grammar,
$S \to (L) | a$
$S \to$

LL(1) table conversion algo

For each production, $A \to \alpha$, repeat, following two steps:-

1) Add $A \to \alpha$, under $M[A, b]$
   where, $b \in First(\alpha)$
2) If, first(α) contain ϵ, then
   add $A \to \alpha$, under $M[A, c]$
   where, $c \in Follow(A)$.

IN Construct LL(1) parsing table, for the following grammar:-

IN $S \to (L) | a$
$L \to SL'$
$L' \to \epsilon | , SL'$

$\Longrightarrow$

$S \to (L) | a$
$L \to L, S | S$

{ actual grammar }

after removing left Recursion)

	(	)	a	,	$
S	$S \to (L)$		$S \to a$		
L	$L \to SL'$		$L \to SL'$		
L'		$L' \to \epsilon$		$L' \to , SL'$	

Given grammar is LL(1), because each entry of LL(1) parsing

parsing table contains maximum one entry.

QN Construct LL(1) parsing table – (or) given grammar is LL1 or not

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id \mid (E)$

Eliminate left recursion –

$E \to TE' \mid T$

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

$T \to F.T' \mid F$

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

$F \to id \mid (E)$

	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	$F \to id$			$F \to (E)$		

QN Check the following grammar LL(1) is not –

$S \to A$

$A \to aB \mid Ad$

$B \to b$

$C \to g$

⇓

$S \to A$

$A \to aBA' \mid$

$A' \to \epsilon \mid dA'$

$B \to b$

$C \to g$ ⇒ LL(1) grammar

	a	b	d	g	$
S					
A					
A'			$A' \to dA'$		$A' \to \epsilon$
B					
C					

QN Check the following grammar is LL(1) or not :–

$S \to AaAb \mid BbBa$

$A \to \epsilon$

$B \to \epsilon$

	a	b	$
S	$S \to AaBb$	$S \to BbBa$	
A	$A \to \epsilon$	$A \to \epsilon$	
B	$B \to \epsilon$	$B \to \epsilon$	

QN. Find the grammar is LL(1) or not-

$S \to AaAb \mid BbBa$

$A \to b$

$B \to a$

⇓

LL(1)

	a	b	$
S	S→AaAb	S→BbBa	
A		A→b	
B	B→a		

Note 1:-

$\to \alpha_1 \mid \alpha_2 \mid \alpha_3$ ⇒ If this is the given form of grammar, then that grammar is said to be LL(1) only if-

$$\begin{aligned} &First(\alpha_1) \wedge First(\alpha_2) = \phi \\ &First(\alpha_1) \wedge First(\alpha_3) = \phi \\ &First(\alpha_2) \wedge First(\alpha_3) = \phi \end{aligned}$$

⇒ pairwise mutually disjoint

Note 2 If the grammar is in the form of -

$A \to \alpha_1 \mid \alpha_2 \mid \in$, Grammar is called LL(1) only if-

$$\begin{aligned} &First(\alpha_1) \wedge First(\alpha_2) = \phi \\ &First(\alpha_1) \wedge Follow(A) = \phi \\ &First(\alpha_2) \wedge Follow(A) = \phi \end{aligned}$$

⇒ pairwise mutually disjoint

QN. Check the following grammar is LL(1) or not - (GATE)

$\to E \mid a$

$\to a$

Ans $S \Rightarrow first(E) \wedge first(a)$

$a \wedge a = a$ not (LL1) Ans

QN. Check the following grammar is LL1 or not-

$S \to aABb$ — A — B

$A \to a \mid \in$ — $first(a) \wedge follow(A)$ — $first(a) \wedge follow(B)$

$B \to d \mid c$ — $(a, \in) \wedge$ — $d \wedge c$

Given grammar is LL(1) grammar. Ans

QN. $S \to aSA \mid \epsilon$
$A \to c \mid \epsilon$

Sol^n S
$first(A) \wedge follow(A)$
$a \wedge c, \$ = \phi$

A
$first(A) \wedge follow(A)$
$c \wedge \$, c = c$

This grammar is not LL(1) grammar. Ans

QN. Check the following grammar is LL(1) or not,
$S \to AB$
$A \to a \mid \epsilon$
$B \to b \mid \epsilon$

Sol^n A
$first(A) \wedge follow(A)$
$a \wedge b, \$ = \phi$

B
$first(b) \wedge follow(b)$
$b \wedge \$ = \phi$

This grammar is LL(1) grammar. Ans

QN. Given grammar is LL(1) or not-
$S \to (L) \mid a$
$L \to L, S \mid S$

Sol^n S
$first(a) \wedge follow$
$(\wedge a = \phi$

$first(L, S) \wedge follow$
$(, a$

Note:- Any left recursive grammar is not LL(1).

Note-2:- Any grammar which contain left factoring is not LL(1)

QN. Construct LL(1) parsing table for the following grammar:
begram $\to$ begin^{①} d Semi X end
$X \to$ d^{②} Semi X $\mid$ S^{③} Y
$Y \to$ Semi^{④} S Y $\mid \epsilon^{⑤}$

	Semi	begin	d	end	S	$
pgram		①				
X			②		③	
Y	④			⑤		

⇒ LL(1) grammar

Q. Find first and follow for the following grammar-

$E \to aA \mid (E)$

$A \to +E \mid *E \mid \epsilon$

Sol^n

	first	follow
E	a, (	), $
A	+, *, ε	), $

Q. Which one of the following is true :-

$E \to E*F \mid E+F \mid F$

$F \to F-F \mid id$

1) * has higher precedance than +.

2) ✓ *

3) *, - has same precedance

4) + has higher precedance.

Sol^n ϕ

Recursive Decent Parsers

Exp:- $E \to E+T \mid T$

$T \to T*F \mid F$ ⟹ Eliminate left Recursion

$F \to id$

$E \to TE'$

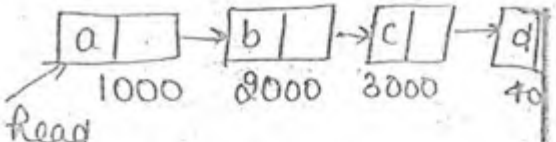
$E' \to \epsilon \mid +TE'$

$T \to FT'$

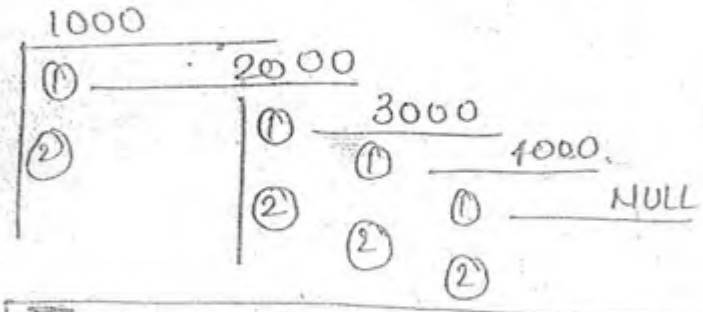
$T' \to \epsilon \mid *FT'$

$F \to id$

```
E()          E'()          T()          T'()
{            {             {            {
  E()          E'()          T()          T'()
}            }             }            }
```

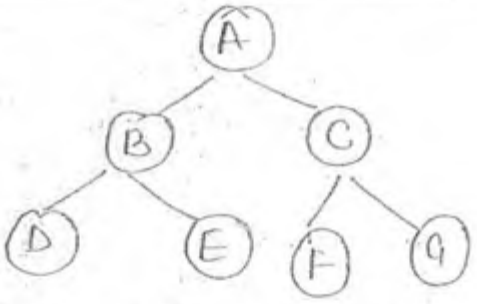



```
Read
Return(head)
{
  if(head==Null)
    return;
  else
  {
    (1) Reverse(head->next)
    (2) printf(Head->data);
  }
}
```

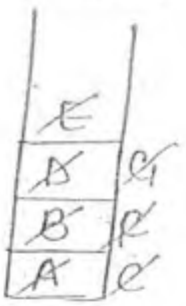


Non Recursive pgm → stack at the place of recursion

Preorder without Recursion



Stack



A, B, D, E, C, F, G

Note:- In recursive decent paiser, we will write the recursive pgm. for every variable, such that if any variable contains more than one possibility, it will choose the correct production.

Bottom-Up Paiser

Bottom-up Paiser (Shift-Reduce Paisers).

- Operator Precedance Paiser
  - Operator Grammar ⇐ Ambiguous, Unambiguous
- LR-Paiser
  - applied only on unambiguous grammar.

