BOTTOM-UP PARSING :-

A bottom up parser creates the parse tree of a given input starting from leaves towords the root.

⇒ A bottom up parser tries to find the right most derivation of the given I/P in the reverse order.

* bottom up parsing is also known as shift-reduce parsing becoz its two main action are shift and reduce.
  - At each shift action, the current symbol in the I/P string is pushed to a stack.
  - At each reduction step, the symbols at the top of the stack will replaced by the non-terminal at the left side of that production.

Shift reduce parsing :

A shift reduce parser tries to reduce the given input string into the starting symbol.

At each reduction step, a substring of the input matching to the right side of a production rule is replaced by the non-terminal at the left side of that production rule.

if the substring is chosen correctly, the right most derivation of that string is created in the reverse order.

ex - G:

$S \rightarrow aABb$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

$w = aaabb$

Pooja Gaud (2116)

$\Rightarrow$ aa<u>a</u>bb

$\Rightarrow$ a<u>aA</u>bb

$\Rightarrow$ a A<u>b</u>b

$\Rightarrow$ a ABb

$\Rightarrow$ S

for above reduction the rightmost derivation:

S $\rightarrow$ aABb $\Rightarrow$ ~~aaABb~~

$\Rightarrow$ a Abb

$\Rightarrow$ aaA bb

$\Rightarrow$ aaabb

ex.

G1 E $\rightarrow$ E+T | T

T $\rightarrow$ T*F | F

F $\rightarrow$ id

w = id * id

<u>id</u> * id

$\Rightarrow$ F * id

$\Rightarrow$ T * id

$\Rightarrow$ T * F

$\Rightarrow$ T

$\Rightarrow$ E

$\Rightarrow$

Corresponding right most derivation

E $\Rightarrow$ T $\Rightarrow$ T * F $\Rightarrow$ T * id

$\Downarrow$

id * id $\Leftarrow$ F * id

Handle pruning :—

A Handle of a string is a substring that matches the right side of a production rule, but every ~~string~~ substring that matches the right side of a production rule is not handle.

" A handle of a right sentential form γ ($\equiv \alpha\beta a$) is a production rule $A \to \beta$ and a position γ where the string β may be found and replaced by A to produce the previous right sententiale form in right most derivation of γ

$\Rightarrow \quad S \Rightarrow \alpha Aw \Rightarrow \alpha\beta w$ "

{ for a production $A \to \alpha$
1) if α contains non terminals, it is called sentential form of a Grammar G,
2) if α contains only terminals, then it is called a sentence of G.
}

★ "A handle is a substring that matches a right hand side of production rule in the grammar and whose reduction to the non-terminal on the left hand side of that grammar rule is a step along to reverse of a right most derivation!"

A rightmost derivation in reverse can be obtained by handle pruning.

if the grammar is unambiguous, then every right sentential form of the grammar has exactly one handle.

example:-

a- $E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id$

$w_1 = id + id * id$ ~~$w_2 = id * id$~~

right most derivation

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

$\Downarrow$

$id+id*id \Leftarrow F+id*id \Leftarrow T+id*id$

right most sentential form	Reducing production
$\underline{id} + id * id$	$F \to id$
$\underline{F} + id * id$	$T \to F$
$\underline{T} + id * id$	$E \to T$
$E + \underline{id} * id$	$F \to id$
$E + \underline{F} * id$	$T \to F$
$E + T * \underline{id}$	$F \to id$
$E + \underline{T * F}$	$T \to T*F$
$\underline{E + T}$	$E \to E+T$
E	

Stack Implementation :-

* There are four possible actions-

Shift : The next i/p symbol is shifted onto the top of the stack.
Reduce : Reduce the handle on the top of the stack by the non-terminal.
Accept : Successful completion of parsing.
Error : parser discovers a syntax error, and calls an error recovery routine.

* Initial stack contains only the end marker $, and the end of the i/p string is marked by the end-marker $.

EX. a:-

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

$w_1 = id+id*id$

Stack	i/p	Action
$	id+id*id$	shift
$id	+id*id$	reduce by F→id
$F	+id*id$	R⇒T→F
$T	+id*id$	R⇒E→T
$E	+id*id$	shift
$E+	id*id$	shift
$E+id	*id$	reduce by F→id
$E+F	*id$	reduce by T→F
$E+T	*id$	shift
$E+T*	id$	shift
$E+T*id	$	reduce by F→id
$E+T*F	$	reduce T→T*F
$E+T	$	reduce by E→E+T
$E	$	accept

ex- a:

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id$

$w \Rightarrow id * id$

Stack	I/P	Action
$	id * id $	shift
$ id	* id $	reduce by $F \to id$
$ F	* id $	reduce by $T \to F$
$ T	* id $	shift
$ T *	id $	shift
$ T * id	$	reduce by $F \to id$
$ T * F	$	reduce by $T \to T*F$
$ T	$	reduce by $E \to T$
$ E	$	Accept.

right most derivation

K - I/P symbols used as a look-head symbol to determine parser action

Conflict during shift reduce parsing :- LR(K)

Input scanned from left to right

for some CFG's shift reduce parser can not be used.

