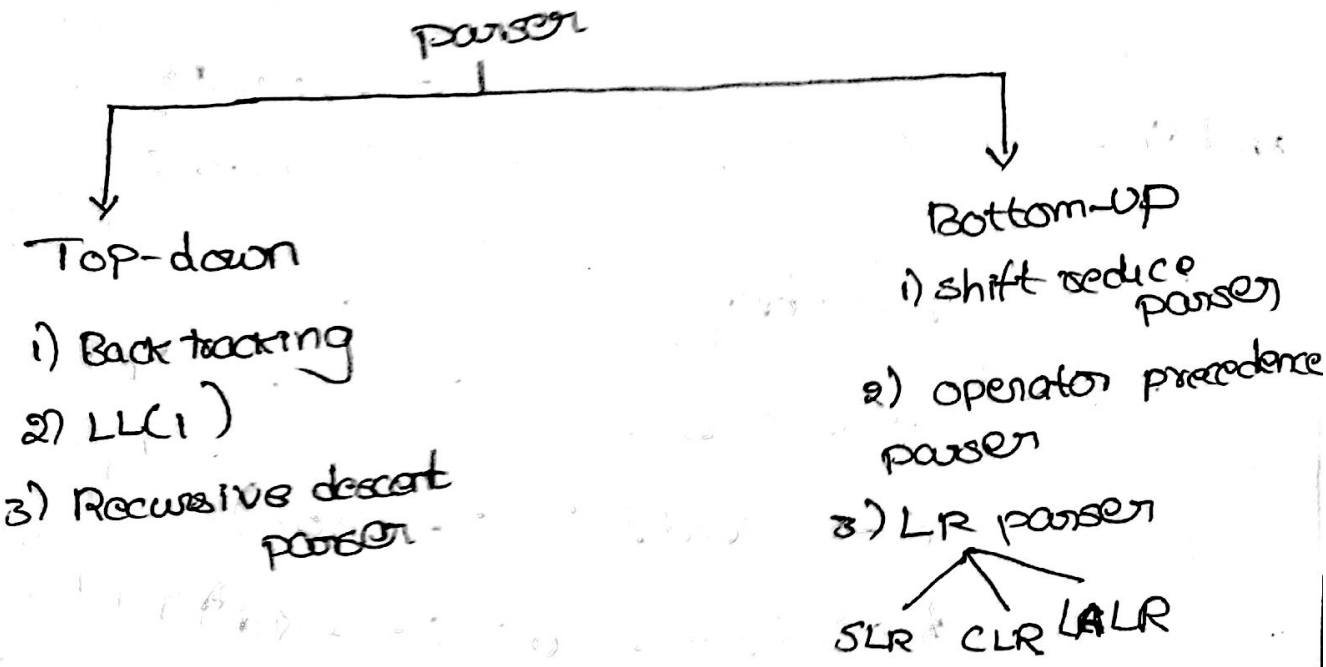
UNIT-3

Bottom UP parser



* Bottom-up parsing

> Bottom-up parsers are those which starts from the input string i.e from the leaves in the parse tree and move towards the root i.e trys to get back the start symbol of the grammer.

> If the start symbol of the grammer is can be obtained from the input string then the string is said to be accepted by the language.

Bottom-up parsers tries to find right most derivation in reverse from the

given input string.

Eg: Consider the grammer $S \rightarrow TL;$

$T \rightarrow int \mid float$

$L \rightarrow L, id \mid id$

and the input string is float id, id;

TOP-down , LMD

$S \rightarrow TL;$

$S \rightarrow float\ L;$ ($T \rightarrow float$)

$S \rightarrow float\ L, id;$ ($L \rightarrow L, id$)

$S \rightarrow float\ id, id;$ ($L \rightarrow id$)

Bottom-up (RMD reverse)

float id, id;

T id, id;

T L, id;

T L;

S

RMD

$S \rightarrow TL;$

$S \rightarrow TL, id;$

$S \rightarrow T\ id, id$

$S \rightarrow float\ id, id;$

* Handle

Handle is a sub string of a string that match right side of a production which when reduced to a non-terminal on the left side of the production represents one step of the reverse right most derivation (or)

Handle is a string of substring that matches the right of the production and we can reduce such string by a non-terminal on left handside production.

> If there is a production $A \to B$, then B is to be handled, since it can be reduced to 'A' in the string α, β, w as '$\alpha B w$'
Reducing B to A in $\alpha B w$ is said to be pruning the handle.

Consider the gramer

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

and derive the string id + id * id

$E \to E*E$

$E \to E*id$ $(E \to id)$

$E \to E+E*id$ $(E \to E+E)$

$E \rightarrow E + id * id \quad (E \rightarrow id)$

$E \rightarrow id + id * id \quad (E \rightarrow id)$

* Shift reduce Parser

→ shift reduce parser is a bottom-up parser which tries to construct a parse tree for an input string beginning at the leaves & moves towards the root

→ The process is to reduce the string 'w' to the start symbol of the grammer

→ At each step of reduction, a particular substring matching the right side of the production is replaced by the symbol on the left of the production.

→ The reduction made by the shift reduce parser are shift → in a shift action the next input symbol is shifted on to the top of the stack.

Reduce → In a reduce action, the handle on the top of the stack will be reduced by the non-terminal which is towards the left side of the production

accept → In an accept action the parser announces successful completion of parsing

Error → In an error action, the parser discovers that a syntax error has occured and calls an error recovery procedure.

* Stack shift implementation of shift reduce parsing.

→ The data structure is used to implement the stack which can hold gramer symbols and an input buffer to hold the string to be parsed

→ The dollar symbol is used to mark the bottom of the stack & also the right end of the input.

→ Initially the stack is empty and the string 'w' is on the input

Stack	Input
$	w$

* Conflicts during shift reduce parsing

→ (1) Shift / Reduce Conflict : The parser even after knowing the entire stack contents & the next input symbols, cannot decide wheather to shift or to reduce this is called shift reduce conflict.

→ (2) Reduce / Reduce Conflict : The parser even after knowing the entire stack contents & the next input symbols, cannot decide which production to reduce This is called reduce / reduce conflict.