LR-Parsers

- LR(0) — finally do parsing (less powerful) — less power
- SLR(1) — Simple LR-parsers — less cost — less power
- CLR(1) — Canonical LR-parsers — more cost — powerful
- LAR(1) — Look Ahead LR Parsers — medium cost — Medium

* Bottom-up parsers are much powerful than top-down parsers, but designing is more complex.

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

Operator Precedance Grammar (Parsers)

Operator Grammar:-

ambiguous, unambiguous

Operator Precedance Parser

Grammar is said to be a operator grammar, if it satisfies following two conditions-

- No null productions
- No two variables side by side on R.H.S. of production

Exp:- E+E | E*E | id

Exp:- E → A+B
A → a
B → b | ∈ → α

Exp:- E → AB
A → a
B → b

α

ϕ Check the following grammar is operator grammar or not

$S \rightarrow SAS \mid a$

$A \rightarrow bSb \mid b$

This grammar is not operator grammar.

conversion into Operator grammer

H $S \rightarrow SbSbS \mid SbS \mid a \Rightarrow S \rightarrow SbSbS \mid SbS \mid a$

$A \rightarrow bSb \mid b$ remove A, useless production

H $P \rightarrow SR \mid S$

$R \rightarrow bSR \mid bS$

$S \rightarrow WbS \mid W$

$W \rightarrow L*W \mid L$

$L \rightarrow id$

$\Rightarrow$

$P \rightarrow SbP \mid SbS \mid S$

$R \rightarrow bP \mid bS$

$S \rightarrow WbS \mid W$

$W \rightarrow L*W \mid L$

$L \rightarrow id$

ϕ Operator Precedence Parser (Parsing algorithm):-

, If a is the top of the stack, then b is the look ahead sym

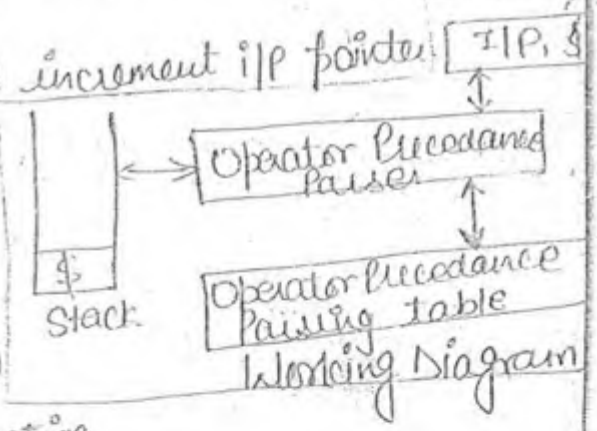
then-

1) If $a < b$ or $a = b$, then shift b and increment i/p pointer

2) If $a > b$, repeat-

{ pop out from the stack } until (top < recently popped out)

3) If $a = b = \$$, then successful completion.



ϕ:- Parsing table

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	

① $w = id + id * id \$$

$ id + id * id

id id id * + A#

② $w = id * id * id \$$

$ id * id * id

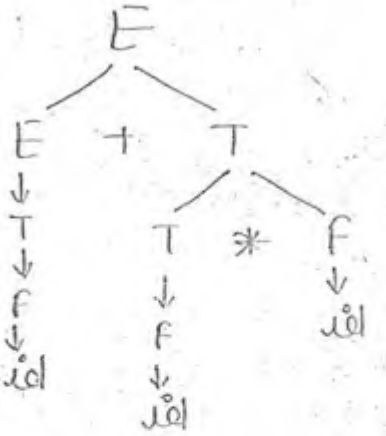
id id * id * A#

E → E+T | T
T → T*F | F
F → id

$ < id > + < id > * < id > $ ⑤ ② ④ ③

+ < *
+ > +
* > *

= 5 Handle



$	id	*	id	*	id	

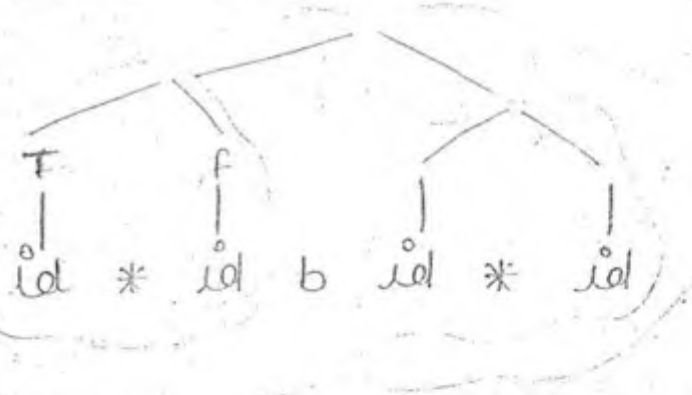
id → ① Handle * → 4 Handle
id → ② Handle + → 5 Handle
id → ③ Handle

DN Define the Operator precedance parsing table for, w=

w = id * id b id * id

Operators ⇒ *, b and * > b b < b

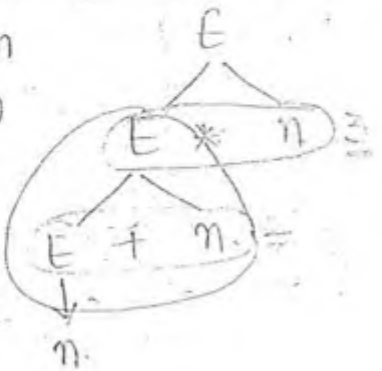
	id	*	b	$
id		>	>	>
*	<	>	>	>
b	<	<	<	>
$	<	<	<	



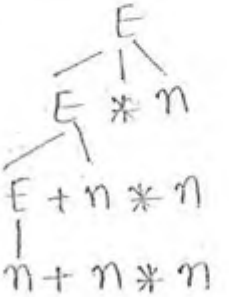
DN Consider the grammar-
E → E+n | E*n | n, for the I/P string,
w = n+n*n, the handels are :-

1) n, E+n and E+n*n
2) n, E+n and E+E*n
3) n, n+n and n+n*n
4) n, E+n and E*n

Parse Tree



1st Handle = n



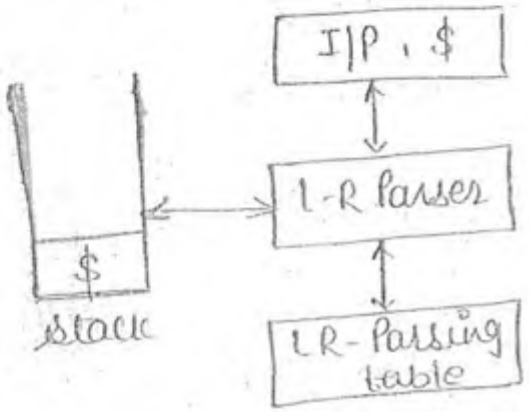
L-R Parsing Algo *

If S- state on the top of the stack and a- lookahead symbol, then-

1. If action [S, a] = Si, then shift a and i and increment the i/p pointer.
2. If action [S, a] = Rj, and if Rj is, $\alpha \rightarrow \beta$, then pop $2*|\beta|$ sym from the stack and replaced by α.

If Sm-1 is the state below α, then push Goto [Sm-1, α].

3. If action [S, a] = acc, then successful completion.



Note:- For all LR- parsers, parsing algorithm is same, parsing tables are different.

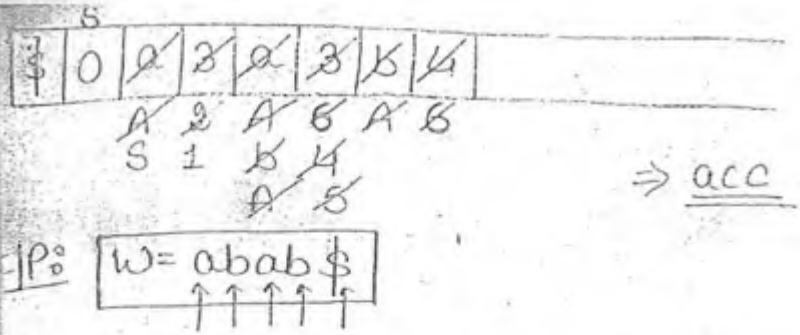
Exp:- $S \rightarrow AA$ ①
$A \rightarrow aA$ ② $| b$ ③

w = aabb$

LR-(0) Parsing table

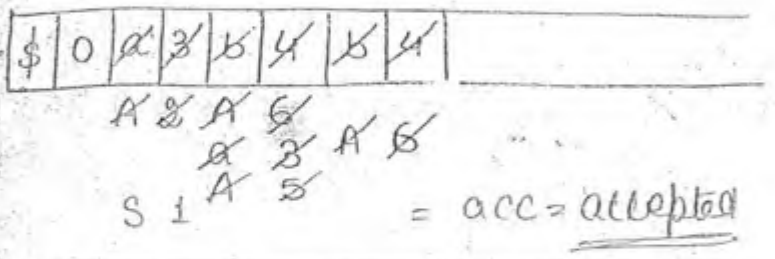
	a	b	$	S	A
	action			Goto	
0	S_3	S_4		1	2
1			acc		
2	S_3	S_4			5
3	S_3	S_4			6
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

Rows = State No.
Columns = Action | Goto



⇒ acc = accepted

Ex: w = abab$



= acc = accepted

r_3 A → b
|b| = 1 * 2 = 2

r_2 A → |aA|
= 2 * 5 = 4

r_1 S → AA
= 2 * 2 = 4

r_3 A → b
|b| = 1 * 2 = 2

Problem with Shift → Reduce ⇒ Action part

↳ because these entries are only on action part

Another parsers other than LR(0), has some minimization in reduction entries, because they can predict the next symbol.

