→ Bottom up parse can be constructed for both Ambigous & Unambigous grammar
↳ Ambigous ⇒ Operator Precedence Parser
↳ Unambigous ⇒ LR(K) parsers

Date _____ Day _____

Error Recovery in Predictive Parsing

→ An error is detected during the predective parsing when the terminal on top of the stack doesnot match the next input symbol

→ BUP is faster than Top-down parser

▶ BOTTOM UP PARSER (Shift Reduce Parser)

- LR Parsers
  - ↳ Right most derivation in reverse order
  - → Input will be scanned from Left to right

→ A Production $A \rightarrow \cdot XYZ$ is called LR(0) item because it contains a dot.

↳ The position of dot determines the no. of symbols read.

↳ $A \rightarrow (X) \cdot YZ$ ← 1 symbol read
$A \rightarrow (XY) \cdot Z$ ← 2 symbol read
$A \rightarrow (XYZ) \cdot$ → whole string read

These all are LR(0) items

↳ This is called the final item / completed item since all symbols are read

→ Closure (I)

Closure ($S \rightarrow \cdot AA$)

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

iska closure ↳ $S \rightarrow \cdot AA$ (1st repeat the production)

↳ $A \rightarrow \cdot aA$ ← dot ke baad non terminal hai to uska tamaam production with dot likh denge.

$A \rightarrow \cdot b$

iska bhi closure dekhenge → phir iska closure nikalenge

→ dot ke baad terminal hai to closure nhi

→ Substring of the i/p string that matches with RHS of any production, is called as Handle. A → αA → handle

→ The process of finding the handle & replacing that handle by it's LHS variable is called handle pruning

Date Day

→ ① Add LR(0) item I to closure (I)

② If $A \to \alpha \cdot B\beta$ is an LR(0) item I & $B \to Y$ is in Grammar "G", then

Add $B \to \cdot Y$ to closure (I)

③ Repeat above two step for every newly added LR(0) item.

→ Goto (I, X)

Goto ($S \to \cdot AA$, A)

G BS Dot ko shift krdeng X. jaani 'A' k aage

$S \to A \cdot A$

now take closure

$A \to \cdot aA$ → closure of this

$A \to \cdot b$ → then closure of this

Goto ($S \to A \cdot A$, A)

G $S \to AA \cdot$ → iska closure nhi hosakta to yehi ans hoga iska

• LR(0) Parsing Table

① Find augmented Grammar

② I_0 = closure (Augmented LR(0) item)

③ Apply closure & Goto function and find all collection of LR(0) item using finite automata (DFA)

④ Reduce DFA = LR(0) passing table.

All the the states of LR(0) items are collectively known as canonical LR(0) collection

LR (0) Parser :