Ex

1) Consider the grammer

$E \to E - E / E * E / id$ perform shift reduce parsing, after input string

id - id * id

id has highest priority
$ " least "

Input - highest - shift
stack - highest - reduce

id - id * id $
E - E * E $
E - E
E

Stack	Input	Action
$	id - id * id $	Shift id
$ id	- id * id $	Reduce E→id
$ E	- id * id $	shift -
$ E -	id * id $	shift id
$ E - id	* id $	Reduce E→id
$ E - E	* id $	shift *
$ E - E *	id $	shift id
$ E - E * id	$	reduce E→id
$ E - E * E	$	Reduce E→E*E
$ E - E	$	Reduce E→E-E
$ E	$	Accept

2) Consider the following grammer

S → TL;
T → int | float
L → L, id | id

parse the input string for int id, id; using shift reduce parser

int id, id; $

T id, id : $

T L, id ; $

(ii) (on

Stack	Input	Action
$	int id, id ; $	shift int
$ int	id, id; $	Reduce T->int
$T	id, id; $	shift id
$ T id	, id ; $	Reduce L->id
$ TL	, id ; $	shift ,
$ TL,	id ; $	shift id
$ TL, id	; $	shift id
$ TL	; $	Shift
$ TL ;	$	Reduce S->TC;
$ S	$	Accept

* LR Parser

→ LR parser uses bottom up parsing which can be used to pass the large class of context free grammer. This method is also called as LR(K) parsing.

→ 'L' stands for Left to Right scanning
'R' stands for Right most derivation in reverse
'K' is number of input symbols.

* Model

* Model of LR Parser :-

→ It consists of input buffer for storing the input string, stack for storing the grammar symbols, output & a parsing table consists of two parts namely 'action', 'goto'.

→ There is one parsing program which is actually a driving program and reads the input symbol at a time from the input buffer.

→ The driving program works as follows

1) It initialize the stack with start symb

and invokes lexical analyzer to get next token

2) which determines 's_j' the state currently on the top of the stack & a_i the current input symbol

3) Then it consult the parsing table for action $[s_j, a_i]$ which can have one of the four values.

i) s_j means shift state j

ii) r_j means reduce by rule j

iii) accept means successful parsing is done

iv) Error indicates syntatical error

* Types of LR parser

SLR (Simple) LR(0)

CLR (canonical) LR(1)

LALR (look-a-head) LR(K)

→ The overall structure of all these LR parsers is same all are table driven parsers.

→ Construct the SLR parsing table

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

Now, Augmented grammer is

$E' \rightarrow E$

canonical set of items

I_0 - $E' \rightarrow \cdot E$

$E \rightarrow \cdot E+T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T*F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot id$

I_1 goto(I_0, E)

$E' \rightarrow E \cdot$

$E \rightarrow E \cdot +T$

I_2 goto(I_0, T)

$E \rightarrow T \cdot$

$T \rightarrow T \cdot *F$

I_3 goto(I_0, F)

$T \rightarrow F \cdot$

(I4) goto(I0, ()

F → (·E)

E → ·E+T

E → ·T

T → ·T*F

T → ·F

F → ·(E)

F → ·id

(I5) goto(I0, id)

F → id·

(I6) goto(I1, +)

E → E+·T

T → ·T*F

T → ·F

F → ·(E)

F → ·id

(I7) goto(I2, *)

T → T*·F

F → ·(E)

F → ·id

(I8) goto(I4, E)

F → (E·)

$E \rightarrow E \cdot + T$

(I2) goto(I1, T)

$E \rightarrow T \cdot$

$T \rightarrow T \cdot * F$

(I3) goto(I1, F)

$T \rightarrow F \cdot$

(I4) goto(I1, ()

$F \rightarrow (\cdot E)$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot id$

(I5) goto(I6, id)

$F \rightarrow id \cdot$

(I6) goto(I6, T)

$E \rightarrow E + T \cdot$

$T \rightarrow T \cdot * F$

(I7) goto(I6, F)

$T \rightarrow F \cdot$

(I4) goto(I0, ( )

F → (· E)

E → · E + T

E → · T

T → · T * F

T → · F

F → · (E)

F → · id

(I5) goto(I0, id)

F → id ·

(I10) goto(I7, F)

T → T * F ·

(I4) goto(I7, ( )

F → (· E)

E → · E + T

E → · T

T → · T * F

T → · F

F → · (E)

F → · id

(I5) goto(I7, id)

F → id .

(I11) goto(I8,))

F → (E) .

(I6) goto(I8, +)

E → E+ . T

T → . T * F

T → . F

F → . (E)

F → . id

(I7) goto(I9, *)

T → T * . F

F → . (E)

F → . id .

E' → E . → I1

γ1 E → E+T → I9

γ2 E → T = I2

γ3 T → T * F = I10

γ4 T → F = I3

γ5 F → (E) = I11

γ6 F → id = I5

To perform reduce operation

FIRST(E) = { (, id }

FIRST(T) = { (, id }

FIRST(F) = { (, id }

FOLLOW(E') = { $ }

$F \rightarrow (E)$

$A \rightarrow \alpha B \beta$

FIRST( ) = FOLLOW(E)

= {) }

$E \rightarrow E + T$

$\alpha \; B \; \beta$

FIRST(+T) = FOLLOW(E)

= { + }

$E' \rightarrow E$

FOLLOW(E') = FOLLOW(E)

= { $ }

FOLLOW(E) = {), +, $ }

now

$E \rightarrow E + T$

$\alpha \; B$

FOLLOW(E) = FOLLOW(T)

= {), +, $ }

$E \rightarrow \in T$

$A \rightarrow \alpha B$

FOLLOW(T) = FOLLOW(E)

= {), +, $ }

$T \rightarrow T * F$

FIRST FOLLOW(*F) = Follo(T)

= { * }

FOLLOW(T) = {), +, * $ }

T → T * F

FOLLOW(T) = FOLLOW(F)
= { +,), * $ }.