Conflict: — Shift and reduce together S_j / r_i

Ex: S → AA
A → aA | b

↓ Augmented Grammar (added to know successful completion)

S' → S ⇒ S·S
S → AA
A → aA | b

Closure of an item

↓

applied only on item

Given Grammar

Ex:- S → AA
A → aA | b

Now, S' → ·S ⇒ item (closure)

1. S' → ·S
2. S → ·AA
3. A → ·aA
   · b

(i) S → A·A ⇒ closure (s)

S → A·A
A → ·aA
· b

$ Item

S → ·xyz (Item)
S → x·yz
S → xy·z
S → xyz· ⇒ (final production / Complete production / reduced production)

ϕ Finding closure of a variable

Closure (I) =

① Add item I to closure(I)

② If $A \to \alpha . B \beta$ is I and $B \to \gamma$ in G, then add $B \to . \gamma$ to closure of

③ repeat previous two steps. for every newly added production

Exp:- $S \to AA$

$A \to aA | b$

① $S \to . S$

② $S \to . AA$

③ $A \to . aA$

$. b$

ϕ Finding Goto (I, X) :-

Find the Goto of (I, X) by following method :-

Write the same item (I) as it is, by moving (.) after X.

Apply closure function on the result of step 1.

ϕ LR(0) Parsing table Construction

Step-1 Find the augmented grammar.

Step-2 Find the closure of augmented production

Step-3 Using closure of augmented production construct the dfa.

Step-4 Reduce dfa into table.

Exp:- Construct LR(0) Parsing table, for the following grammar:-

$S \to AA$ ①

$A \to aA | b$ ② ③

Sol"

① $S' \to S$
$S \to AA$
$A \to aA | b$

② I_0: $S' \to . S$, $S \to . AA$, $A \to . aA$, $A \to . b$

$I_0 \xrightarrow{S} I_1$: $S' \to S .$

$I_0 \xrightarrow{A} I_2$: $S \to A . A$, $A \to . aA$, $A \to . b$

$I_2 \xrightarrow{A} I_5$: $S \to AA .$

$I_0 \xrightarrow{a}$ / $I_2 \xrightarrow{a}$ / $I_3 \xrightarrow{a}$: $A \to a . A$, $A \to . aA$, $A \to . b$

$I_0 \xrightarrow{b}$ / $I_2 \xrightarrow{b}$ / $I_3 \xrightarrow{b}$: $A \to b .$ I_4

$I_3 \xrightarrow{A}$ I_6: $A \to aA .$

Construction of table

M	Action			Goto	
	a	b	$	S	A
0	S3	S4		1	2
1			acc		
2	S3	S4			5
3	S3	S4			6
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

* If in a Ij more than one production is completed, it will become **Reduce-Reduce Conflict**

* **Shift-Reduce Conflict = Reduce-Shift Conflict**

* **No Shift-Shift Conflict** ⇓ – because it is a d.f.a.

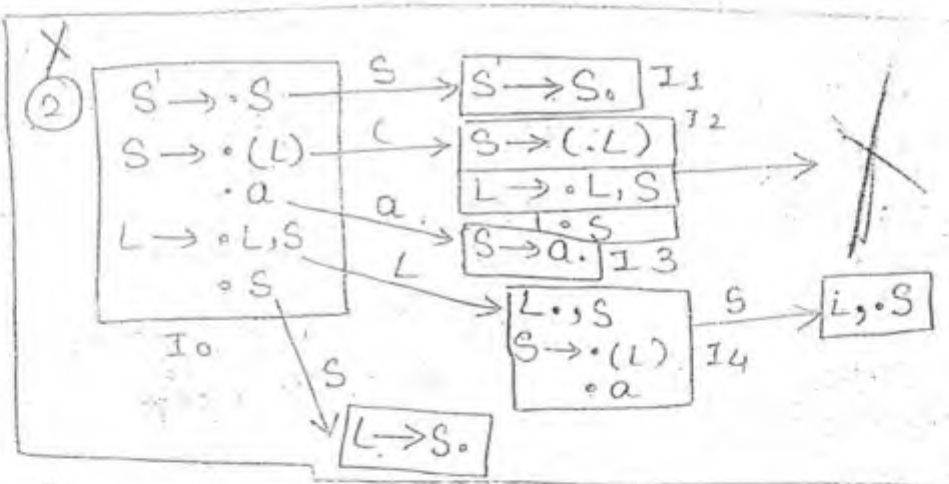
⇒ given grammar is LR(0) grammar, because no entry of this table contain more than one value AS

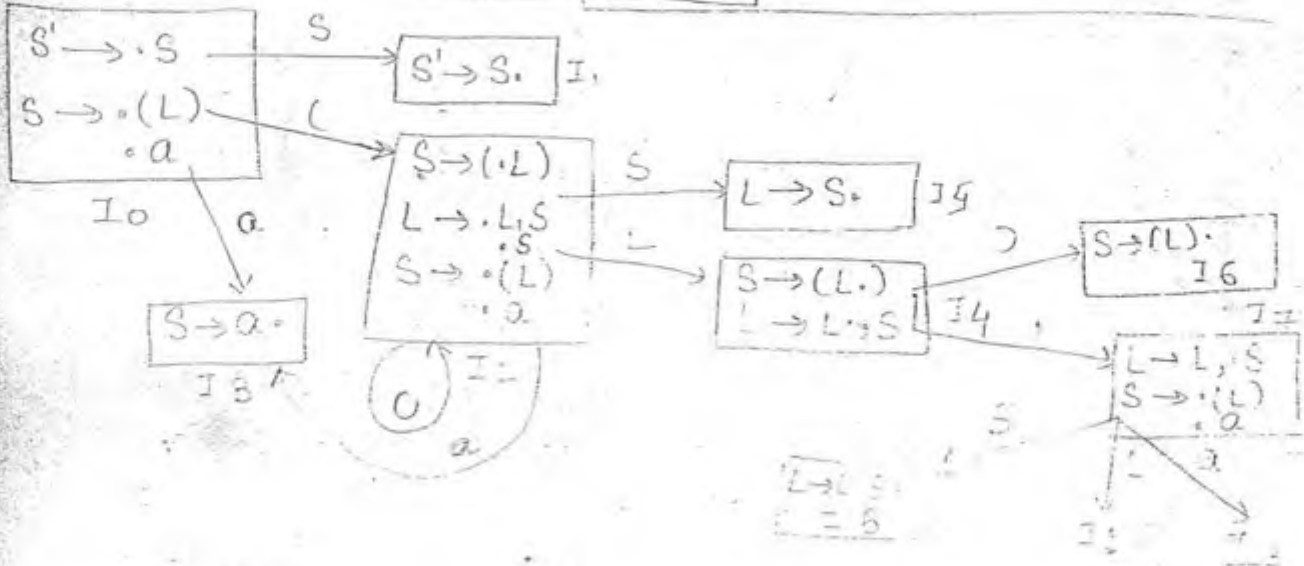
Note:- S-S Conflict is not possible, because given diagram is d.f.a.