$S \rightarrow AA$

$A \rightarrow aA \mid b$

① $S \rightarrow AA$
② $A \rightarrow aA$
③ $A \rightarrow b$

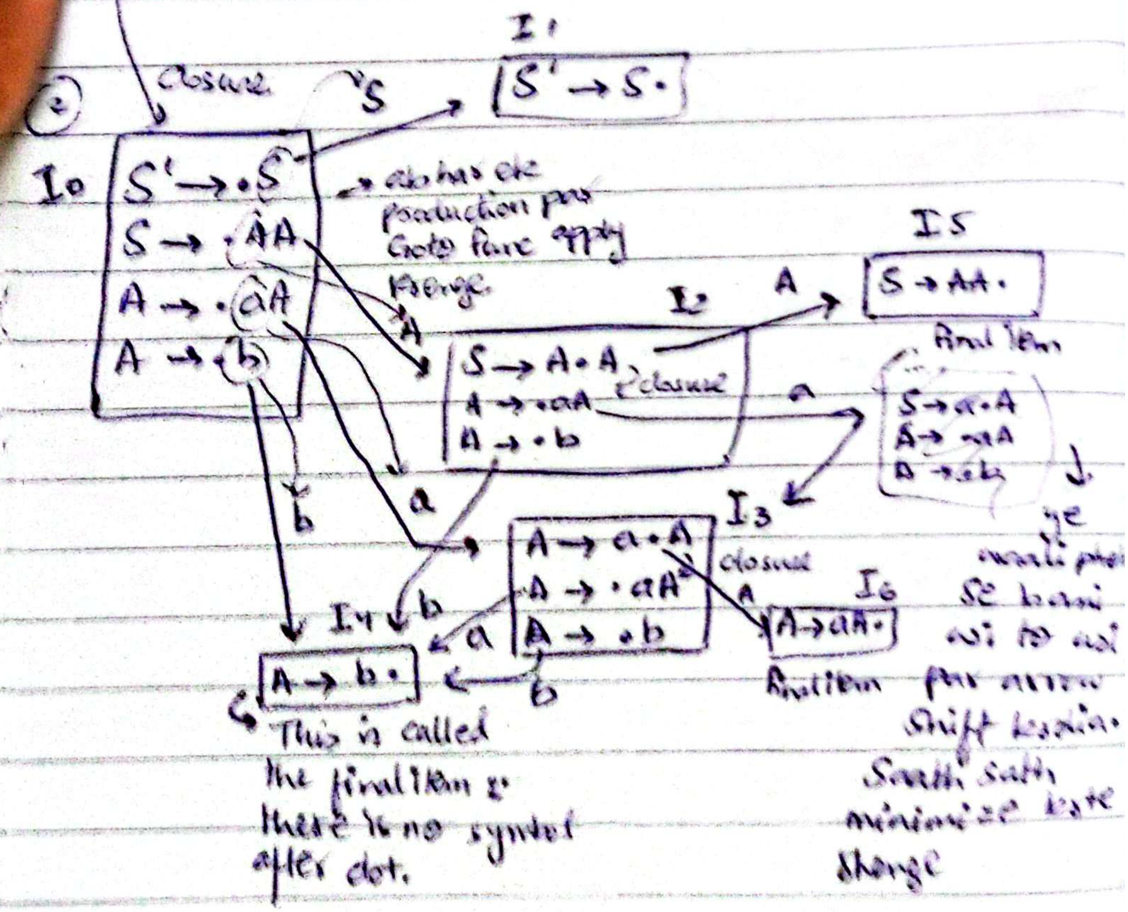
① Augmented Grammar

$S' \rightarrow S$ → Introduce a new variable such that Start symbol is on right hand side

$S \rightarrow AA$

$A \rightarrow aA \mid b$

② Closure



I₁ mein Start Symbol ke baad dot hai to is wajase accept likha.

Shift 3

terminal par agr transition hoti hai to Shift + state no.

Date _____ Day _____

	Action			Goto	
	a	b	$	S	A
0	S_3	S_4		1	2
1	~~S_3~~	~~S_4~~	Accept		5
2	S_3	S_4			5
3	S_3	S_4			6
4	γ_3	γ_3	γ_3		
5	γ_1	γ_1	γ_1		
6	γ_2	γ_2	γ_2		

total no. of States

State 4 is a final item. final item ka production # dekhenge which is (3) so we right Reduce 3 (R_3) in all action symbols

→ If any cell has multiple entry than we cannot make LR Passing table.

→ Input String ⇒ a a b b $

0	a

↳ 0th state par "a" ki value dekhenge which is S_3, so we write 3

a a b b

0	a	3	a

↳ 3rd state par "a" kiwal i.e S_3

0	a	3	a	3

Page No.