T → * F

FOLLOW(T) = FOLLOW(F)
=), + * $ }

FOLLOW(F) = {), + * $ }

state	Action						GotO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				Accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

2) $S \rightarrow SS$

$S \rightarrow a$

$S \rightarrow \epsilon$

now, Augmented grammer

$S' \rightarrow S$

Canonical set of items

I_0 $S' \rightarrow \cdot S$

$S \rightarrow \cdot SS$

$S \rightarrow \cdot a$

$S \rightarrow \cdot$

I_1 goto(I_0, S)

$S' \rightarrow S \cdot$

$S \rightarrow S \cdot S$

$S \rightarrow \cdot a$

$S \rightarrow \cdot$

$S \rightarrow \cdot S S$

I_2 goto(I_0, a)

$S \rightarrow a \cdot$

I_3 goto(I_1, S)

$S \rightarrow SS \cdot$

$S \rightarrow S \cdot S$

$S \rightarrow \cdot a$

$S \rightarrow \cdot$

(I2) goto(I_1, a)

$S \rightarrow a \cdot$

(I_3) goto(I_3, S)

$S \rightarrow SS \cdot$

$S \rightarrow S \cdot S$

$S \rightarrow \cdot SS$

$S \rightarrow \cdot a$

$S' \rightarrow S.$ (I_1)

r_1 $S \rightarrow SS$ (I_3)

r_2 $S \rightarrow a$ (I_2)

~~r_0 $S \rightarrow \epsilon$ (I_0)~~

(I_2) goto(I_3, a)

$S \rightarrow a \cdot$

To perform reduce operation

FIRST(S) = { a, ∈ }

FOLLOW(S') = {$}

$S' \rightarrow S$

$A \rightarrow \alpha B$

FOLLOW(S') = FOLLOW(S)

= { a, $ }

$S \rightarrow \epsilon SS$

$A \rightarrow \alpha B\beta$

FIRST(S) = FOLLOW(B)

= { a, ∈ }

FOLLOW(S) = FOLLOW(S)

= { a }

state	Action		goto
	c	$	S
0	S2, S3	r3	1
1	S2	Accept	3
2	r2		
3	S2, r1		3

3) $S \to cA \mid ccB$

$A \to cA \mid a$

$B \to ccB \mid b$

now Augmented grammer

$S' \to S$ $S' \to S$

Canonical set of items

I_0 $S' \to \cdot S = I_{01}$

r_1 $S \to \cdot cA = I_{10}$

r_2 $S \to \cdot ccB = I_{11}$

r_3 $A \to \cdot cA = I_5$

r_4 $A \to \cdot a = I_3$

r_5 $B \to \cdot ccB = I_7$

r_6 $B \to \cdot b = I_4$

(I1) goto(I0, S)

$S' \rightarrow S.$ — C

(I2) goto(I0, c)

$S \rightarrow c.A$

$S \rightarrow c.cB$

$A \rightarrow .c.A$

$B \rightarrow c.cB$

$A \rightarrow .cA$

$A \Rightarrow .a$

(I3) goto(I0, a)

$A \rightarrow a.$ — C

(I4) goto(I0, b)

$B \rightarrow b.$

(I5) goto(I2, A)

$S \rightarrow cA.$

$B \rightarrow cA.$

(I6) goto(I2, c)

$S \rightarrow cc.B$

$B \rightarrow cc.B$

$A \rightarrow c.A$

$B \to \cdot ccB$

$A \to \cdot cA$

$A \to \cdot a$

$B \to \cdot b$

I_3 goto(I_2, a)

$A \to a \cdot$

I_4 goto(I_6, B)

$S \to CCB \cdot$

$B \to ccB \cdot$

I_8 goto(I_6, A)

$A \to CA \cdot$

I_9 goto(I_6, C)

$B \to c \cdot cB$

$A \to c \cdot A$

$A \to \cdot cA$

$A \to \cdot a$

I_4 goto(I_6, b) $B \to b \cdot$

I_5 goto(I_6, a)

$A \to a \cdot$

I_{10} goto(I_9, C)

$B \to cc \cdot B$

$A \to cA \cdot$

$A \to c \cdot A$

$A \to \cdot cA$

$A \to \cdot a$

$B \to \cdot ccB$ $B \to \cdot b$

I_3 goto(I_9, a)

$A \to a \cdot$

I_8 goto(I_9, A) $A \to cA \cdot$

I_{11} goto(I_{10}, B)

$B \to ccB \cdot$

I_8 goto(I_{10}, A)

$A \to cA \cdot$

I_9 goto(I_{10}, C)

$B \to c \cdot cB$

$A \to c \cdot A$

$A \to \cdot cA$

$A \to \cdot a$

(I3) goto ($I_{q_1}^{10}$ a)

$A \rightarrow a \cdot$

(I4) goto ($I_{q_{10}}$, b)

$B \rightarrow b \cdot$

To perform reduce operations

FIRST(S) = { c }

FIRST(A) = { c, a }

FIRST (B) = { c, b }

FOLLOW(S') = $

S:

$S' \rightarrow \epsilon S$
αB

FOLLOW(S') = FOLLOW(S)

= { $ }

$S \rightarrow c A$
$A \rightarrow \alpha B$

FOLLOW(B) = FOLLOW(A)

= { $ }

$A \rightarrow cA$

$A \rightarrow \alpha B$

FOLLOW(A) = FOLLOW(A)

= { $ }

$S \rightarrow \underset{\alpha}{cc} \underset{B}{B} \in$

~~FOLLOW~~(∈) = F

FIRST(∈) = FOLLOW(B)

{ ∈ }

$B \rightarrow \underset{\alpha}{cc} \underset{B}{B}$

FOLLOW(S) = FOLLOW(B)

{ $ }

$B \rightarrow \underset{\alpha}{cc} \underset{B}{B}$

FOLLOW(B) = FOW(B)

= { $ }

state	Action				goto		
	a	b	c	$	S	A	B
0	s_3	s_4	s_2		1		
1				Accept			
2	s_3		s_6			5	
3							
4							
5				r_1/r_3			
6	s_3	s_4	s_9			8	7
7				r_3/r_5			
8							
9	s_3		s_{10}			8	
10	s_3	s_4	s_9			8	11
11				r_5			

p) $S \to L = R / R$

$L \to *R / id$

$R \to L$

Augmented grammer

$S' \to S$

canonical form

I_0 $S' \to \cdot S$

$S \to \cdot L = R$

$S \to \cdot R$

$L \to \cdot *R$

$L \to \cdot id$

$R \to \cdot L$

I_1 goto(I_0, S)

$S' \to S \cdot$

I_2 goto(I_0, L)

$S \to L \cdot = R$

$R \to L \cdot$ — C

~~$L \to \cdot *R$~~

~~$L \to \cdot id$~~

I_3 goto(I_0, R)