Q1. Check the following grammar is LR(0) or not-

G → (L) | a
L → L, S | S

(1) S' → S
S → (L) | a
L → L, S | S





Parsing table

M	Action					Goto	
	a	(	)	,	$	S	L
0	S3	S2				1	
1					acc		
2	S3	S2				5	4
3	r2	r2	r2	r2	r2		
4				S6	S7		
5	r4	r4	r4	r4	r4		
6	r1	r1	r1	r1	r1		
7	S3	S2				8	
8	r3	r3	r3	r3	r3		

⇒ LR(0)

Q. Check the following grammar is LR or not:-

$S \to dA \mid aB$

$A \to bA \mid c$

$B \to bB \mid c$

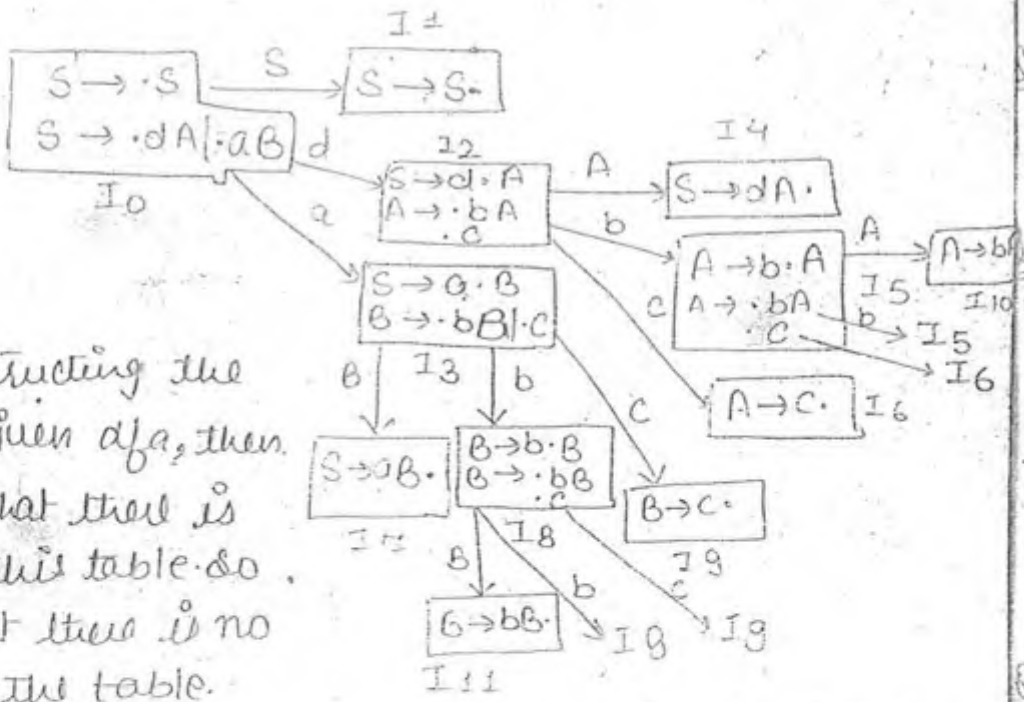
Sol^n

$S' \to S$

$S \to dA \mid aB$

$A \to bA \mid c$

$B \to bB \mid c$



If we are constructing the table, for the given dfa, then we will find, that there is no conflict in this table. So, it is clear, that there is no any conflict in the table.

⇒ There is no conflict in this given Grammar. So it is LR(0) Grammar.

States = 0 · 11

Action = d, b, c, a, $

Goto → S, A, B, $

Q. Check the following grammar is LR(0) or not:-

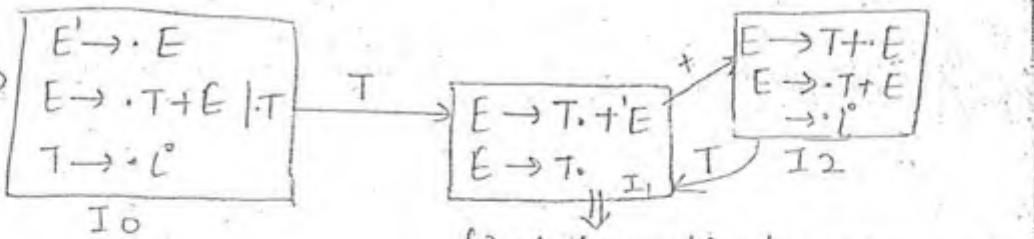
$E \to T+E \mid T$

$T \to i$

Ans $E' \to E$

$E \to T+E \mid T$

$T \to i$



final & Non final ⇒ SR conflict

Not LR(0) Ans

Exp:- $S \to A \cdot B$, $A \to b \cdot$ ⇒ final and non final but still conflict not

bcoz $S \to A \cdot B$ (action part) not ⇓ Go to part

$S \to b \cdot$ (purely action part)

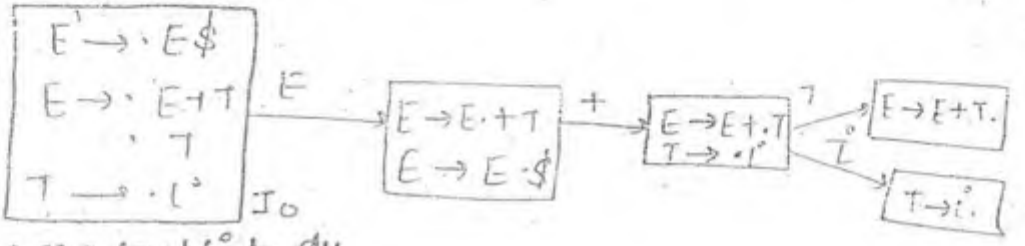
Q. $E \to E+T \mid T$

$T \to i$

$E' \to E\$$

$E \to E+T \mid T$

$T \to i$



It is LR(0) → no conflict Ans

Q. following grammar is LR(0) or not:-

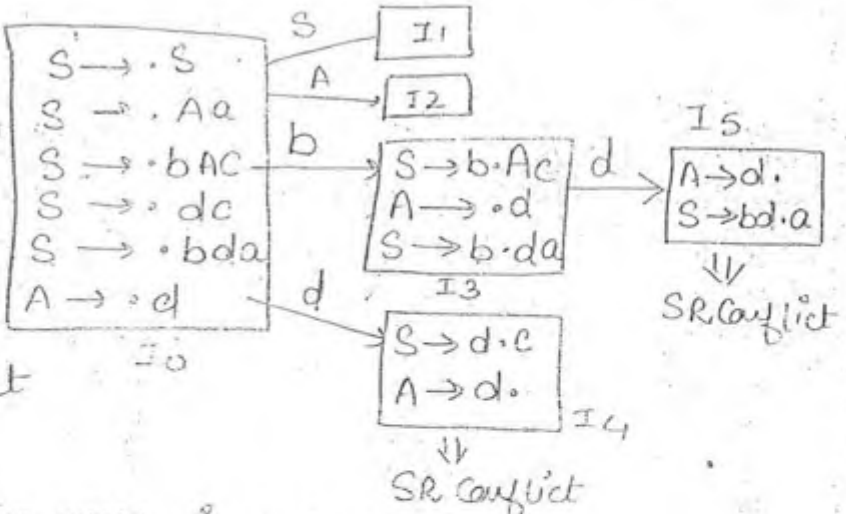
$S \to Aa \mid bAc \mid dc \mid bda$

$A \to d$

Ans $S' \to S$

$S \to Aa \mid bAc \mid dc \mid bda$

$A \to d$



This grammar is not LR(0) because two states are not acceptable.

Q. Check the following grammar is SLR(1) or not:-

Ans $S \to Aa \mid bAC \mid dc \mid bda$

$A \to d$

given grammar is not SLR(1) because, the conflicts which are there in LR(0) are not eliminated by SLR(1).

QN Check the following grammar is SLR(1) or not.