Date _____ Day _____

aabb$
↑
↳ 3rd par b ki val i.e $4

| 0 | a | 3 | a | 3 | b | 4 |

↳ ~~3rd par b ki value~~

A
↑
a a b b $

a a b b $
↑
4th k b par δ3, 5th production b

| 0 | a | 3 | a | 3 | b | 4 | A |

A → b
length of right side
(X) by 2 = 2
so pop 2 element

5th state par A = 6

| 0 | a | 3 | a | 3 | A | 6 |

A
A
A
a a b b b

a a b b
↑
top of stack & current symbol

6 b → δ2 (A → aA)
2 × 2 = 4
pop 4

| 0 | a | 3 | a | 3 | A | 6 | A | 6 |

3rd par A = 6

A
A
A
a a b b $

a a b b
↑
6th par b = R2 (A → aA)
2×2 = 4

| 0 | a | 3 | A | 6 | A | 2 |

0th of A = 2

2nd of b = $4

(pointer of input string is only incremented on shift)

| 0 | A | 2 | b | 4 |

a a b b $
↑

Date Day

4th of $ = R_3 $(A \to b)$

1 x 2 = 2 pop

| 0 | A | 2 | b | Y | A | 5 |

2nd of A = 5

Now 5th of $ = γ_1 $(S \to AA)$

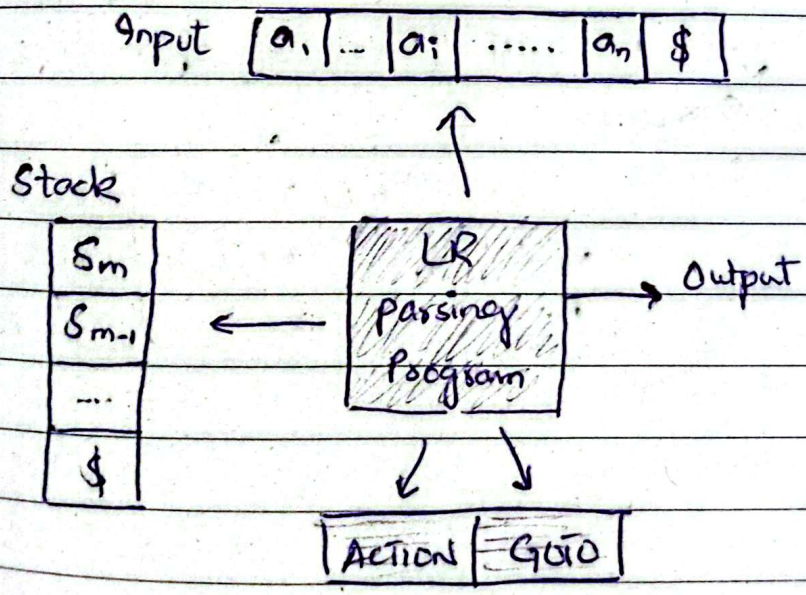
2 x 2 = 4 pop

| 0 | A | 2 | A | $ | $ | 1 |

0th of S = 1 $

1th of $ = ACCEPT ✓

→ Top of Stack hamesha state no. hoga.