⇒ Stack contents and the next I/P symbol may not decide action —

- shift / reduce conflict : whether make a shift operation or a reduction.
- reduce / reduce conflict :- the parser can not decide which of the several reduction to make.

⇒ If a shift reduce parser can not be used for a grammar, that grammar is called as non-LR(K) grammar. (an ambiguous grammar can never be a LR grammar)

~~LR(K)~~ ⇒

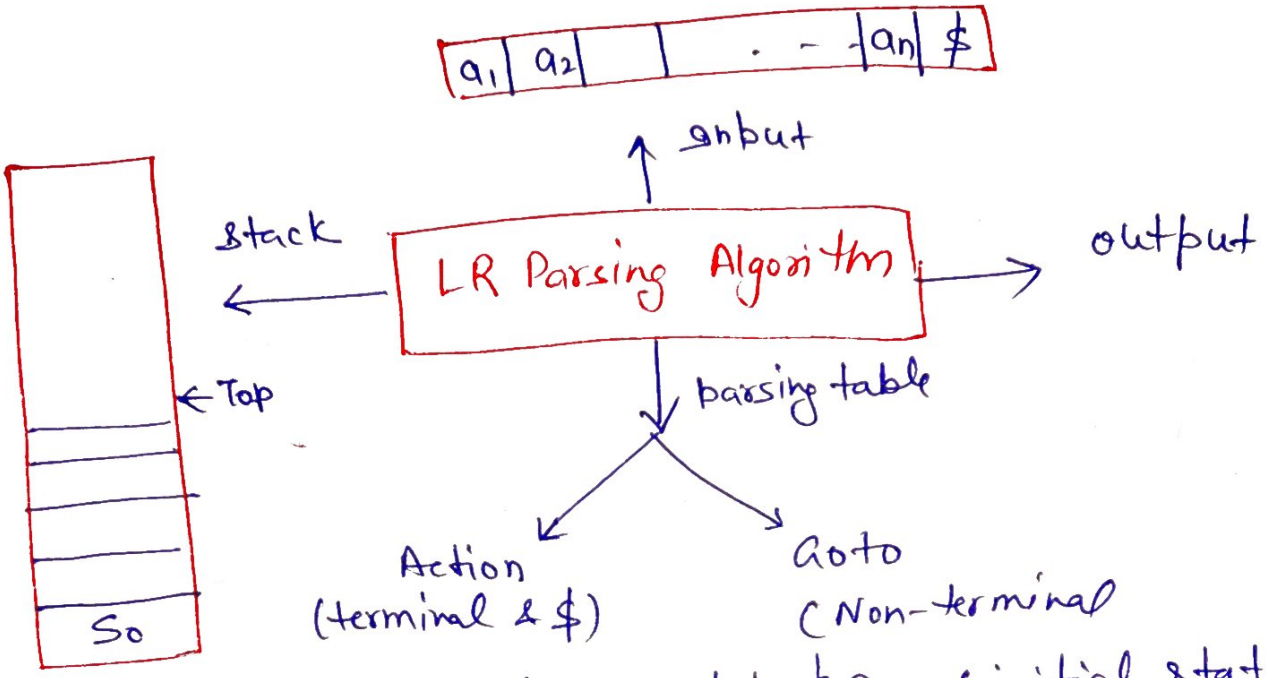
LR PARSING :- (L- left to right scan, R→ right most derivation)

LR parsing is most general non-backtracking shift-reduce parsing.

The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars that can be parsed with predictive parsers.

⇒ LL(1) grammar ⊂ LR(1) Grammar

parser configuration :-



the stack initially contain $So, (initial state the symbol at the top of stack and current i/p symbol decides the parser action by consulting the parsing table action.

Parser Action :—

1) Shift s :
shift the next input symbol and state s onto the stack
ex—

Stack	I/P	Action
$ 0	id + id $	shift 5 (S5)
0 id 5	+ id $	

(2) Reduce—

ex—

Stack	I/P	Action	O/P
0 id 5	* id $	r6	F → id
⇒ 0 F 3	* id $		

r6 mean - reduce according to production no - 6

(3) Accept :
parsing successfully completed.

