Chapter: 3 bottom-up parsing

Date: / /
Page No.

Bottom-up parsing:

→ In bottom-up parsing method, the input string is taken first and we try to reduce this string with the help of grammar and try to obtain the start symbol.

→ The parse tree is constructed from bottom to up that is from leaves to root.

→ In this process, the input symbols are placed at the leaf nodes after successful parsing.

→ The bottom-up parse tree is created starting from leaves, the leaf nodes together are reduced further to internal nodes, these internal nodes are further reduced and eventually a root node is obtained.

→ In this process, basically parser tries to identify R.H.S. of production rule and replace it by corresponding L.H.S. This activity is called reduction.

→ Thus the crucial but prime task in bottom-up parsing is to find the productions that can be used for reduction.

→ The bottom-up parse tree construction process indicates that the tracing of derivations are to be done in reverse order.

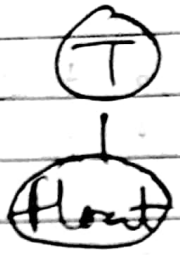
S → TL ;
T → int | float
L → L , id | id

input string : float id , id , id ;

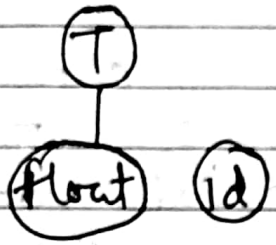
Parse tree :

1)

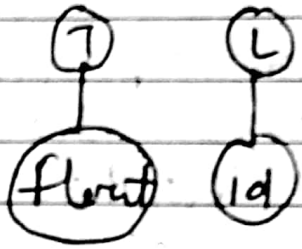
2)



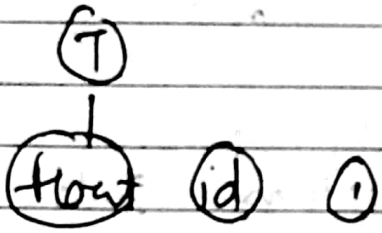
3)



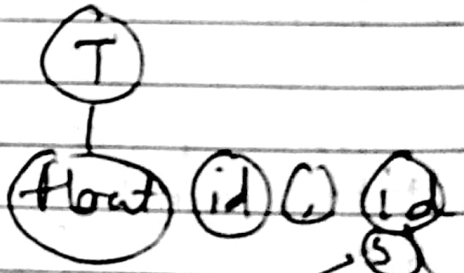
4)



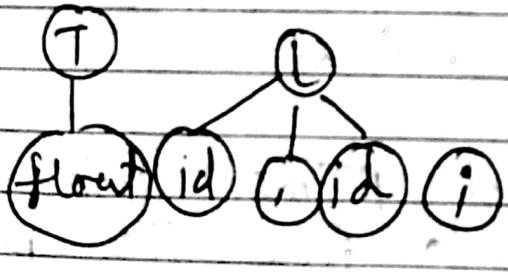
5)



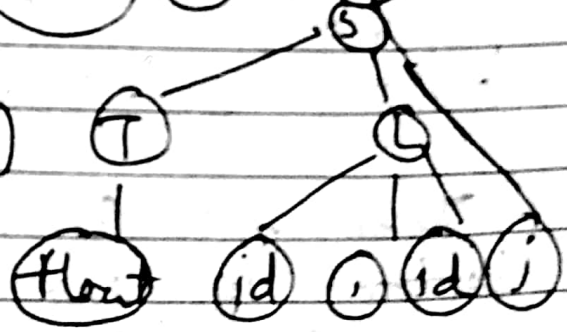
6)



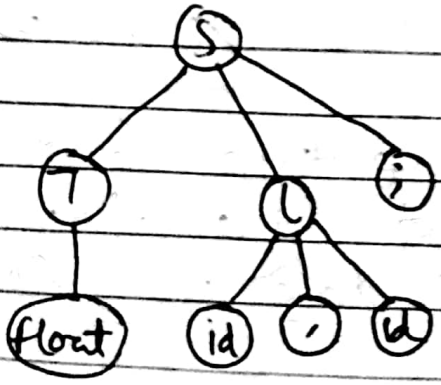
7)



8)



9)



10) Sentential form:

float id, id ;
T id, id ;
T L , id ;
T L ;
S

LR Parsers: Table driven parsers:

This is the most efficient method of bottom up parsing which can be used to parse the large class of CFG. This method is also called LR(k) parsing.