$S \to R \cdot$ — C

I_4 goto$(I_0, *)$

$L \to * \cdot R$

$R \to \cdot L$

$L \to \cdot *R$

$L \to \cdot id$

I_5 goto(I_0, id)

$L \to id \cdot$

I_6 goto$(I_2, =)$

$S \to L = \cdot R$

$R \to \cdot L$

$L \to \cdot *R$

$L \to \cdot id$

(I7) goto(Iu, R)

L → *R •

(I8) goto(Iu, L)

R → L •

(Iu) goto(Iu, *)

L → * • R

R → • L

L → • *R

L → • id

(I5) goto(Iu, id)

L → id • — C

(I6) goto(I6, R)

S → L = R • — C

(I8) goto(I6, L)

R → L • — C

(I4) goto(I6, *)

L → * • R

R → • L

L → id

L → • *R

(I5) goto(I6, id)

L → id •

To perform reduce operation

FIRST(S) = { *, id }

FIRST(L) = { *, id }

FIRST(R) = { *, id }

S' → S

FOLLOW(S') = { $ }

S' → ES

A → αB

FOLLOW(S') = FOLLOW(S)

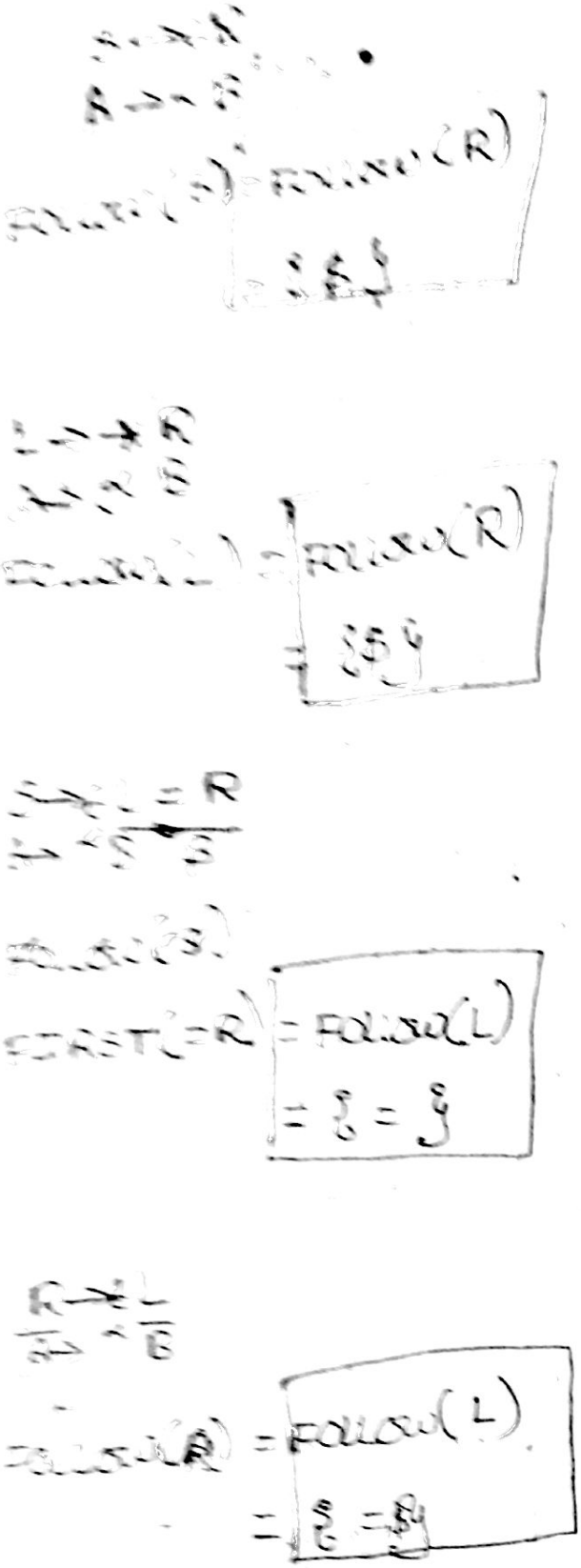
= { $ }

S → L = R

A → αβB

FOLLOW(S) = FOLLOW(R)

= { $ }



Scanned by CamScanner

State	Action				goto		
	=	*	id	$	S	L	R
0		S4	S5		1	2	3
1							
2	S6						
3							
4		S4	S5			8	7
5							
6		S4	S5			8	9
7							
8							
9							

1) Consider the grammer $S \to CC$

$C \to aC$

$C \to d$

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

$A \to \alpha \cdot B\beta, a$

$A = S', \quad \alpha = \epsilon, \quad B = S, \quad \beta = \epsilon, \quad a = \$$

$FIRST(\epsilon\$) = \{\$\}$

$FIRST(\epsilon\$) = \boxed{FOLLOW(S) = \{\$\}}$

$S \to \cdot CC, \$$

$A \to \alpha \cdot B\beta, a$

$FIRST(C\$) = FOLLOW(C)$

$= \{a, d\}$

$C \to \cdot aC, a/d$

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

$S \to \cdot CC, \$$

$A \to \alpha \cdot B\beta, a$

$FIRST(C\$) = \boxed{FOLLOW(C) = \{a, d\}}$

C → .aC , a/d

C → .d , a/d

I_0 :- S' → .S , $

S → .CC , $

C → .aC , a/d

C → .d , a/d

(I1) :- goto (I_0, S)

S' → S. , $

(I2) goto (I_0, C)

S → C.C , $

∝ Bβ , a

FIRST (C$) = {$}

S → C.C , $

C → .aC , $

C → .d , $

(I3) goto (I_0, a)

C → a.C , a/d

A → ∝.Bβ , a

FIRST (Ca) = {a}

FIRST (Cd) = {d}

C → a . C , a/d

C → . a C , a/d

C → . d , a/d

(I4) goto(I0, d)

C → d . , a/d

(I5) goto(I2, C)

S → CC . , $

(I6) goto(I2, a)

C → a . C , $

A → α B β , a

FIRST(ε$) = {$}

C → a . C , $

C → . a C , $

C → . d , $

(I7) goto(I2, d)

C → d . , $

(I8) goto(I3, C)

C → a C . , a/d

(I3) goto(I3, a)

C → a·C, a/d

A → αBβ, a

FIRST(Ca) = {a}

FIRST(Cd) = {d}

C → a·C, a/d

C → ·aC, a/d

C → ·d, a/d

(I4) goto(I3, d)