$E \rightarrow E+T \mid T$

$T \rightarrow TF \mid F$

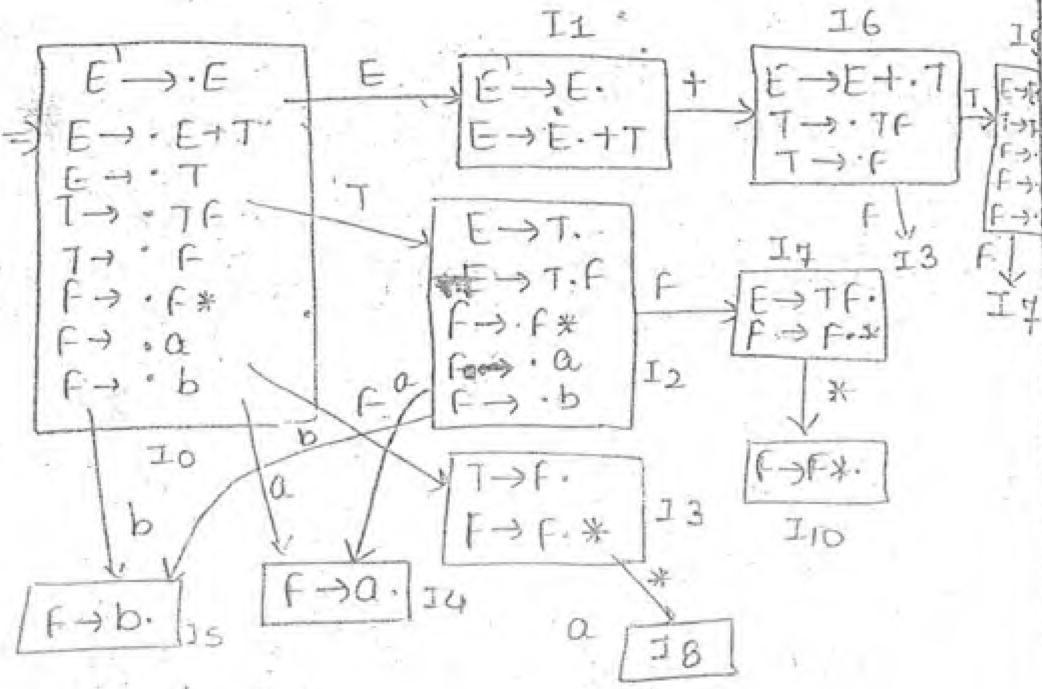
$F \rightarrow F* \mid a \mid b$

Soln $E' \rightarrow E$

$E \rightarrow E+T \mid T$

$T \rightarrow TF \mid F$

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



	a	b	*		$
2	r2/S4	r2/S5	r2	r2	r2
3	r4	r4	r4/S8	r4	r4
9	r1/S4	r1/S5	r1	r1	r1
7	r3	r3	r3/S10	r3	r3

⇒ not LR(0), just because of conflicts

⇓

Now remaining conflicts, apply SLR(1). Find follow of the completed entries and intersection with shift entries. If we got φ = no conflict and LR(1)

I_2 = R = + \$, S = a, b → φ

I_3, R = + \$ a, b, S = * → φ

I_7 + \$, a, b, * → φ

I_9, + , \$, a, b → φ

SLR(1) table

	a	b	*	+	$
2	S4	S5	r2	r2	r2
3	r	r	S	r	r
9	S	S		r	r
7	r	r	S	r	r

⇒ SLR(1) ⇒ all the conflicts of LR(0) are removed here.

1* Check the following grammar is SLR(1) or not:-

$S \to AaAb \mid BbBa$

$A \to \epsilon$

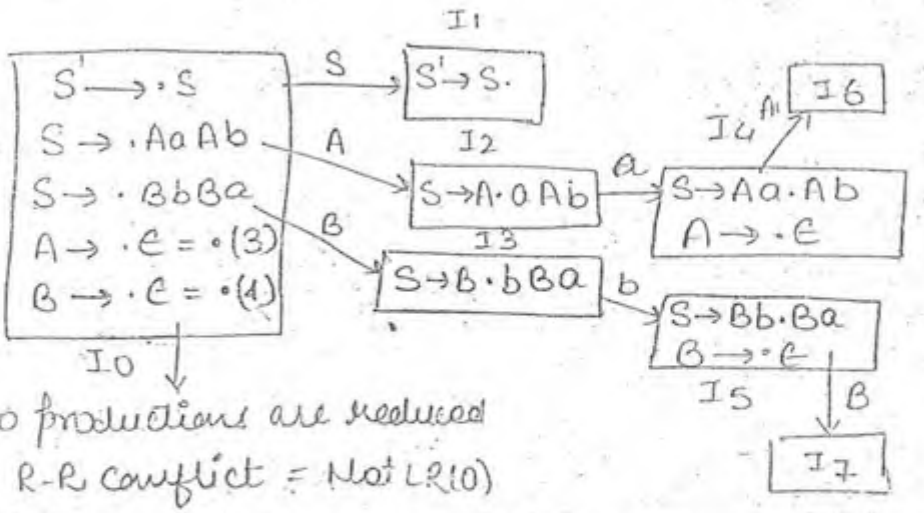
$B \to \epsilon$

Sol^n $S' \to S$

$S \to AaAb \mid BbBa$

$A \to \epsilon$

$B \to \epsilon$



Two productions are reduced so R-R conflict = Not LR(0)

Given grammar is not LR(0) because LR(0) is faulty state which will contain R-R conflict

→ Check for SLR(1)

follow(A) = b, a

follow(B) = a, b

	a	b
0	r3/r4	r3/r4

⇒ not SLR(1)

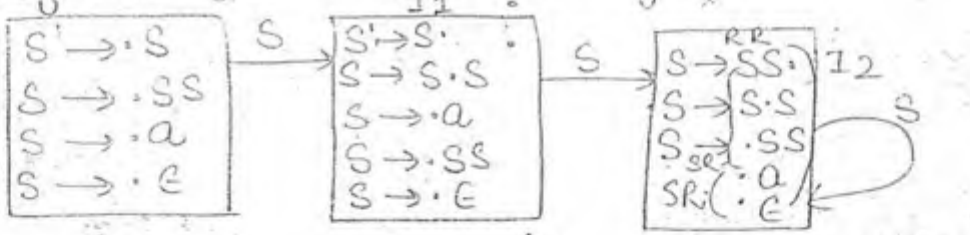
N Consider the following grammar-

$S \to SS \mid a \mid \epsilon$

Find the number of inadequate state in LR(1) grammar

Sol^n $S' \to S$

$S \to SS \mid a \mid \epsilon$



I0 S-R Conflict

S-R Conflict

SR, RR → Conflict

	a	b
2	r1/r3	r1/r3
2	r1/s3	r1
2	r3/s3	r3

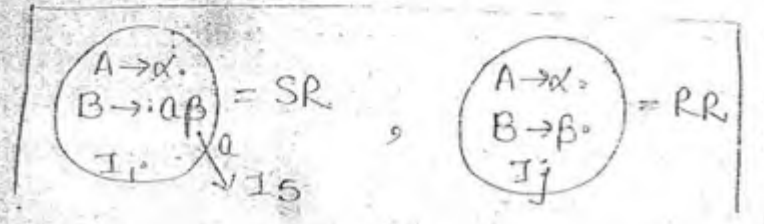
4 - SR

1 - RR

3 → I.S.

Φ CLR(1) Grammar:-

Conflicts in LR(0)



Ii°

	a	$
i	r1/s5	r1

Ij

	a	$
j	r1/r2	r1/r2

LR(0) → item

LR(0) — SLR(1)

⇓

LR(1) item

⇓

LR(0) item + 1 - look ahead symbol.

LR(1) item

CLR(1) — LALR(1)

DH. $S \to AA$

$A \to aA|b$

⇓

$S' \to .S, \$$

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

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

(2) $S \to .A(A, \$)$

(3) $A \to .aA, a|b$

$.b, a|b$

Find the closure of LR(1) item

Closure of Item (I) =

1) Add the same LR(1) item. $(S' \to .S, \$)$

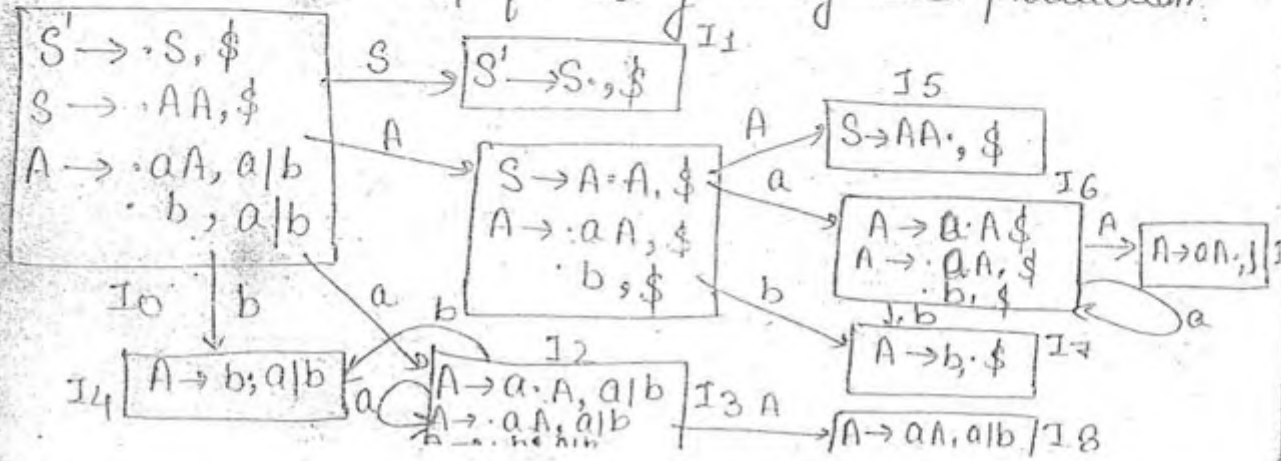
2) If $A \to \alpha.B(\beta, \$)$ is LR(1) item I,

and $B \to C$ is in G.

then

add $B \to .C$, first $(\beta, \$)$ to closure of LR(1) item I.