▲ Model of an LR Parser

Date _____ Day _____

② $S \to (L) \mid a$

$L \to L,S \mid S$

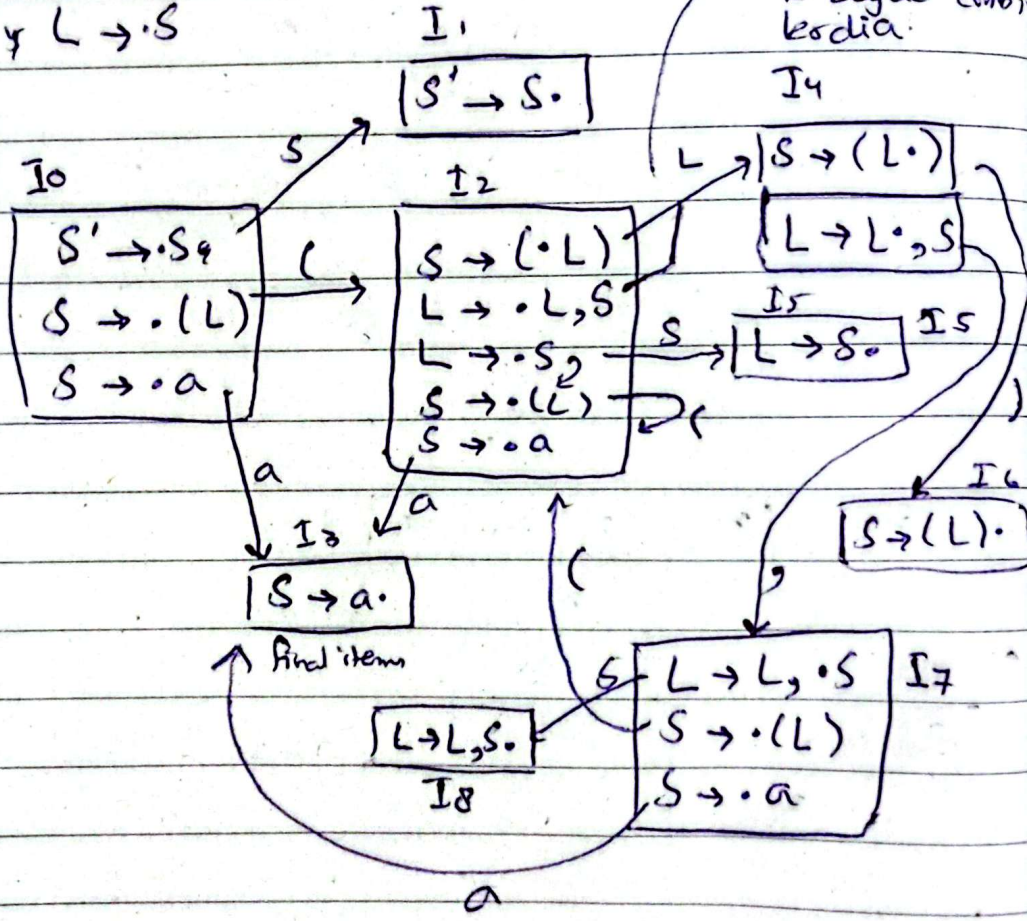
Augmented Grammar

① $S' \to .S$

2 $S \to .(L) \mid a$

3 $L \to .L,S$

4 $L \to .S$



Date _____ Day _____

	ACTION					GOTO	
	(	)	a	,	$	S	L
0	S_2		S_3			1	
1					ACCEPT		
2	S_2		S_3			5	4
3	γ_2	γ_2	γ_2	γ_2	γ_2		
4		S_6		S_7			
5	γ_4	γ_4	γ_4	γ_4	γ_4		
6	γ_1	γ_1	γ_1	γ_1	γ_1		
7	S_2		S_3			8	
8	γ_3	γ_3	γ_3	γ_3	γ_3		

▲ LR(0) Passing Table

String → (a,a,a)$

Stack	Input	Action
$0	(a,a,a)$	Shift ((S_2)
$0(2	a,a,a)$	Shift a (S_3)
$0(2a3	,a,a)$	Reduce ⇒ S→a (Pop 2)
$0(2S5	,a,a)$	Reduce → L→S (Pop 2)
$0(2L4	,a,a)$	Shift , (S_7)
$0(2L4,7	a,a)$	Shift a (S_3)
$0(2L4,7a3	,a)$	Reduce S→a (Pop 2)
$0(2L4,7S8	,a)$	Reduce L→L,S
$0(2L4	,a)$	Shift , (S_7)

3×2=6

Day

Date

$0(2L4,7	a) $	Shift a (s_3)
$0(2L4,7a3	) $	Reduce S → a
$0(2L4,7S8	) $	Reduce L → L,S
$0(2L5	) $	Reduce L → S
$0(2L4	) $	Shift) (s_6)
$0(2L4)6	$	Reduce S → (L)
$0S1	$	ACCEPT