C → d·, a/d

(I5) goto(I6, C)

C → aC·, $

(I6) goto(I6, a)

C → a·C, $

A → αBβ, a

FIRST(C$) = {$}

C → a·C, $

C → ·aC, $

C → ·d, $

(I7) goto(I6, d)

C → d·, $

State	Action			GoTO	
	a	d	$	S	C
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		

Q: $S \rightarrow L=R \mid R$

$L \rightarrow *R \mid id$

$R \rightarrow L$

Sol:- Argumented grammer

$\boxed{S' \rightarrow .S, \$}$

$\alpha = \alpha . B\beta, a$

FIRST(ϵ, $) = FOLLOW(S)
= { $ }

S → L = R , $

S → R , $

S → . L = R , $
A → α . B β , a
FIRST(= R $) = FOLLOW(L)
= { = }

S → . R , $
A → B . , a
FIRST($) = FOLLOW(R)
= { $ }

R → L , $

L → * R , =

L → id , =

R → . L , $
A → α B , a
FIRST($) = FOLLOW(L)
= { $ }

L → * R , $

L → id , $

I0 :
S' → . S , $
I1 S → . L = R , $ (I4)
I2 S → . R , $ (I3)
I3 R → . L , $ (I2)
I4 L → . * R , = / $ (I5, I6)
I5 L → . id , = / $ (I7, I8)

goto(Io, S)

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

$A \to \alpha \cdot B\beta , a$

FIRST($) = { $ }

(I1) goto(Io, S)

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

(I2) goto(Io, L)

$S \to L \cdot = R , \$$

$R \to L \cdot , \$$

(I3) goto(Io, R)

$S \to R \cdot , \$$

(I4) goto(Io, *)

$L \to * \cdot R , = / \$$

$A \to \alpha \cdot B\beta , a$

FIRST(ϵ) = { = }

FIRST($\epsilon\$$) = { =, $ }

$R \to \cdot L , = / \$$

$A \to \alpha \cdot B\beta , a$

FIRST(ϵ) = $\alpha = \gamma$

($\Leftarrow$) $\alpha\beta$

~~$L \to * \cdot R , \$$~~

~~FIRST($\epsilon\$$) = { $ }~~

~~$R \to \cdot L , \$$~~

$L \to * \cdot R , = / \$$

$R \to \cdot L , = / \$$

$L \to \cdot * R , = / \beta$

$L \to \cdot id , = / \beta$

FIRST(L) = {*, id}

L → *·R, =

L → *·R, $

R → ·L, =

R → ·L, $

(I5) goto(I0, id)

L → id·, =/$

(I6) goto(I2, =)

S → L = ·R, $

A → α · Bβ, a

FIRST(ϵ$) = {$}

R → ·L, $

FIRST(ϵ.$) = {$}

S → L = ·R, $

R → ·L, $

L → ·*R, $

L → ·id, $

(I7) goto(I4, R)

$L \rightarrow * R \cdot, =/\$$

(I8) goto(I4, L)

$R \rightarrow L \cdot, =/\$$

(I4) goto(I4, *)

$L \rightarrow * \cdot R, =/\$$

$FIRST(\epsilon =) = \{=\}$

$FIRST(\epsilon \$) = \{\$\}$

$R \rightarrow \cdot L, =/\$$

$FIRST(\epsilon =) = \{=\}$

$FIRST(\epsilon \$) = \{\$\}$

$L \rightarrow \cdot * R, =/\$$

$L \rightarrow \cdot id, =/\$$

$R \rightarrow L \cdot, =/\$$

$L \rightarrow * \cdot R, =/\$$

$R \rightarrow \cdot L, =/\$$

$L \rightarrow \cdot * R, =/\$$

$L \rightarrow \cdot id, =/\$$

(I5) goto (I4, id)

$L \to id \cdot , =/\$$

(I9) goto (I6, R)

$S \to L = R \cdot , \$$

(I2) goto (I6, L)

$R \to L \cdot , \$$

(I10) goto (I6, *)

$L \to * \cdot R , \$$

FIRST ($\in \$$) = {$}

$R \to \cdot L , \$$

FIRST ($\in \$$) = {$}

$L \to \cdot * R , \$$

$L \to \cdot id , \$$

~~$R \to L \cdot , \$$~~

$L \to * \cdot R , \$$

$R \to \cdot L , \$$

$L \to \cdot * R , \$$

$L \to \cdot id , \$$

(I11) goto (I6, id)

$L \to id \cdot , \$$

(I12) goto (I10, R)

$L \to * R \cdot , \$$

(I2) goto (I10, L)

$R \to L \cdot , \$$

(I10) goto (I10, *)

$L \to * \cdot R , \$$

FIRST ($) = {$}

$R \to \cdot L , \$$

$L \to \cdot * R , \$$

$L \to \cdot id , \$$

~~$R \to L \cdot , \$$~~

$L \to * \cdot R , \$$

$R \to \cdot L , \$$

$L \to \cdot * R , \$$

$L \to \cdot id , \$$

(I4) goto(I10, id)

L → id., $

state	Action =	Action *	Action id	Action $	GOTO S	GOTO L	GOTO R
0		S4	S5		1	2	3
1				Accept			
2	S6			r5			
3				r2			
4		S4	S5			8	7
5	r4			r4			
6		S10	S11			2	9
7	r3			r3			
8				r5			
9				r1			
10		S10	S11			2	12
11				r4			
12				r3			

* LALR

Construct LALR parsing table for the following grammer.