3) Repeat the second step for every newly added production.



The given grammar is CLR(1), because no state contain conflicts.

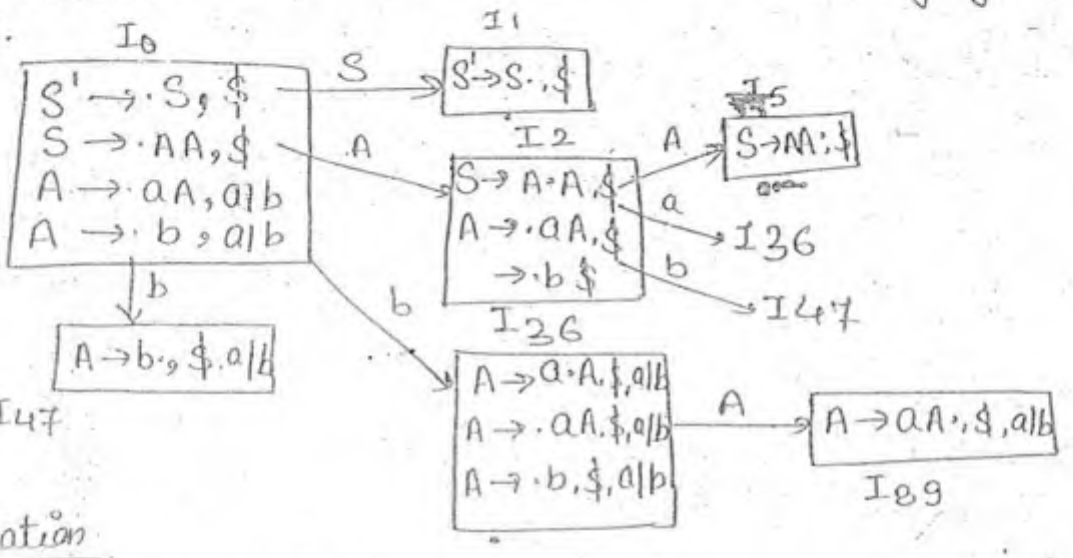
Relation b/w the states of LR(0), SLR(1), CLR(1)-

LR(0)	CLR(1)	SLR(1)	LALR(1)
⇓	⇓	⇓	⇓
n_1	n_3	n_2	n_4

$n_1 = n_2 \leq n_3$ $\quad$ $n_1 = n_2 = n_4 \leq n_3$

LASLR(1) Parsers

It will combine the two states, which are differe only by lookahead symbol.



minimization

Q1. Check the following grammar is CLR(1) or not -

S → AaAb/ BbBa

A → ∈

B → ∈

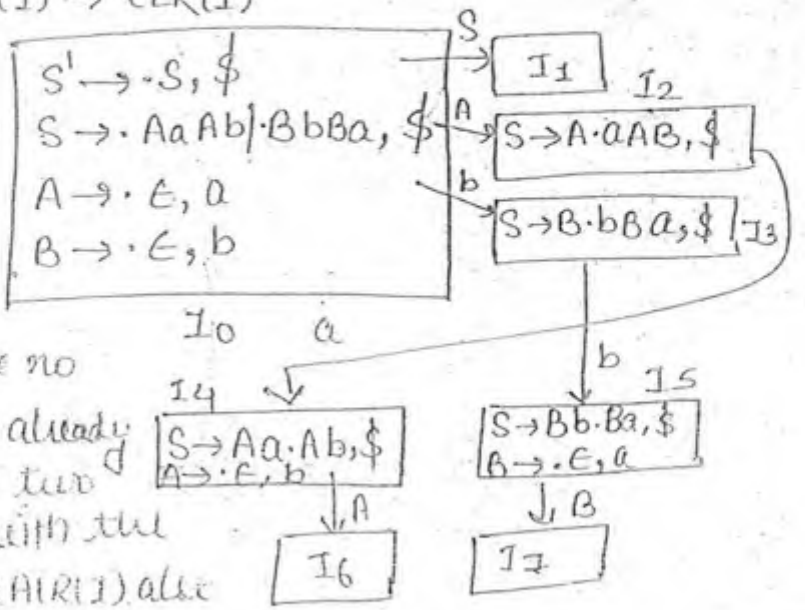
Soln LL(1) → LALR(1) → CLR(1)

S' → S

S → AaAb/BbBa ⇒

A → ∈

B → ∈



The given grammar is CLR(1) grammar because no conflicts. The given dfa is already minimized, because no two states are differed with the lookahead symbol (LALR(1)) also.