- Conflicts in Shift Reduce Parsing

→ Every shift reduce parser for a grammar can reach a configuration in which the parser, knowing the entire stack and also next input symbol cannot decide

↳ Wether to shift or reduce (a shift/reduce conflict)

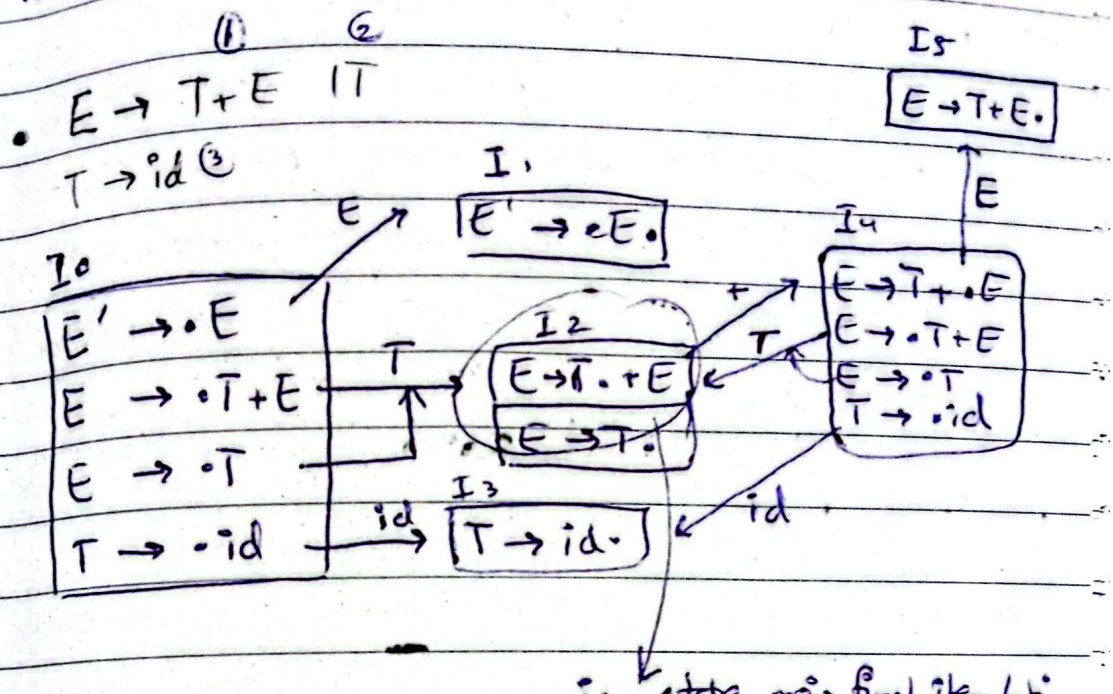
↳ which of several reductions to make (a reduce / reduce conflict)

→ To avoid conflicts do right most derivation for the given input string, as per grammar

→ Lekn LR(1) Parser mein iski need nhi because hum states maintain krrhe hote hai.

Date _____ Day _____

→ Agr kisi state mein S-R ya R-R conflict aajye to we inadequate State bolte hai.



is state mein final item bhi hai which is used in reducing and also there is shift So when we make table we will have conflict.

	ACTION			GOTO	
	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		

→ Coz S-R conflict Therefore this ~~cannot~~ Grammar is not LR(1)

Date _____ Day _____

▷ SLR(1) Passer :

→ LR(1) or SLR(1) ki working bilkul same hai
→ Bs ~~hum~~ reduction (δ_n) hum har jaga nhi likhenge, bike phele hum δ_n production ka LHS ka follow lenge or uske follow main jo terminals aayenge sirf whi reduction likhenge.

	ACTION			GOTO	
	id	+	$	E	T
0	S_3			1	2
1			ACCEPT		
2			δ_2		
3		δ_3	δ_3		
4	S_3			5	2
5			δ_1		

$E \to T$
Follow(E) = $

$E \to T+E$
Follow(E) = $

$T \to id$
Follow(T) = +, $

Date _____ Day _____

→ Next all steps are same.