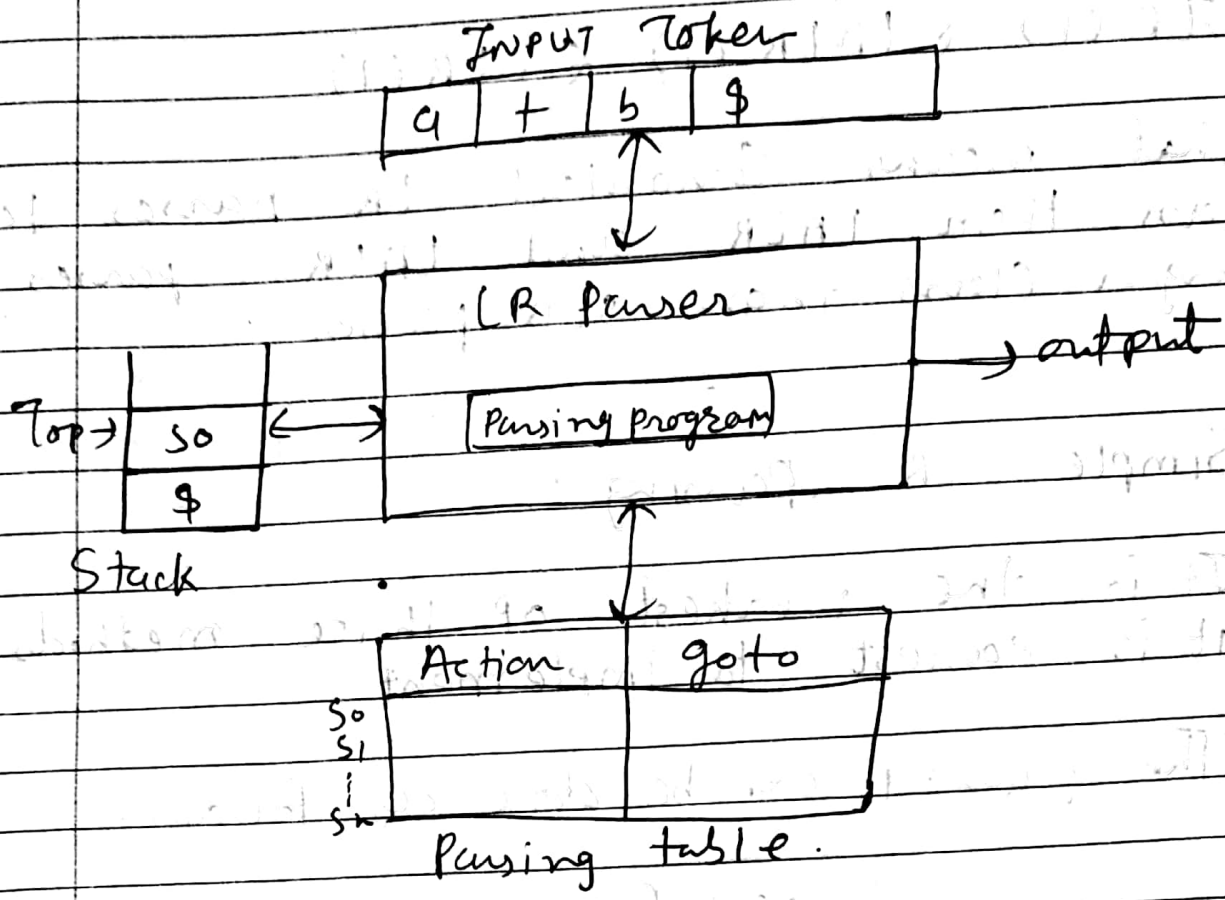
L Stands for ~~right most~~ left to right Scanning.

R Stands for right most derivation in reverse.