11) Check the following grammar is CLR(1) or not →

$S \to AA / bAC / dC / bda$

$A \to d$

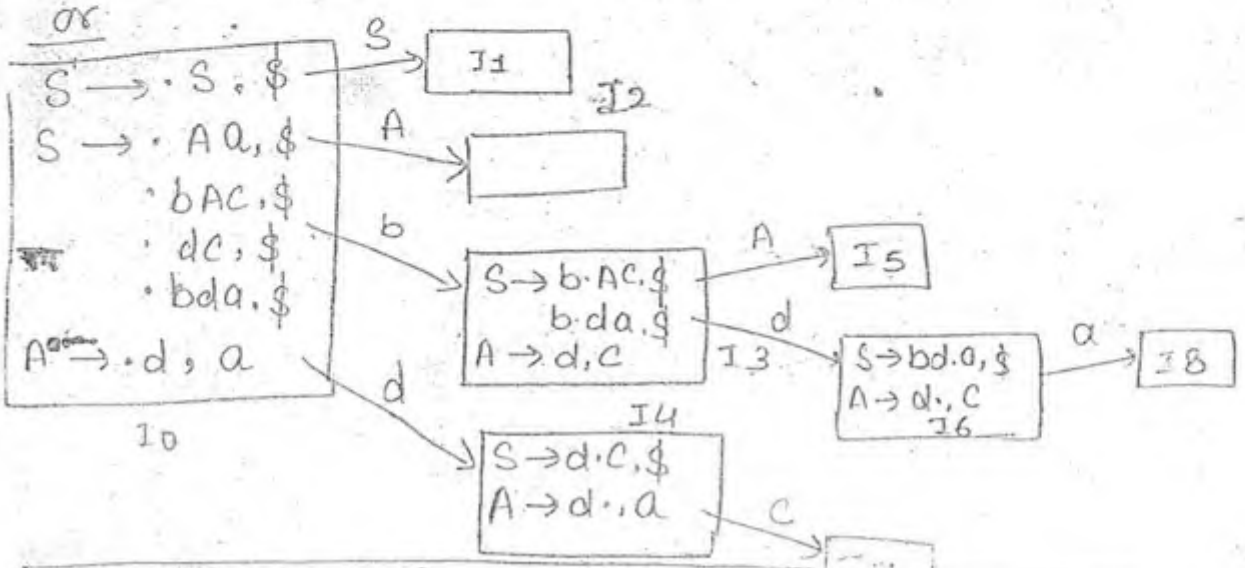
Sol^n $S' \to S$

$\to AA / bAC / dC / bda \Rightarrow S' \to \cdot S, \$$

$S \to \cdot AA / \cdot bAC / \cdot dC / \cdot bda, \$$

$A \to \cdot d, d$

or



no conflict (CLR(1)) = LALR(1) = already minimized. du

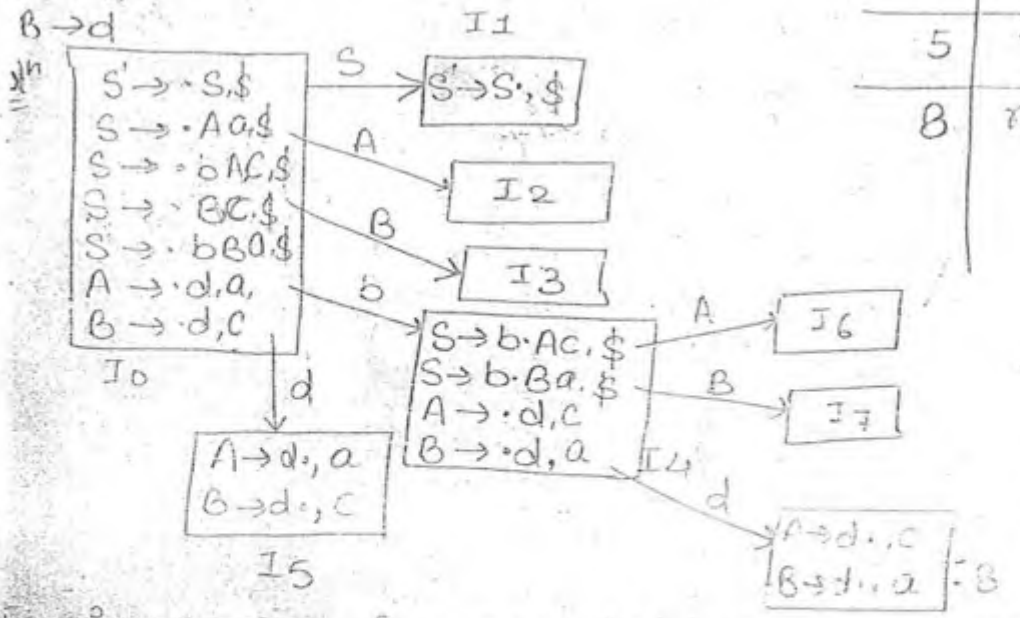
11. Check the following grammar is LALR(1) or not -

$S \to Aa / bAC / BC / bBa$

$A \to d$

$B \to d$

Sol^n



	a	c
5	r5	r6
8	r6	r5

LALR(1)

I56

A → d · , a/c

A → d · , a/c

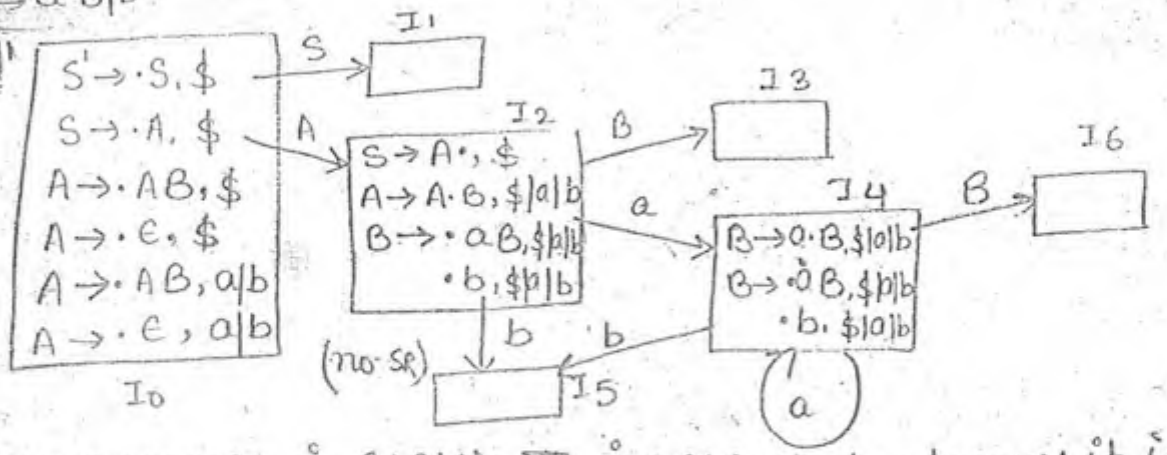
	a	c
56	r5, r5	r5, r6

The given grammar is CLR(1), because of no conflicts The given gram ar is not LALR(1), because after combining states 5 and 8 then is conflict

Q1 Check the following grammar is LALR(1) or not-

$S \to A$
$A \to AB \mid \epsilon$
$B \to aB \mid b$

Sol'

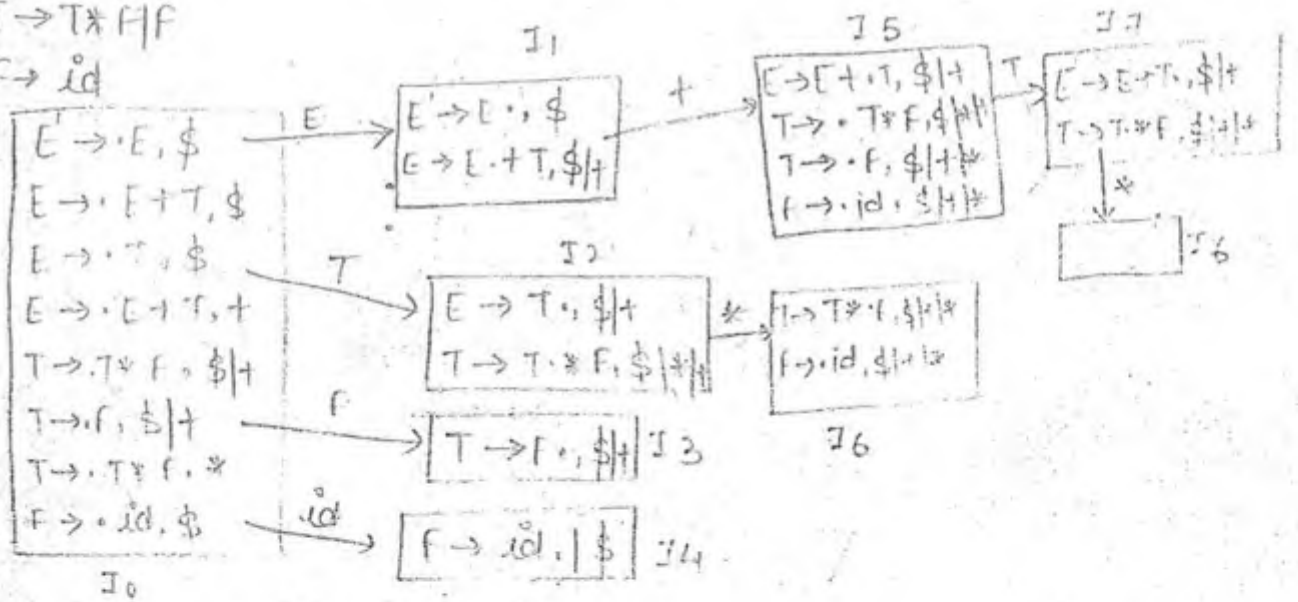


Given grammar is CLR(1). It is LALR(1) also, because it is already minimized.

Q2 Check the following grammar is LALR(1) or not-

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

Sol'n



It is CLR(1). It is LALR(1) also

GATE QUESTIONS

Q1 Consider SLR(1) and LALR(1) table for the given CFG:-
Which of the following is true-

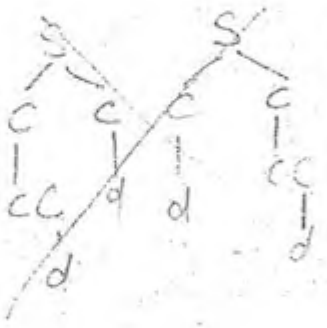
(A) GO TO of both the tables may be different
(B) Shift entries are identical in both the tables
(C) Reduce entries in both the tables may be different
(D) None of above.

1.2 Let, SLR(1) parsing table will take n_1 rows, LALR(1) will take n_2 rows, relation b/w n_1 and n_2 :-

$$n_1 == n_2$$ Ans

a) n_1
b) n_1
c) n
d) n_1

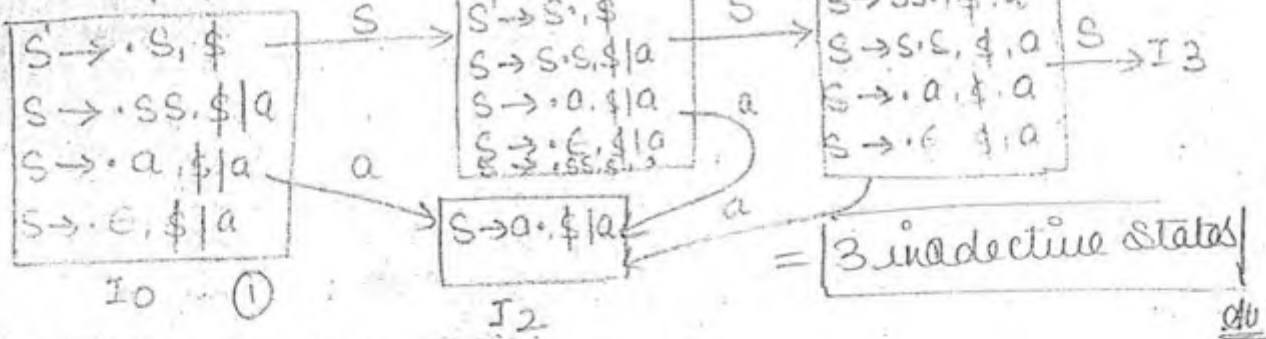
1.3 Consider the following CFG-
$S \to CC$
$C \to cC \mid d$
on which one of the following is true:-
LL(1)
SLR(1) not LL(1)
LALR(1) not SLR(1)
CLR(1) not LALR(1).