(4) Error : (an empty entry in parsing table)
parser detect an error.

example →

Consider the Grammar

1) E → E+T
2) E → T
3) T → T*F
4) T → F
5) F → (E)
6) F → id

w = id+id * id

State	Action						Goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
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			

Stack	I/P	Action	O/P
0	id + id * id $	s5	
0 id 5	+ id * id $	r6	F → id
0 F 3	+ id * id $	r4	T → F
0 T 2	+ id * id $	r2	E → T
0 E 1	+ id * id $	s6	
0 E 1 + 6	id * id $	s5	
0 E 1 + 6 id 5	* id $	r6	F → id
0 E 1 + 6 F 3	* id $	r4	T → F
0 E 1 + 6 T 9	* id $	s7	
0 E 1 + 6 T 9 * 7	id $	s5	
0 E 1 + 6 T 9 * 7 id 5	$	r6	F → id
0 E 1 + 6 T 9 * 7 F 10	$	r3	T → T * F
0 E 1 + 6 T 9	$	r1	E → E+T
0 E 1	$	Accept	

Example :-

Stack	I/P	Action	O/P
0	id*id+id$	S5	
0id5	*id+id$	r6	F→id
0F3	*id+id$	r4	T→F
0T2	*id+id$	S7	
0T2*7	id+id$	S5	
0T2*7id5	+id$	r6	F→id
0T2*7F10	+id$	r3	T→T*F
0T2	+id$	r2	E→T
0E1	+id$	S6	
0E1+6	id$	S5	
0E1+6id5	$	r6	F→id
0E1+6F3	$	r4	E→F
0E1+6T9	$	r1	E→E+T
0E1	$	Accept	

X X X

Kernal items :

the initial item S'→.S and all items whose dot are not at the left end.

Non-kernal item:-

all items with their dot at the left end except for for S'→.S

Constructing SLR parsing table:-

If a grammar G has an SLR parsing table, it is called SLR Grammar.

Every SLR Grammar is unambiguous, but every unambiguous grammar is not SLR.

Augmented Grammar :-

Add new production rule $S' \to S$, where S' is new starting symbol and S is old starting symbol

ex

$S \to CC$
$C \to eC|d$

$\Rightarrow$

$S' \to S$
$S \to CC$
$S \to eC|d$

} Augmented grammar

① LR(0) items :-

An LR(0) item of a grammar G is a production of G with a dot at some position of the right side.

ex. $A \to abB$

possible LR(0) items—

$A \to .abB$
$A \to a.bB$
$A \to ab.B$
$A \to abB.$

if $A \to \epsilon$ is a production the LR(0) item —
$A \to .$

→ sets of LR(0) items will be the states of action and goto table of SLR parser.

→ A collection of sets of LR(0) items (canonical LR(0) collection) is the basic for constructing SLR parser

$A \to X.YZ$ indicates that a string derivable from X has been seen so far on the i/p and we hope to see a string derivable from YZ next on the i/p.

Pragya

② CLOSURE operation :-

If I is a set of LR(0) items for a grammar G, then closure(I) is the set of LR(0) items constructed from I by rules-

① Initially every LR(0) item in I is added to closure(I)

② If $A \to \alpha . B\beta$ is in closur(I) and $B \to \gamma$ is a production rule of G, then $B \to .\gamma$ will be in the closur(I). We will apply this rule until no more new LR(0) item can be added to closure (I).

example-

G:

$E' \to E$
$E \to E+T$
$E \to T$
$T \to T*F$
$T \to F$
$F \to id$

$\Rightarrow$

closure ($E' \to .E$) $\Rightarrow$

I_0 { $E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .id$ }

③ GOTO operation :-

If I is a set of LR(0) items and X is a grammar symbol (terminal or non-terminal), then Goto(I, X) is defined as-

- If $A \to \alpha . X\beta$ in I then every item in ~~closure~~ closur ($A \to \alpha X . \beta$) will be in goto (I, X)

example-

I_0
$E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .id$

goto (I_0, E) = { $E' \to E.$, $E \to E. +T$ }
goto (I_0, T) = { $E \to T.$, $T \to T. *F$ }
goto (I_0, F) = { $T \to F.$ }
goto (I_0, id = { $F \to id.$ }

Construction of Canonical LR(0) Collection –

To create the SLR parsing table for a grammar G, we will create the canonical LR(0) collection of the Grammar G'.

Algorithm :

C is { closur ({S' → .S}) }
repeat the following until no more set of LR(0) items can be added to C.
for each I in C and each grammar symbol X
if goto (I,X) is not empty and not in C
then add goto (I, X) to C

Example :-

G: E → E+T | T
T → T*F | F
F → id

Step-1 augment the grammar

E' → E
1) E → E+T
2) E → T
3) T → T*f
4) T → f
5) f → id

Canonical LR(0) items are

I0 E' → .E
E → .E+T
E → .T
T → .T*F
T → .f
f → .id

I1 E' → E.
E → E.+T

I2 E → T.
T → T.*f

I3 T → f.

I4 f → id.

I5 E → E+.T
T → .T*F
T → .f
F → .id

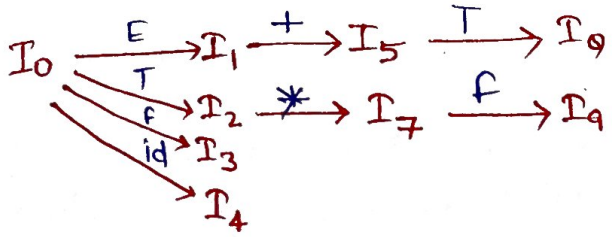
I7 T → T*.f
f → .id

Pragya Gaur

I_0

$E \to E+T.$

$T \to T.*f$



I_9

$T \to T*F.$

Progya Ganv

LR Parsing :-

LR(0) table Construction-

for each edge

if X is terminal put shift J at [I, X]

if X is Non-terminal, put goto J at [I, X]

if I contains $S' \to S.$, put accept at [I, $]

if I contains $A \to \alpha.$ where $A \to \alpha.$ has grammar rule no. n-

for each terminal x, put reduce n at [I, X]

ex.