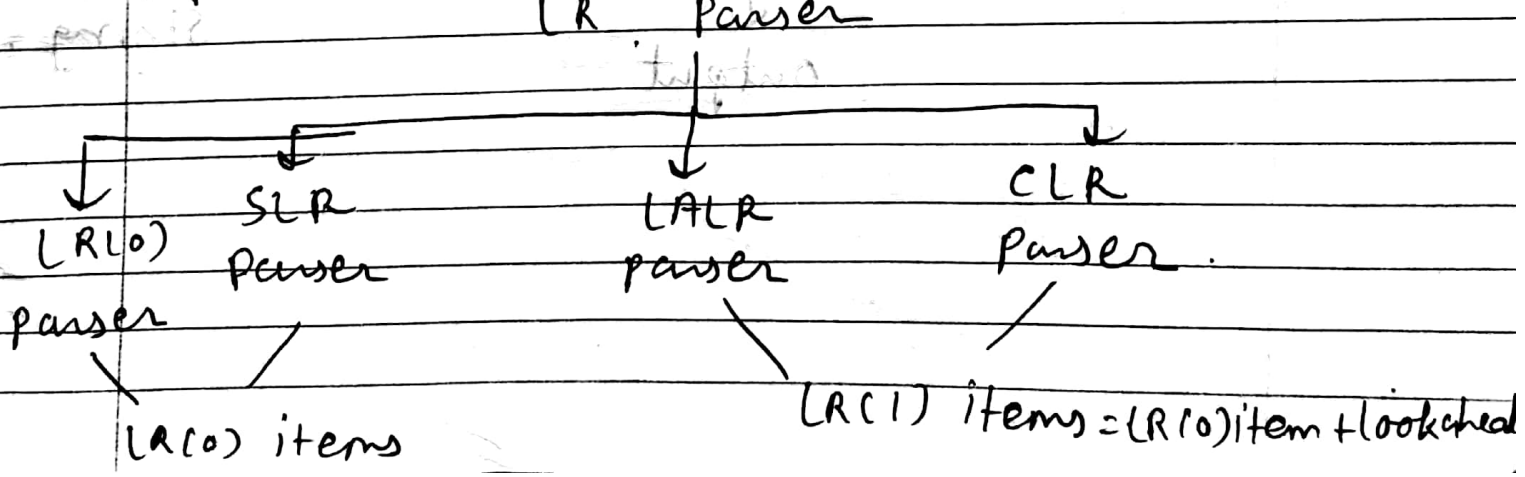
k is number of input symbols. When k is omitted k is assumed to be 1

Date: / /
Page No.

Structure of LR parsers:



→ It Consists of input buffer for storing the input string, a stack for storing the grammar symbols, output and parsing table comprised of two parts, namely action and goto. There is one parsing program which is actually a driving program and reads the input symbol one at a time from the input buffer.



Date: / /
Page No.

1 LR(0):

$S \to AaAb \mid BbBa$
$A \to \varepsilon$
$B \to \varepsilon$

$S \to AaAb$ 1
$S \to BbBa$ 2
$A \to \varepsilon$ 3
$B \to \varepsilon$ 4

Augmented grammer:

$S' \to S$
$S \to AaAb$
$S \to BbBa$
$A \to \varepsilon$
$B \to \varepsilon$

Canonical Set of LR(0) Item:

I_0: Closure $(\{ S' \to .S \})$
$S' \to .S$
$S \to .AaAb$
$S \to .BbBa$
$A \to .$ //
$B \to .$ //

I_1: goto $(I_0, S) = \{ S' \to S. \}$ //

I_2: goto $(I_0, A) = \{ S \to A.aAb \}$

I_3: goto $(I_0, B) = \{ S \to B.bBa \}$

I_4: goto $(I_2, a) = \{ S \to Aa.Ab$
$A \to . \}$

I_5: goto $(I_3, b) = \{ S \to Bb.Ba$
$B \to . \}$

I_6 : goto (I_4 , A) = { S → AqA · b }

I_7 : goto (I_5 , B) = { S → BbB · q }

I_8 : goto (I_6 , b) = { S → AqAb · }

I_9 : goto (I_7 , q } = { S → BbBq · }

LR(0) parsing table :

	Action			goto		
	q	b	$	S	A	B
0	r3/r4	r3/r4	r3/r4			
1			Accept			
2	S4					
3		S5				
4	r3	r3	r3			
5	r4	r4	r4			
6		S8				
7	S9					
8	r1	r1	r1			
9	r2	r2	r2			

grammer is not LR(0).