$S \rightarrow CC$

$C \rightarrow aC$

$C \rightarrow d$

Core argument

$S' \rightarrow .S \;,\; \$$

$A \rightarrow \cdot B\beta,\; a$

$FIRST(\epsilon\$) = \boxed{\begin{matrix} FOLLOW(S) \\ = \$ \end{matrix}}$

$S \rightarrow .CC,\; \$$

$A \rightarrow \cdot B\beta,\; a$

$FIRST(C\$) = \boxed{\begin{matrix} FOLLOW(C) \\ = \{a, d\} \end{matrix}}$

$C \rightarrow .aC,\; a|d$

$C \rightarrow .d,\; a|d$

b) $S' \rightarrow .S,\; \$$

$S \rightarrow .CC,\; \$$

$C \rightarrow .aC,\; a|d$

$C \rightarrow .d,\; a|d$

(I1) goto(I0, S)

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

(I2) goto(I0, C)

$S \rightarrow C\cdot C, \$$ $\quad \alpha : B\beta$ $\quad FIRST(C\$) = \{\$\}$

$C \rightarrow \cdot aC, \$$

$C \rightarrow \cdot d, \$$

goto(I2, C)

$S \rightarrow C\cdot C, \$$

$C \rightarrow \cdot aC, \$$

$C \rightarrow \cdot d, \$$

(I3) goto(I0, a)

$C \rightarrow a\cdot C, ald$

$A \rightarrow \alpha B\beta, a$

$FIRST(Ca) = \{a\}$

$FIRST(Cd) = \{d\}$

$C \rightarrow \cdot aC, ald$

$C \rightarrow \cdot d, ald$

$C \rightarrow a\cdot C, ald$

$C \rightarrow \cdot aC, ald$

$C \rightarrow \cdot d, ald$

I_1 goto(I_0, d)

$C \to d \cdot, a|d$

I_5 goto(I_2, C)

$S \to CC \cdot, \$$

I_6 goto(I_2, a)

$C \to a \cdot C, \$$

FIRST($\epsilon\$$) = {$\$$}

$C \to \cdot aC, \$$

$C \to \cdot d, \$$

~~S → CC, $~~

$C \to a \cdot C, \$$

$C \to \cdot aC, \$$

$C \to \cdot d, \$$

I_7 goto(I_2, d)

$C \to d \cdot, \$$

I_8 goto(I_3, C)

$C \to aC \cdot, a|d$

I_3 goto(I_3, a)

$C \to a \cdot C, a|d$

FIRST(ϵa) = {a}
{d}

$C \to a \cdot C, a|d$

$C \to \cdot aC, a|d$

$C \to \cdot d, a|d$

I_4 goto(I_3, d)

$C \to d \cdot, a|d$

I_9 goto(I_6, C)

$C \to aC \cdot, \$$

I_6 goto(I_6, a)

$C \to a \cdot C, \$$

$C \to \cdot aC, \$$

$C \to \cdot d, \$$

I_7 goto(I_6, d)

$C \to d \cdot, \$$

go

After construction of CLR construct LALR items.

Now we will merge the states 3,6 then 4,7 & 8,9.

To construct LALR

I_0 $S' \to .S, \$$ (I_1)

R_1 $S \to .CC, \$$ (I_5)

R_2 $C \to .aC, a/d$ ($I_{8,9}$)

R_3 $C \to .d, a/d$ ($I_{4,7}$)

I_1 goto(I_0, S)

$S' \to S., \$$

I_2 goto(I_0, C)

$S \to C.C, \$$

$C \to .aC, \$$

$C \to .d, \$$

I_{36} goto(I_0, a)

$C \to a.C, a/d/\$$

$C \to .aC, a/d/\$$

$C \to .d, a/d/\$$

(I4) goto (I0, d)

$C \rightarrow d \cdot , a|d$

(I5) go

(I47) goto (I0, d)

$C \rightarrow d \cdot , a|d|\$$

(I5) goto (I2, C)

$S \rightarrow CC \cdot , \$$

(I89) goto (I3, C)

$C \rightarrow aC \cdot , a|d|\$$

	Action			goto	
states	a	d	$	S	C.
0	S_{36}	S_{47}		1	2
1			Accept		
2					5
36	S_{36}	S_{47}			89
47	R_3	R_3	R_3		
5			R_1		
89	R_2	R_2	R_2		

R3 $C \to d \cdot , a/d \Rightarrow I_4$

$C \to d \cdot , \$ \quad = I_7$

R2 $C \to aC \cdot , a/d \quad = I_8$

$C \to aC \cdot , \$ \quad = I_9$

$R_1 \quad S \to \cdot CC \cdot , \$ \quad = I_5$

2) S.T the following grammer is LR(1) but not LALR(1) ?

A: $S \to Aa \mid bAc \mid Bc \mid bBa$

$A \to d$

$B \to d$

Argument

$S' \to .S , \$$

$A \to B$

FIRST($\epsilon \$$) = FOLLOW(S) $\neq \$$

$S \to .Aa , \$$

$S \to .bAc , \$$

$S \to .Bc , \$$

$S \to .bBa , \$$

$S \rightarrow .Aa, \$$

$A \rightarrow .B\beta, a$

$FIRST(a\$) = FOLLOW(A)$

$= \{a\}$

$A \rightarrow .d, a$

$S \rightarrow .BC, \$$

$FIRST(C\$) = FOLLOW(B)$

$= \{c\}$

$B \rightarrow .d, c$

$S' \rightarrow .S, \$$

$S \rightarrow .Aa, \$$

$S \rightarrow .bAC, \$$

$S \rightarrow .BC, \$$

$S \rightarrow .bBa, \$$

$A \rightarrow .d, a$

$B \rightarrow .d, c$

(I1) goto(I0, S)

$S' \rightarrow S., \$$

(I2) goto(I0, A)

$S \rightarrow A.a, \$$

(I3) goto(I0, b)

S → b.AC, $

A → α.Bβ, a

FOLLOW(A) = {C}
FIRST(C, $)
A → .d, C

S → b.Ba, $
A → α.Bβ, a

FOR
FIRST(a$) = {a}

B → .d, a

S → b.AC, $
S → b.Ba, $
A → .d, C
B → .d, a

(I4) goto(I0, B)

S → B.C, $

(I5) goto(I0 d)

A → d., a
B → d., C

(I6) goto(I2, a)

S → Aa., $

(I7) goto(I3, A)

S → bA.C, $

(I8) goto(I3, d)

B → d., a
A → d., C

(I9) goto(I3, B)

S → bB.a, $

(I10) goto(Iu, C)

S → BC., $

(I11) goto(I7, C)

S → bAC., $

(I12) goto(I9, a)

S → bBa., $

now we will merge : 5, 8. states

To construct LALR

I_0 : $S' \to .S, \$$

r_1 : $S \to .Aa, \$$ (I_6)

r_2 : $S \to .bAc, \$$ (I_{11})

r_3 : $S \to .Bc, \$$ (I_{10})

r_4 : $S \to .bBa, \$$ (I_{12})

r_5 : $A \to .d, a$ (I_{58})

r_6 : $B \to .d, c$ (I_{58})