$S' \to S$

1) $S \to (L)$

2) $S \to x$

3) $L \to S$

4) $L \to L, S$

w = (x, (x))

	x	(	)	,	$	S	L
0	s1	s2				3	
1	r2	r2	r2	r2	r2		
2	s1	s2				6	4
3					acc		
4			s5	s7			
5	r1	r1	r1	r1	r1		
6	r3	r3	r3	r3	r3		
7	s1	s2				8	
8	r4	r4	r4	r4	r4		

0	(x,(x))$	s2
0(2	x,(x))$	s1
0(2x1	,(x))$	r2
0(2S6	,(x))$	r3
0(2L4	,(x))$	s7
0(2L4,7	(x))$	s2
0(2L4,7(2	x))$	s1
0(2L4,7(2x1	))$	r2
0(2L4,7(2S6	))$	r3
0(2L4,7(2L4	))$	s5
0(2L4,7(2L4)5	)$	r1
0(2L4,7S8	)$	r4
0(2L4	)$	s5
0(2L4)5	$	r1
0S3	$	acc,

(2/29.1)

G: $S \to +SS$

$S \to -SS$

$S \to a$

LR(0) items

I_0

$S' \to .S$

$S \to .+SS$

$S \to .-SS$

$S \to .a$

I_1 $S' \to S.$

I_2

$S \to +.SS$

$S \to .+SS$

$S \to .-SS$

$S \to .a$

I_3

$S \to -.SS$

$S \to .+SS$

$S \to .-SS$

$S \to .a$

I_4 $S \to a.$

I_5

$S \to +S.S$

$S \to .+SS$

$S \to .-SS$

$S \to .a$

I_6

$S \to -S.S$

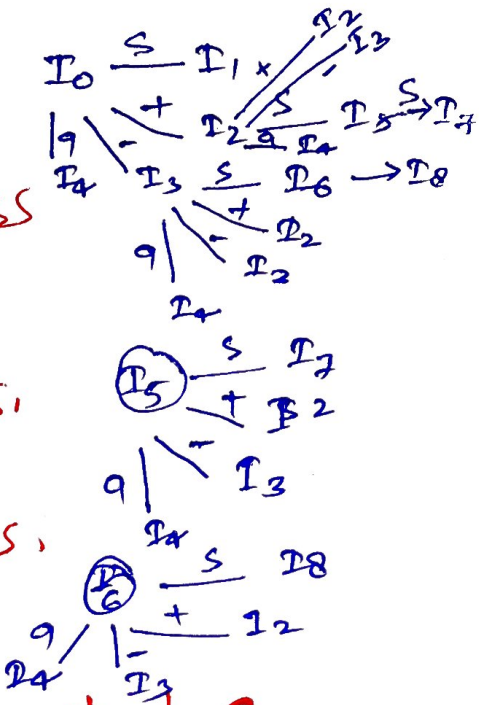
$S \to .+SS$

$S \to .-SS$

$S \to .a$

I_7 $S \to +SS.$

I_8 $S \to -SS.$



	+	-	a	$	S
0	s2	s3	s4		1
1				acc	
2	s2	s3	s4		5
3	s2	s3	s4		6
4	r3	r3	r3	r3	
5	s2	s3	s4		7
6	s2	s3	s4		8
7	r1	r1	r1	r1	
8	r2	r2	r2	r2	

for $w = +aa$

Stack	I/P	Action	O/P
0	+aa$	s2	
0+2	aa$	s4	
0+2a4	a$	r3	S→a
0+2S5	a$	s4	
0+2S5a4	$	r3	S→a
0+2S5S7	$	r1	S→+SS
0S1	$	Acc	

SLR

Rules to Construct the parsing table :—

1) Construct the canonical collection of sets of LR(0) items for augmented Grammare G'.
   $C \leftarrow \{I_0, \ldots I_n\}$