S
S.
S:-

LL(1) → surely LALR(1)
⇓
have to check for LALR(1)

1.4 Find the number of inadequate states while constructing CLR(1) parser-
$S \to SS \mid a \mid \epsilon$



= 3 inadequate states

1.5 Consider the following grammar:-
$A \to AA \mid (A) \mid \epsilon$ is not suitable for Operator precedence parser bcz
ambiguous
left recursion
right recursion
none of the above (operator Grammar)

1.6 Consider the following grammar-
$S \to (S) \mid a$
SLR(1) → n_1
LR(1) → n_2
LALR(1) → n_3

$$n_1 = n_3 \leq n_2$$

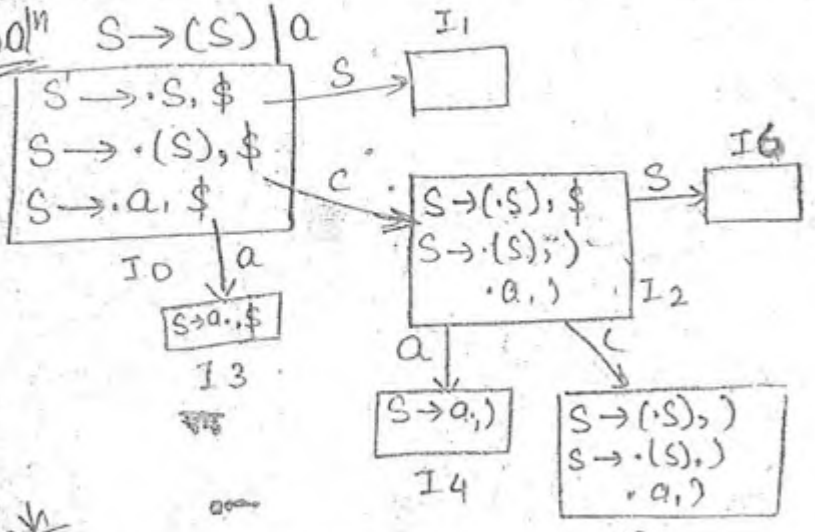
a) $n_1 < n_2 < n_3$

b) $n_1 = n_3 < n_2$

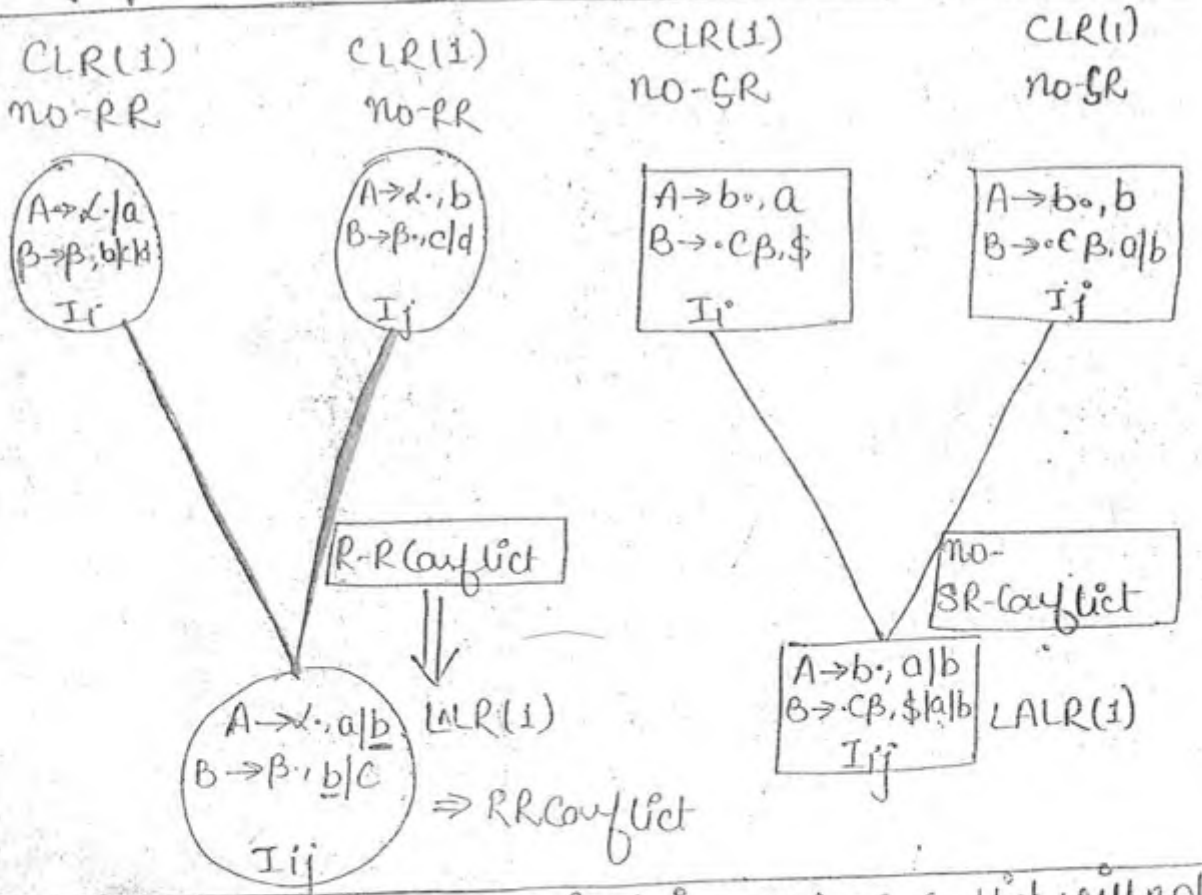
c) $n_1 = n_2 = n_3$

d) $n_1 \geq n_3 \geq n_2$

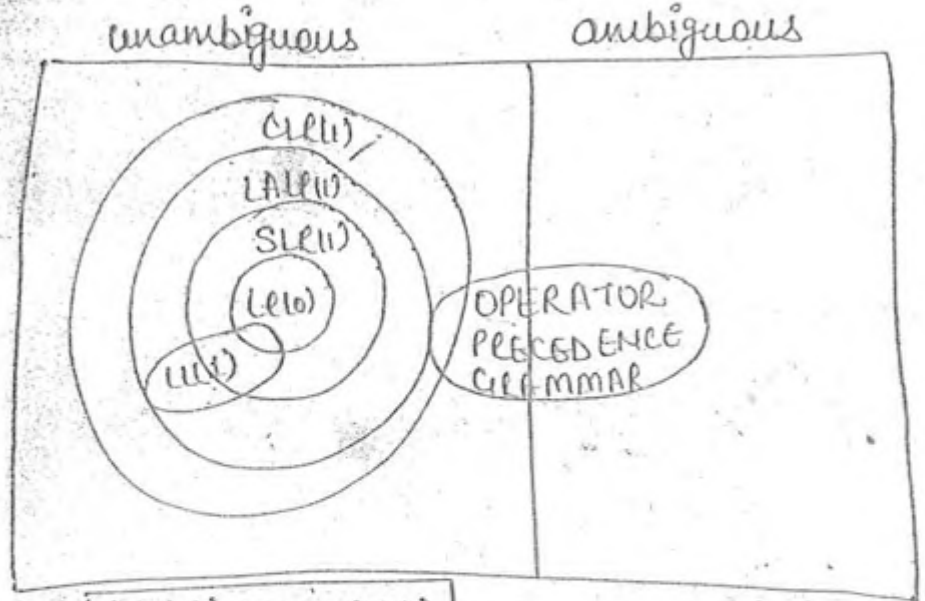
Soln $S \to (S) \mid a$



**Note 1:- Even though there are no R-R conflict in CLR(1), still RR conflict may present in LALR(1).



**Note 2:- If there are no SR-conflicts in CLR(1), SR-conflict will not present in LALR(1) also.



$LL(1) \subseteq LALR(1)$

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

$LL(k) \subseteq LR(k)$

$S \to a/ab$: it is LALR(1) not LL(1)

Note-1

① Bottom up parsers are more complex to design as compared to T.D.P

② Bottom up parser is accepting more no of grammars comparing with the top down parser.

③ Size of Bottom Up Parser table = 2 * top down Parser table size