1) goto (I_0, S)

$S' \to S., \$$

2) goto (I_0, A)

$S \to A.a, \$$

3) goto (I_0, b)

$S \to b.Ac, \$$

$S \to b.Ba, \$$

$A \to .d, c$

$B \to .d, a$

4) goto (I_0, B)

$S \to B.c, \$$

I_{58} goto (I_0, d)

$A \to d., a|c$

$B \to d., a|c$

I_6 : goto (I_2, a)

$S \to Aa., \$$

I_7 goto (I_3, A)

$S \to bA.c, \$$

I_9 goto (I_3, B)

$S \to bB.a, \$$

I_{10} goto (I_4, c)

$S \to Bc., \$$

I_{11} goto (I_7, c)

$S \to bAc. \$$

I_{12} goto (I_9, a)

$S \to bBa., \$$

State	Action					Goto		
	a	b	c	d	$	S	A	B
0		S_3		S_{58}		1	2	4
1					Acpt			
2	S_6						7	9
3				4				
4			S_{10}					
58	γ_5 γ_6		γ_5 γ_6					
6					γ_1			
7			S_{11}					
8	9							
9	S_{12}				γ_3			
10					γ_2			
11					γ_4			
12								

P: Construct LALR parsing table for following grammer

$S \rightarrow Aa$

$S \rightarrow bAc$

$S \rightarrow dC$

$S \rightarrow bda$

$A \rightarrow d$

Augmented Grammar

$S' \to .S, \$$

$A \to \alpha B\beta, a$

Follow(S) = {$}

$S \to .Aa, \$$

Follow(A) = {a}

$S \to .bAc, \$$

$S \to .dc, \$$

$S \to .bda, \$$

I_0 $S' \to .S, \$$

r_1 $S \to .Aa, \$$ (I_2)

r_2 $S \to .bAc, \$$ (I_4)

r_3 $S \to .dc, \$$ (I_3)

r_4 $S \to .bda, \$$ (I_6)

r_5 $A \to .d, a$ (I_5, I_7)

I_1 goto(I_0, S)

$S' \to S., \$$

I_2 goto(I_0, A)

$S \to A.a, \$$

(I8) goto(I4, C)

S → dC •, $

(I9) goto(I6, C)

S → bAC •, $

(I10) goto(I7, a)

S → bda •, $

State	Action					goto	
	a	b	c	d	$	S	A
0		S_3		S_4		1	2
1					Accept		
2	S_5						
3				S_7			6
4	γ_5		S_8				
5					γ_1		
6			S_9				
7	S_{10} / γ_5						
8					γ_3		
9					γ_2		
10					γ_4		

* Dangling else Ambiguity

In the parsing methods if the ambiguous is used when the conflicts occur and then we cannot pass the input string. If the two entries that appear in the parsing table [M] M[A,a] are for reduce action then Reduce-Reduce conflict occurs.

→ If one entry is for shift action and another for reduce action on M[A,a] then Shift-Reduce conflict occurs

→ Using Dangling else Ambiguity: Consider the grammer $S \to iSeS \mid iS \mid a$

Sol: $S \to iSeS$

$S \to iS$

$S \to a$

Augument grammer

$S' \to .S$

Canonical form

I_0 $S' \to .S$

or $S \to .iSeS$ (I_6)

or $S \to .iS$ (I_4)

or $S \to .a$ (I_3)

I_1 goto(I_0, S)

$S' \to S.$

I_2 goto(I_0, i)

$S \to i.SeS$

$S \to i.S$

$S \to .iSeS$

$S \to .iS$

$S \to .a$

I_3 goto(I_0, a)

$S \to a.$

I_4 goto(I_2, S)

$S \to iS.eS$

$S \to iS.$

I_2 goto(I_2, i)

$S \to i.SeS$

$S \to i.S$

$S \to .iSeS$

$S \to .iS$

$S \to .a$

I_3 goto(I_2, a)

$S \to a.$

I_5 goto(I_4, e)

$S \to iSe.S$

$S \to .iSeS$

$S \to .iS$

$S \to .a$

I_6 goto(I_5, S)

$S \to iSeS.$

I_2 goto(I_5, i)

$S \to i.SeS$

$S \to i.S$

S → • i S e S

S → i • S

S → • a

(I3) goto(I5, a)

S → a • —

To perform Reduce

FOLLOW(S') = { $ }

S' → • S

A → • α B β

FIRST(ε) = FOLLOW(S)

FOLLOW(S') = FOLLOW(S)

= { $ }

S → i S • e S

α B β

FIRST(e$) = FOLLOW(S)

= { e }

$S \to \underset{\alpha}{i} \underset{}{S} \underset{}{e} \underset{\beta}{S} \mid$

FOLLOW(S) = FOLLOW(S)
= {$}

$S \to i S$
αB

FOLLOW(S) = FOLLOW(S)
= {$}

FOLLOW(S) = {e, $}

state	Action				goto
	e	i	a	$	S
0		S_2	S_3		1
1				Accept	
2		S_2	S_3		4
3	r_3			r_3	
4	S_5 / r_2			r_2	
5		S_2	S_3		6
6	r_1			r_1	

Consider the input 'ii', 'aea', '$' 'iiaea$' for processing

Stack	Input	Action
$0	iiaea$	shift 'i'
$012	iaea$	shift 'i'
$01212	aea$	shift 'a'
$012123	ea$	Reduce S→a
$0121254	ea$	Reduce S→iS
$01254	ea$	Reduce S→iS
$051	ea$	Error.

The choice of R2 in action[4,e] is not valid hence we will try it by ~~error~~ chosing the shift action

stack	input	Action
$0	iiaea$	shift 'i'
$012	iaea$	shift 'i'
$01212	aea$	shift 'a'
$012123	ea$	Reduce S→a
$0121254	ea$	shift e
$012125465	a$	shift a
$0121254653	$	Reduce S→a
$01212546556	$	Reduce S→iSeS

$ 0 1 2 S 4 $ Reduce S → 1 S

$ 0 S 1 $ Accept.

state	c	1	a	$	S
0		S_2	S_3		1
1				Accept	
2		S_2	S_3		4
3	r_3			r_3	
4	S_5			r_2	
5		S_2	S_3		6
6	r_1			r_1	

* Operator precedence Passer

A Grammar 'G' is said to be operator precedence if it is having following properties. 1) There is no epsilon productions on right hand side

2) No two non-terminals are adjacent to each other.

eg:- Consider the grammar

$E \rightarrow EAE \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid /$

The above grammer contains consecutive non-terminals.