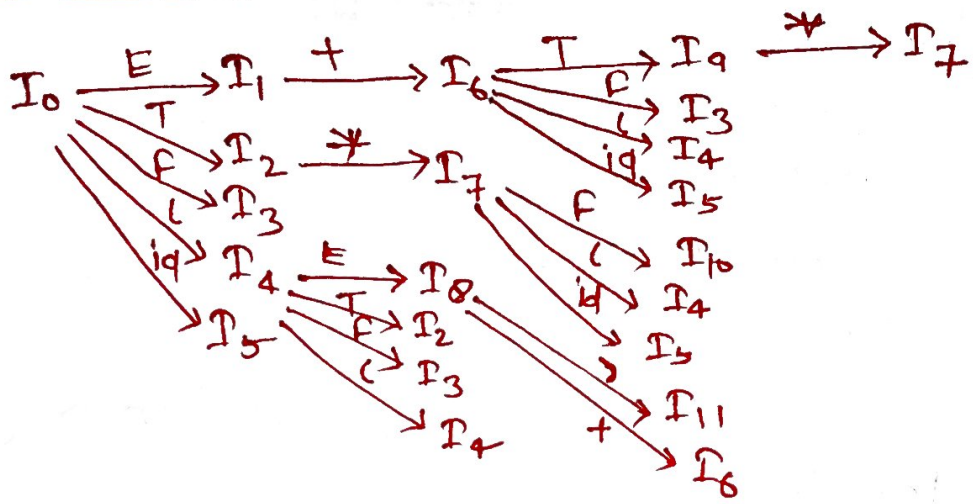
2) Create the parsing action table as follows—
   a) If a terminal a, $A \to \alpha . a\beta$ in I_i and goto$(I_i, a) = I_j$ then action $[i, a]$ is shift j.
   b) if $A \to \alpha .$ is in I_i, then action $[i, a]$ is reduce $A \to \alpha$ for all a in FOLLOW(A) where $A \neq S'$
   c) If $S' \to S.$ is in I_i, then action $[i, \$]$ is accept.
   d) If any conflicting action generated by these rules then the grammar is not SLR(1).

3) Create the parsing goto table—
   for all Non-terminal A, if goto$(I_i, A) = I_j$ then goto$(i, A) = j$

4) All entries not defined by (2) and (3) are errors.

Example :-

G: 1) E → E+T
2) E → T
3) T → T*f
4) T → f
5) f → (E)
6) f → id



Augment the grammar –

G' →
E' → E
E → E+T
E → T
T → T*f
T → f
f → (E)
f → id

Canonical LR(0) item set –

I_0
E' → ·E
E → ·E+T
E → ·T
T → ·T*f
T → ·f
f → ·(E)
f → ·id

I_1 E' → E·
E → E·+T

I_2 E → T·
T → T·*f

I_3 T → f·

I_4 f → (·E)
E → ·E+T
E → ·T
T → ·T*f
T → ·f
f → ·(E)
f → id

I_5 f → id·

I_6 E → E+·T
T → ·T*f
T → ·f
f → ·(E)
f → ·id

I_7 T → T*·f
f → ·(E)
f → ·id

I_8 f → (E·)
E → E·+T

I_9 E → E+T·
T → T·*f

I_{10} T → T*f·

I_{11} f → (E)·

Prajya

for $I_0 \rightarrow$

goto $(I_0, E) \Rightarrow I_1$ Action $\Rightarrow$ goto $(0, E) \Rightarrow 1$

goto $(I_0, T) \Rightarrow I_2$ Action $\Rightarrow$ goto $(0, T) \Rightarrow 2$

goto $(I_0, f) \Rightarrow I_3$ Action $\Rightarrow$ goto $(0, f) \Rightarrow 3$

goto $(I_0, ($ $) \Rightarrow I_4$ Action $\Rightarrow$ $(0, ($ $) \Rightarrow S4$

goto $(I_0, id) \Rightarrow I_5$ Action $\Rightarrow$ $(0, id) \Rightarrow S5$

for I_1

goto $(I_1, +) \Rightarrow I_6$ Action $(1, +) \Rightarrow S6$

$E' \rightarrow E.$ Action $(1, \$) \Rightarrow$ Accept

for I_2

$E \rightarrow T.$ $\Rightarrow$ Action $(2, a) \Rightarrow r2$

a is everything in FOLLOW(E)

goto $(I_2, *) \Rightarrow I_7$ Action $\Rightarrow (2, *) \Rightarrow S7$

for I_3

$T \rightarrow f.$ $\Rightarrow$ Action $(3, a) \rightarrow r4$

a is everything in FOLLOW(T)

for I_4

goto $(I_4, E) \Rightarrow I_8$ $\Rightarrow$ Action $\Rightarrow$ goto $(4, E) \Rightarrow 8$

goto $(I_4, T) \Rightarrow I_2$ $\Rightarrow$ Action $\Rightarrow$ goto $(4, T) \Rightarrow 2$

goto $(I_4, f) \Rightarrow I_3$ $\Rightarrow$ Action $\Rightarrow$ goto $(4, f) \Rightarrow 3$

goto $(I_4, ($ $) \Rightarrow I_4$ $\Rightarrow$ Action $(4, ($ $) \Rightarrow S4$

goto $(I_4, id) \Rightarrow I_5$ $\Rightarrow$ Action $(4, id) \Rightarrow S5$

for I_5

$f \rightarrow id.$ $\Rightarrow$ Action $(5, a) \Rightarrow r6$

a is everything in FOLLOW(F).