Q: Check wether the grammar is SLR(1)

① $S \xrightarrow{1} A\,a\,A\,b \mid \xrightarrow{2} B\,b\,B\,a$ Follow(S) = $

$A \xrightarrow{3} \in$ Follow(A) = a, b

$B \xrightarrow{4} \in$ Follow(B) = a, b

I0

$S' \to .S$

$S \to .AaAb$

$S \to .BbBa$

$A \to .$

$B \to .$

→ Two final items means two reductions

✗ LR(0)

	a	b	$
0	γ_3/γ_4	γ_3/γ_4	γ_3/γ_4

✓ SLR(1)

	a	b	$
0	γ_3/γ_4	γ_3/γ_4	

② $S \to Aa \mid bAc \mid dc \mid bda$

$A \to d$

I0

$S' \to .S$

$S \to .Aa$

$S \to .bAc$

$S \to .dc$

$S \to .bda$

$A \to .d$

I1: $S' \to S.$

A → I2: $S \to A.a$

b → I3: $S \to b.Ac$, $A \to .d$, $S \to b.da$

d → I4: $S \to d.c$

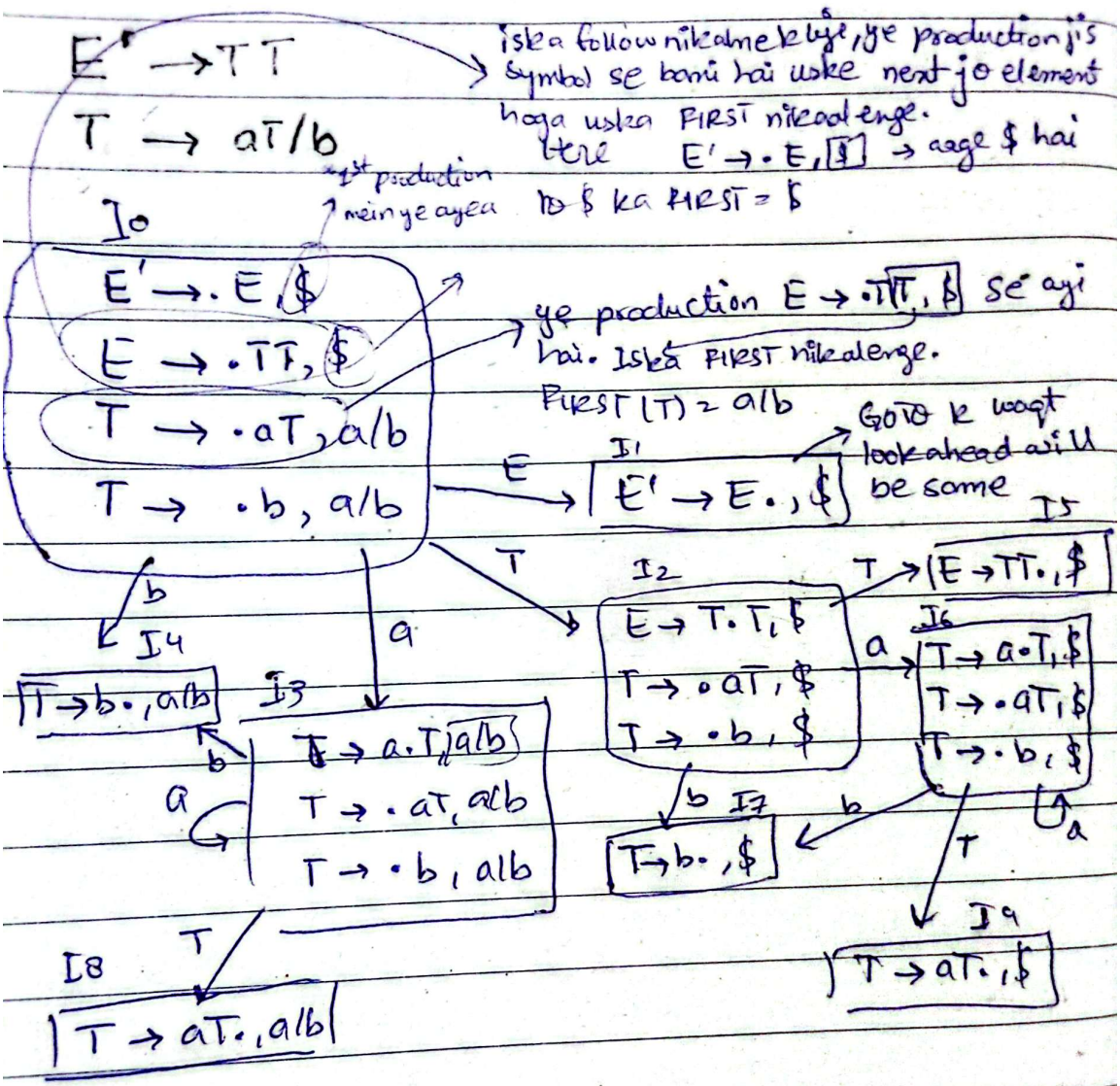
d → I5: $A \to d.$

Date _____ Day _____