→ First convert it into operator precedence grammer by removing 'A'

$E \rightarrow EAE \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid /$

$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid (E) \mid -E \mid id$

In operator precedence parsing we will first define precedence relation $<\cdot, \cdot>, \doteq$ between pair of terminals

$p <\cdot q$ ('q' gives more precedence than p)

$p \cdot> q$ ('p' gives more precedence than 'q')

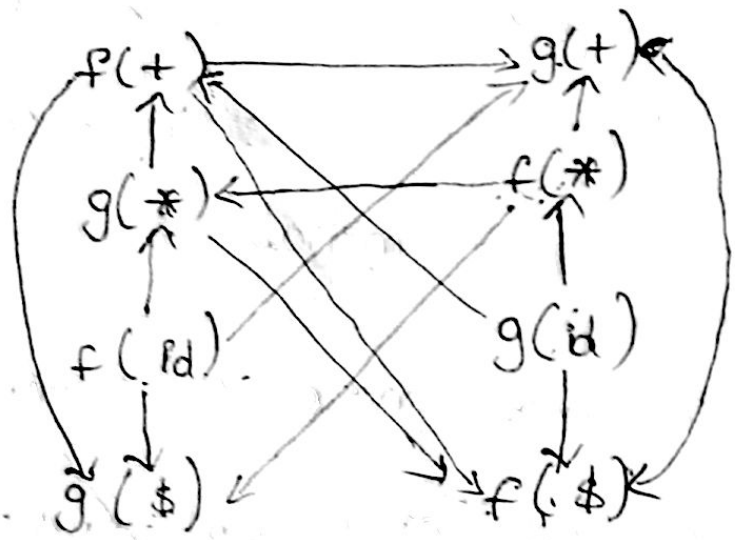
$p \doteq q$ ('p' & 'q' have same precedence)

Q:- Consider the grammar $E \to E+E \mid E*E \mid id$ construct operator precedence parser table

(g)

	+	*	id	$
+	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$
*	$\cdot>$	$\cdot>$	$<\cdot$	$\cdot>$
id	$\cdot>$	$\cdot>$	–	$\cdot>$
$	$<\cdot$	$<\cdot$	$<\cdot$	–

$f \to g$ $f \to g$
$f > g$
$f < g$ $f \leftarrow g$



~~f(id) → f($) ⇒~~

f(+) → g(+) → f($)

f(*) → g(*) → f(+) → g(+) → f($)

f(*) → g(*) → f($)

f(*) → g(+) → f($)

f(id) → g(*) → f(+) → g(+) → f($)

g(id) → f(*) → g(*) → f(+) → g(+) → f($)

precedence table

	+	*	id	$
f	2	4	4	0
g	1	3	5	0

* Construction of SLR parsing table

→ In the structure of SLR parsing table.. There are two parts i) Action 2) goto.

→ By considering basic parsing actions such as shift, reduce, accept & error, we will the parsing table

→ Input: An augmented grammar G'

Output: SLR parsing table

Algorithm

1. Initially construct set of items 'C' = {I0, I1 - - - In} where 'C' is a collection of set of LR(0) items for the input grammar G'
2. The parsing actions are based on each item I_i. The actions are as given below

a) If $A \rightarrow \alpha . a\beta$ is in I_i and

goto $(I_i, a) = I_j$ then set action of $[i, a]$ = shift j

where 'a' is terminal

2. if there is rule $A \to \alpha .$ is in I_i then set action $[i, a]$ = reduce

3. if $S' \to S.$ is in I_i then set action $[i, \$]$ accept

3. The goto part of SLR table can be filled as

If $goto(I_i, A) = I_j$ then action $[i, A] = j$

4. All the entries not defined by rule 2 & 3 are considered to be errors.

* CLR parsing table

Input : augmented grammar

output : CLR parsing table

Algorithm :-

1. Initially construct set of items

$C = \{I_0, I_1, I_2 \ldots I_n\}$ where 'C' is a collection of set of LR(1) items for the input grammar 'G'.

→ The parsing actions are based on each item I_i.

The actions are as given below

a) If $A \to \alpha . a\beta, b$ is in I_i and $goto(I_i, a) = I_j$ then set action$[I_i, a]$ = shift j.

b) If there is a production $A \to \alpha ., a$ is in I_i then action of $[I_i, a]$ = reduce

c) If there is a production $S' \to S., \$$ is in I_i action$[i, \$]$ = accept.

3) The goto part of CLR table can be filled as: if goto $[I_i, A] = I_j$ then action$[i, A] = j$.

4) All the entries not defined by rule 2 & 3 are considered to be error.

* LALR

Input : augmented grammar.

Output : LALR parsing table

Algorithm

1) Construct LR(1) set of items

2) Merge the two states I_i & I_j if the 1st component is matched and create a new state replacing the older states such as $I_{ij} = I_i \cup I_j$

3) The parsing actions are based on each item I_i.

a) If $A \to \alpha . a\beta, b$ is in I_i and goto(I_i, a) = I_j then set action $[I_i, a]$ = shift

b) If there is a production $A \to \alpha ., a$ is in I_i then action $[I_i, a]$ = reduce

If there is a production $S' \to S ., \$$ is in I_i, action $[i, \$]$ = accept

The goto part of LALR table can be filled as : if goto$(I_i, A) = I_j$ then

action [i, A] = j

5) If there is a conflict in the parsing table then the grammar is not LALR(1).

6) All the entries not defined by rule 3 & 4 are considered to be errors.

* Difference between Top-down and Bottom-up parser (or) Difference between LL & LR parsers

Top-down parser (LL)	Bottom-up parser (LR)
1) Parse tree can be built from root to leaves.	1) Parse tree can be built from leaves to root
2) These are simple to implement.	2) These are complex to implement
3) For LR(1) The first 'L' means the input is scanned from Left to	3) For LR(1) the first 'L' means the input is scanned from left to

right most derivation in reverse and ',' indicates one look-a head symbol to the parsing process

4) It is applicable to broad class of languages.

5) These are more efficient.

6) Various parsing techniques are SLR, CLR, LALR, shift reduce parser