for I_6

goto $(I_6, T) \Rightarrow I_9$ — Action $\Rightarrow$ goto $(6, T) \Rightarrow 9$

goto $(I_6, f) \Rightarrow I_3$ — Action $\Rightarrow$ goto $(6, f) \Rightarrow 3$

goto $(I_6, ($ $) \Rightarrow I_4$ — Action $\Rightarrow$ $(6, ($ $) \Rightarrow S4$

goto $(I_6, id) \Rightarrow I_5$ — Action $\Rightarrow$ $(6, id) \Rightarrow S5$

for I_7

goto $(I_7, f) \Rightarrow I_{10}$ — Action $\Rightarrow$ goto $(7, f) \Rightarrow 10$

goto $(I_7, ($ $) \Rightarrow I_4$ — Action $\Rightarrow$ ~~goto~~ $(7, ($ $) \Rightarrow S4$

goto $(I_7, id) \Rightarrow I_5$ — Action $\Rightarrow$ $(7, id) \Rightarrow S5$

for I_8

goto $(I_8,))$ $\Rightarrow I_{11}$ — Action $\Rightarrow$ $(8,)) \Rightarrow S11$

goto $(I_8, +) \Rightarrow I_6$ — Action $\Rightarrow$ $(8, +) \Rightarrow S6$

for I_9

goto $(I_9, *) \Rightarrow I_7$ — Action ~~goto~~ $(9, *) \Rightarrow S7$

$E \rightarrow E+T. \Rightarrow$ Action $(9, a) \Rightarrow r1$

a is everything in FOLLOW(E),

for I_{10}

$T \rightarrow T*f. \Rightarrow$ Action $\Rightarrow (10, a) = r3$

where a is everything in FOLLOW(T)

for I_{11}

$f \rightarrow (E). \Rightarrow$ Action $(11, a) \Rightarrow r5$

where a is everything in FOLLOW(F)

Pragya

	Action						goto		
State	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
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				8	10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Construct an SLR parsing table for the following grammar:

$R \rightarrow R \mid R$

$R \rightarrow RR$

$R \rightarrow R^*$

$R \rightarrow (R)$

$R \rightarrow a$

$R \rightarrow b$

Resolve the parsing action conflicts in such a way that regular expression will be parsed normally.

Answer: In addition to the rules given above, one extra rule $R' \rightarrow R$ as the initial item. Following the procedures for constructing the LR(1) parser, here is the resulting state transition:

The initial state for the SLR parser is:

I_0: (0) $R' \rightarrow .R$
(1) $R \rightarrow .R \mid R$
(2) $R \rightarrow .RR$
(3) $R \rightarrow .R^*$
(4) $R \rightarrow .(R)$
(5) $R \rightarrow .a$
(6) $R \rightarrow .b$

$\text{goto}(I_0,R) = I_1$: $R' \rightarrow R.$
$R \rightarrow R. \mid R$
$R \rightarrow R.R$
$R \rightarrow R.^*$
$R \rightarrow .R \mid R$
$R \rightarrow .RR$
$R \rightarrow .R^*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$\text{goto}(I_0,() = I_2$: $R \rightarrow (.R)$
$R \rightarrow .R \mid R$
$R \rightarrow .RR$
$R \rightarrow .R^*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$\text{goto}(I_0,a) = I_3$: $R \rightarrow a.$

$\text{goto}(I_0,b) = I_4$: $R \rightarrow b.$

$\text{goto}(I_1,|) = I_5$: $R \rightarrow R \mid . R$
$R \rightarrow .R \mid R$
$R \rightarrow .RR$
$R \rightarrow .R^*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$\text{goto}(I_1,R) = I_6$: $R \rightarrow RR.$
$R \rightarrow R. \mid R$
$R \rightarrow R.R$
$R \rightarrow R.^*$
$R \rightarrow .R \mid R$
$R \rightarrow .RR$
$R \rightarrow .R^*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$\text{goto}(I_1,^*) = I_7$: $R \rightarrow R^*.$

$\text{goto}(I_1,() = I_2$

$\text{goto}(I_1,a) = I_3$

$\text{goto}(I_1,b) = I_4$

$goto(I_2,R)= I_8$: $R \rightarrow (R.)$
$R \rightarrow R. \mid R$
$R \rightarrow R.R$
$R \rightarrow R.*$
$R \rightarrow .R \mid R$
$R \rightarrow .RR$
$R \rightarrow .R*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$goto(I_2,( )= I_2$
$goto(I_2,a)= I_3$
$goto(I_2,b)= I_4$

$goto(I_5,R)= I_9$: $R \rightarrow R \mid R.$
$R \rightarrow R. \mid R$
$R \rightarrow R.R$
$R \rightarrow R.*$
$R \rightarrow .R \mid R$
$R \rightarrow .RR$
$R \rightarrow .R*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$goto(I_5,( )= I_2$
$goto(I_5,a)= I_3$
$goto(I_5,b)= I_4$