▷ CLR(1) PARSER :

↳ Uses LR(1) items.

↳ LR(1) items = LR(0) items + look ahead symbols

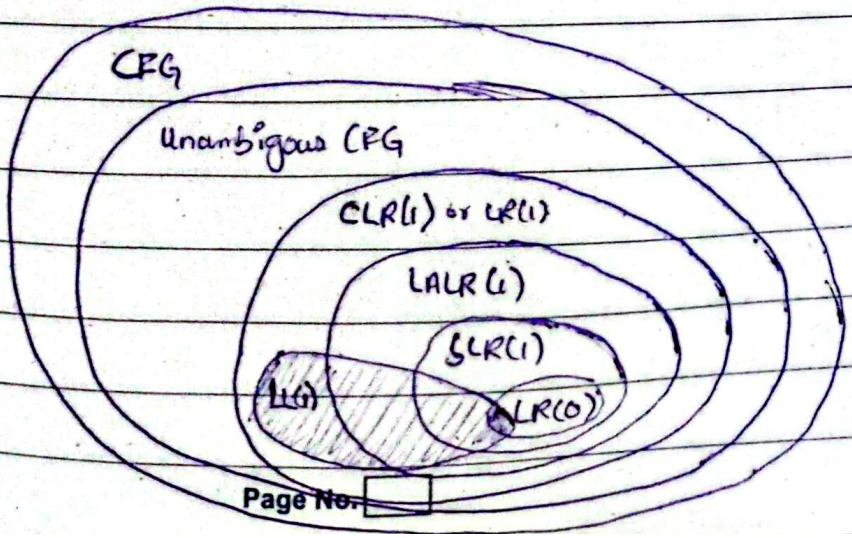


Date _____ Day _____

	ACTION			GOTO	
	a	b	$	E	T
0	S_3	S_4		1	2
1			ACCEPT		
2	S_6	S_7			5
3	S_3	S_4			8
4	γ_3	γ_3			
5			γ_1		
6	S_6	S_7			9
7			γ_3		
8	γ_2	γ_2			
9			γ_2		

→ 4 is the reduced state (final state) jismein hum bs look-ahead symbols par hi entry krenge.

- ~~Conflicts of~~ can RR & SR conflicts can also occur in this Parser.
- This is the most powerful parser



Page No.

Date _____ Day _____

LALR(1) Parser

→ CLR(1) was too much costly, because hum sirf ek lookahead symbol change hone ki wajah se alag alag state banate the, jiski wajah se no. of states increased too much.