The relative power of these parsers is

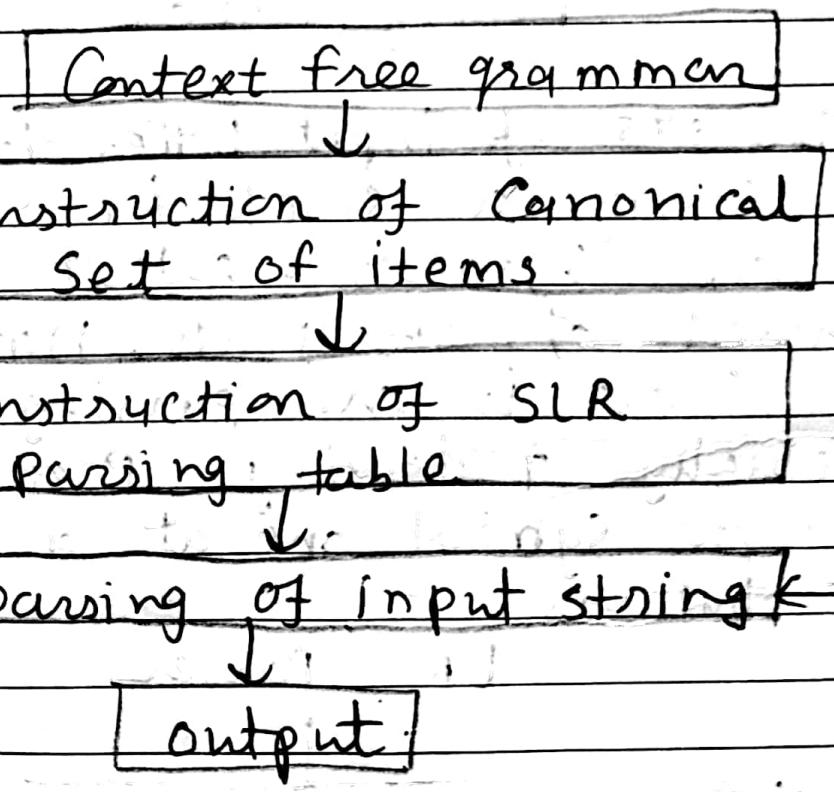
SLR(1) ≤ LALR(1) ≤ CLR(1)

That means Canonical LR parses larger class than LALR and LALR parses larger class than SLR parser.

1) **Simple LR parsing:**

It is the weakest of three methods but it is easiest to implement.

The parsing can be done as follows:



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

Thus, production $A \to XYZ$ yields the four items:

$A \to \cdot XYZ$
$A \to X \cdot YZ$
$A \to XY \cdot Z$
$A \to XYZ \cdot$

→ The production $A \to \varepsilon$ generates only one item $A \to \cdot$.

→ The central idea in the SLR method is first to construct from the grammar a DFA to recognize viable prefixes.

→ One collection of sets of LR(0) items, which we call the canonical LR(0) collection provides the basis for constructing SLR parsers. To construct the canonical LR(0) collection for a grammar we define an augmented grammar and two functions closure and goto.

→ If G is a grammar with start symbol S then G', the augmented grammar for G is G with new start symbol S' and production $S' \to S$. The purpose of this new starting production is to indicate to

the parser when it should stop parsing and announce acceptance of the input.
That is, acceptance occurs when and only when the parser is about to reduce by $S' \to S$.

→ The closure operation:

→ If I is a set of items for a grammar G, then closure(I) is the set of items constructed from I by the two rules:

1) Initially, every item in I is added to closure(I).

2) If $A \to \alpha . B\beta$ is in closure(I) and $B \to \gamma$ is production, then add the item $B \to .\gamma$ to I, if it is not already there. We apply this rule until no more new items can be added to closure(I).