$goto(I_6,|)= I_5$
$goto(I_6,R)= I_6$
$goto(I_6,*)= I_7$
$goto(I_6,( )= I_2$
$goto(I_6,a)= I_3$
$goto(I_6,b)= I_4$

$goto(I_8,))= I_{10}$: $R \rightarrow (R).$
$goto(I_8,|)= I_5$
$goto(I_8,R)= I_6$
$goto(I_8,*)= I_7$
$goto(I_8,( )= I_2$
$goto(I_8,a)= I_3$
$goto(I_8,b)= I_4$

$goto(I_9,|)= I_5$
$goto(I_9,R)= I_6$
$goto(I_9,*)= I_7$
$goto(I_9,( )= I_2$
$goto(I_9,a)= I_3$
$goto(I_9,b)= I_4$

From the grammar, we have computed Follow(R)={ |, *, (,), a, b, $ }. In the sets of items mentioned above, we can easily find shift-reduce conflicts, e.g. states I_6 and I_9, but we can use the operator precedence and associativity mentioned in Section 3.3 to resolve it. Here is the operator precedence:

( ) > * > catenate† > |

And all these operators are left-associative. Based on this extra information, we construct the parssing table as follows:

State	Action							Goto
	|	*	(	)	a	b	$	R
0			s2		s3	s4		1
1	s5	s7	s2		s3	s4	acc	6
2			s2		s3	s4		8
3	r5	r5	r5	r5	r5	r5	r5	
4	r6	r6	r6	r6	r6	r6	r6	
5			s2		s3	s4		9
6.	r2	s7	r2	r2	r2	r2	r2	6
7	r3	r3	r3	r3	r3	r3	r3	
8	s5	s7	s2	s10	s3	s4		6
9	r1	s7	s2	r1	s3	s4	r1	6
10	r4	r4	r4	r4	r4	r4	r4	

† The catenate operator is implicit, and always exists between two consecutive R nonterminals.

✱ Shift/Reduce & reduce/reduce Conflicts in SLR parsing

1) if a state does not know whether it will make a shift operation or reduction for a terminal, we say that there is a shift/reduce conflict.

2) if a state does not know whether it will make a reduction operation using the production rule i or j for a terminal, we say that there is a reduce/reduce conflict.

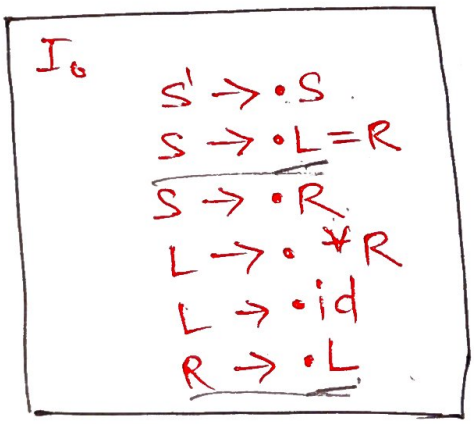
Ex.

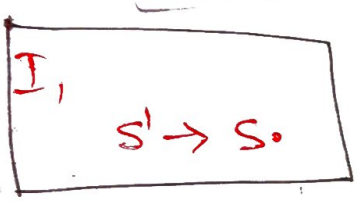
$S \to L=R$
$S \to R$
$L \to *R$
$L \to id$
$R \to L$

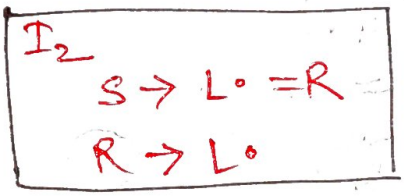
$\Rightarrow$

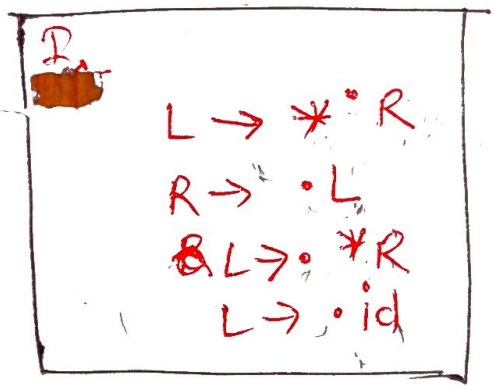
$S' \to S$
1) $S \to L=R$
2) $S \to R$
3) $L \to *R$
4) $L \to id$
5) $R \to L$