→ LALR(1) main hum "states having same rules but diff lookahead symbols" ko merge krne ki koshish krte hai.

→ CLR(1) ki canonical states nikaalne k baad hum bs check krenge k konsi states k rules same hai (dots should also match) or unko note krlenge.

→ Phir jo table banega usko merge krdenge.

→ From Prev examples:

→ table mein we cut the I_6 row and uski entries I_3 mein krdenge. New state will become I_{36}

$I_3 - I_6$] Similar
$I_4 - I_7$] States
$I_8 - I_9$

↳ Agar merge krne mein conflict aaye so it cannot be parsed then LALR(1)

Date _____ Day _____

	ACTION			GOTO	
	a	b	$	E	T
0	s_3	s_4		1	2
1			ACCEPT		
2	s_6	s_7			5
3(6)	s_3	s_4		9	8
4(7)	r_3	r_3	r_3		
5			r_1		
6	s_6	s_7		9	
7			r_3		
8(9)	r_2	r_2	r_2		
9			r_2		

→ Agr kisi ~~grammar~~ CLR(1) mein merge krne ki capacity na ho (~~like~~ similar states na ho), it means it is already optimize, therefore it will also be considered as LALR(1)

→ Agr kisi CLR(1) mein merge krne ki capacity ho, but merge krne se RR conflict aaye to it will not be classified as LALR(1)

→ Agr kis CLR(1) mein SR conflict nhi hai to LALR(1) mein bhi nhi hoga.

→ But agr kisi CLR(1) mein RR conflict nhi, to LALR(1) mein bhi RR conflict Page No. ☐ na ho esa zaroori nhi.

Date _____ Day _____

▷ OPERATOR PRECENDENCE PARSER

→ Can be constructed for both ambigous & unambigous grammar.

→ In general, operator precedence gramar has less complexity

→ Every CFG is not operator precedence grammar

→ Generally used for languages which one useful in scientific calculation applications.

→ It has the following 2 properties

(1) Doesnot contain "ϵ" production

(2) No adjacent (consecutive) non-terminals on RHS of a production

$S \to SA$	$S \to Ab$	$S \to Ab$
$A \to ABd \mid ab$ ✗	$A \to B \mid \epsilon$ ✗	$A \to AbB$ ✓

→ Converting a Grammar to OPG:

$S \to SAS \mid a$

$A \to bSb \mid b$

Substitute A

$S \to SbSbS \mid aSbS \mid a$ ✓

$A \to bSb \mid b$

Date Day

$P \to SR.|\epsilon$
$R \to bSR \mid bS$
$S \to WbS \mid W$
$W \to L*W \mid L$
$L \to id$

→

$P \to SbSR \mid SbS \mid S$
$R \to bSR \mid bS$ (original grammar)
$\because P \to SR$
$P \to SbP \mid SbS \mid S$
$R \to bP \mid bS$
$S \to WbS \mid W$
$W \to$ — —

▷ Parsing a String

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

Symbol will always have greater precedence

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

* ki ← * se associativity dekhenge is waja se -> ye lagaya or + par < lagaya.

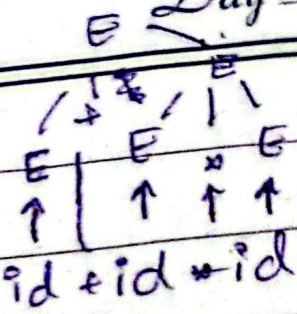
→ We will mantain a stack & push $ initially than ~~start~~ start traversing.

↳ If the precedence of current symbol on i/p string is greater than top of stack element we push in stack & increment i/p pointer.

↳ If the precedence of current symbol on i/p string is less than the top of stack element then we pop and reduce that symbol. Pointer remains in its place Then again compare with top of the stack.

Date _____ Day _____

String: id + id * id $



$	id	+	id	*	id
	1		5		2		4		3	