→ $A \to \alpha . B\beta$ in closure(I) indicates that at some point in the parsing process we might next see a substring derivable from γ at this point. For this reason we also include $B \to .\gamma$ in closure(I)

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

Date: / /
Page No.

Consider the augmented expression grammar:

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

If I is the set of one item

$I: \{ E' \to \cdot E \}$ then closure(I) contains the items.

$E' \to \cdot E$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$

Here $E' \to \cdot E$ is put in closure(I) by rule (1) $\overline{T}$ there is an E immediately to the right of a dot, by rule (2) we add E-productions with dots at the left end. that is,

$E \to \cdot E + T$
$E \to \cdot T$ by rule(2)

$T \to \cdot T * F$
$T \to \cdot F$ by rule(2)

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

it turns out that we can divide all the sets of items we are interested in into two classes of items.

1. Kernel items: Which include the initial item $S' \to .S$ and all items whose dots are not at the left end.

2. Nonkernel items: Which have their dots at the left end.

The Goto operation:

The second useful function is goto(I, X) where I is a set of items and X is a grammar symbol. goto(I, X) is defined to be the closure of the set of all items $[A \to \alpha X . \beta]$ such that $[A \to \alpha . X \beta]$ is in I. Intuitively, if I is the set of items that are valid for some viable prefix γ, then goto(I, X) is the set of items that are valid for viable prefix γX.

Ex: If I is the set of two items

$\{ E' \to E . , \quad E \to E . + T \}$

then goto(I, +) consist of

$E \to E + . T$
$T \to . T * F$
$T \to . F$
$F \to . (E)$
$F \to . id.$

Date: / /
Page No.

We Computed goto(I, +) by examining I for items with + Immediately to the right of the dot. $E' \to E.$ is not such an Item but $E \to E.+T$ is. We moved the dot over the + to get $\{E \to E+.T\}$ and then took the closure of this set.

Simple LR: example:

ex: SLR 2.

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id.$

1 $E \to E+T$
2 $E \to T$
3 $T \to T*F$
4 $T \to F$
5 $F \to (E)$
6 $F \to id$

Augmented grammar:

$E' \to E$
$E \to E+T$
$E \to T$
$T \to T*F$
$T \to F$
$F \to (E)$
$F \to id.$

Canonical Set of LR(0) Item:

I0: closure($\{E' \to .E\}$)

$E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id.$

I_1 : goto (I_0 , E) = { $E' \to E.$ //
$E \to E. + T$ }

I_2 : goto (I_0 , T) = { $E \to T.$ //
$T \to T. * F$ }

I_3 : goto (I_0 , F) = { $T \to F.$ } //

I_4 : goto (I_0 , ( ) = { $F \to (. E.)$
Closure { $E \to .E + T$
$E \to .T$
$T \to .T * F$
$T \to .F$
$F \to .(E)$
$F \to .id$ }

I_5 : goto (I_0 , id) = { $F \to id.$ } //

I_6 : goto (I_1 , +) = { $E \to E + .T$
$T \to .T * F$
$T \to .F$
$F \to .(E)$
$F \to .id$ }

I_7 : goto (I_2 , *) = { $T \to T * .F$
$F \to .(E)$
$F \to .id$ }

I_8 : goto (I_4 , E) = { $F \to (E.)$
$E \to E. + T$ }

I_2 : goto (I_4 , T) = { E → T. = I_2
T → T. * F }

I_4 goto (I_4 , F) = { F → (.E)
E → .E+T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

I_3 : goto (I_4 , F) = { T → F. }

I_4 : goto (I_4 , ( ) = { F → (.E)
E → .E+T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

I_5 : goto (I_4 , id) = { F → id. }

I_9 : goto (I_6 , T) = { E → E+T. //
T → T.*F }

I_3 goto (I_6 , F) = { T → F. }

I_4 goto (I_6 , ( ) = { F → .(E)
E → .E+T
T → .T*F
T → .F
F → .(E)
F → .id }

I_5 goto (I_2, ( ) : { $F \to id.$ }

Date: / /
Page No.