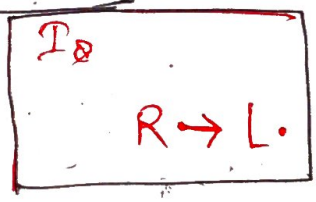
$\Rightarrow$

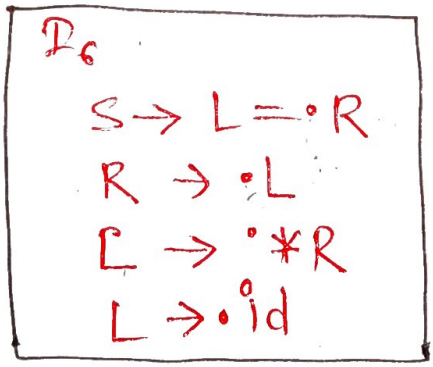


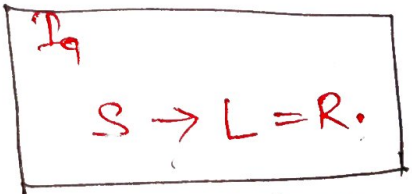












goto (I_0, S) → I_1
GoTo (0, S) → 1

goto (I_0, L) → I_2
GoTo (0, L) → 2

goto (I_0, R) → I_3
GoTo (0, R) → 3

goto (I_0, *) → I_4
Action (0, *) → S4

goto (I_0, id) → I_5
Action (0, id) → S5

~~goto (I_1, $)~~
I_1 S' → S·
Action (1, $) → acc

goto(I_2, =) → I_6

Action(2, =) → S6

R → L•

Action(2, follow(R)) → γ5

Action(2, =) → γ5

Action(2, $) → γ5

S6/γ5

follow(S) → { $ }
follow(L) → { =, $ }
follow(R) → { =, $ }

Shift/reduce Conflict as Action(2, =) → S6/γ5

so the grammer is not SLR(1) grammer

example :-

S → AaAb
S → BbBa
A → ϵ
B → ϵ

⇒

{ S' → S
S → AaAb
S → BbBa
A → ϵ
B → ϵ }

I_0 S' → .S
S → .AaAb
S → .BbBa
A → .
B → .

I_1 S' → S.

I_2 S → A.aAb

I_3 S → B.bBa

I_4 S → Aa.Ab
A → .

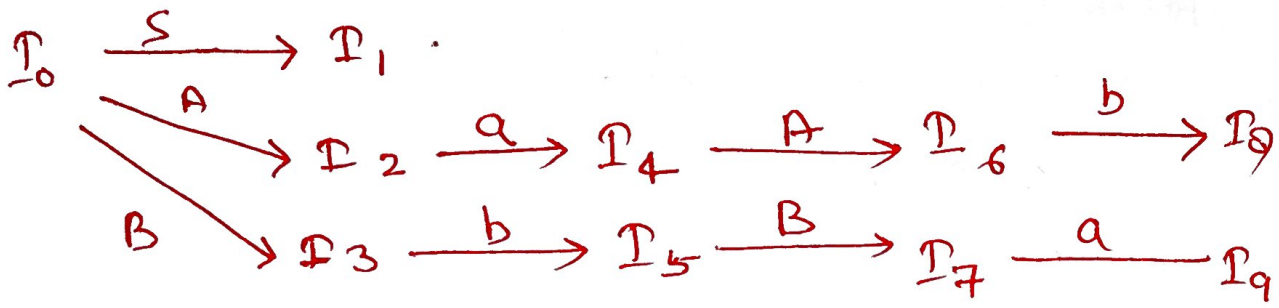
I_5 S → Bb.Ba
B → .

but

I_6 S → AaA.b

I_7 S → BbB.a

I_8 S → AaAb.



for I_0

goto $(I_0, S) \Rightarrow I_1$ Action $\Rightarrow$ goto $(0, S) \Rightarrow 1$

goto $(I_0, A) \Rightarrow I_2$ Action $\Rightarrow$ goto $(0, A) \Rightarrow 2$

goto $(I_0, B) \Rightarrow I_3$ Action $\Rightarrow$ goto $(0, B) \Rightarrow 3$

$A \to \cdot\,\cdot \Rightarrow$ Action $\Rightarrow$ $(0, a) \Rightarrow r3$

where a is everything in FOLLOW(A) $\Rightarrow \{a, b\}$

$\Rightarrow$ $(0, a) \Rightarrow r3$, $(0, b) = r3$

$B \to \cdot$ Action $\Rightarrow$ $(0, b) \Rightarrow r4$

FOLLOW(B) $= \{a, b\}$

$\Rightarrow$

$\Rightarrow$ $(0, a) \Rightarrow r4$, $(0, b) \Rightarrow r4$

$\Rightarrow$ reduce/reduce conflict

as Action $(0, a)$ & $(0, b) \Rightarrow$ r3/r4

	a	b	$	S	A	B
0	r3/r4	r3/r4				
1						
2						
3						
4						
5						
6						
7						
8						

Explanation:

⇒ Consider the production-

$S \rightarrow AaAb$

FOLLOW(A) = {a, b}

FOLLOW(B) = {a, b}

So parser can do the reduction by the $A \rightarrow \epsilon$ or $B \rightarrow \epsilon$ in the state I_0, if the next i/p symbol happens to be either a or b.

because both a and b are in FOLLOW(A)

this is reduce/reduce conflict.

But reduction will be proper only when next i/p symbol is a, if anyhow i/p is b then parsing fails.

These kind of problems can be solved if list of terminals (lookheads) are attached with the items.