I_{10} : goto (I_7, F) = { $T \to T * F.$ }

I_2 goto (I_7, ( ) =

I_5 goto (I_7, id)

I_{11} : goto (I_8,)) = { $F \to (E).$ } //

I_6 : goto (I_1, +)

I_7 goto (I_9, *)

SLR parsing table:

follow (E') = { $ }
follow (E) = { $, +,) }
follow (T) = { *, $, +,) }
follow (F) = { *, $, +,) }

	Action						goto		
	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				Accept			
2		r_2	S7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S5			S4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		r_1	S7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Parse string id + id * id.

Stack	Input	Action
$0	id+id*id$	S5 (Shift)
$0 id 5	+id*id$	reduce (F→id)
$0 F 3	+id*id$	reduce (T→F)
$0 T 2	+id*id$	reduce (E→T)
$0 E 1	+id*id$	S6 (Shift)
$0 E 1 + 6	id*id$	S5 (Shift)
$0 E 1+6 id 5	*id$	reduce (F→id)
$0 E 1+6 F 3	*id$	reduce (T→F)
$0 E 1+6 T 9	*id$	S7 (Shift)
$0 E 1+6 T 9 * 7	id$	S5 (Shift)
$0 E 1+6 T 9 * 7 id 5	$	reduce (T→F)
$0 E 1+6 T 9 * F 10	$	reduce (T→T*F)
$0 E 1+6 T 9	$	reduce (E→E+T)
$0 E 1	$	Accept.

Date: / /
Page No.

The codes for actions are:

1. Sj means Shift and stack state j
2. rj means reduce by the production numbered j.
3. blank means error.
4. Accept.

→ The parsing actions for state i are determined as follows:

1(a) If $A \to \alpha . a\beta$ is in I_i and goto $(I_i, a) = I_j$ then set Action[i, a] to "Shift j" Here a must be terminal.

→ 1(b) The goto transitions for state i are constructed for all nonterminals A using the rule: If goto $(I_i, A) = I_j$ then goto [i, A] = j.

2. if $[A \to \alpha .]$ is in I_i, then set Action [i, a] to "reduce $A \to \alpha$" for all a in follow (A), here A may not be S'.

3. If $[S' \to S .]$ is in I_i, then set Action [i, $] to "accept".

Q-B 13 SLR

Show that the following grammar

$S \to AaAb \mid BbBa$

$A \to \varepsilon$

$B \to \varepsilon$

1. $S \to AaAb$
2. $S \to BbBa$
3. $A \to \varepsilon$
4. $B \to \varepsilon$

Augmented grammer:

$S' \to S$

$S \to AaAb \mid BbBa$

$A \to \varepsilon$

$B \to \varepsilon$

Canonical set of LR(0) Item:

I_0: closure ({ $S' \to \cdot S$ })

$S' \to \cdot S$

$S \to \cdot AaAb$

$S \to \cdot BbBa$

$A \to \cdot$ //

$B \to \cdot$ //

I_1: goto (I_0, S) = { $S' \to S \cdot$ } //

I_2: goto (I_0, A) = { $S \to A \cdot aAb$ }

I_3: goto (I_0, B) = { $S \to B \cdot bBa$ }

I_4: goto (I_2, a) = { $S \to Aa \cdot Ab$
$A \to \cdot$ }

I_5: goto (I_3, b) = { $S \to Bb \cdot Ba$
$B \to \cdot$ }

I_6: goto(I_3, A) = { $S \to \cdot AaA \cdot b$ }

I_7: goto(I_5, B) = { $S \to BbB \cdot ba$ }

I_8: goto(I_6, b) = { $S \to AaAb \cdot$ }

I_9: goto(I_7, a) = { $S \to BbBa \cdot$ }

SLR: The parsing table:

State	Action			goto		
	a	b	$	S	A	B
0	r_3/r_4	r_3/r_4		1	2	3
1			Accept			
2	S_4					
3		S_5				
4	r_3	r_3			6	7
5	r_4	r_4				
6		S_8				
7	S_9					
8			r_1			
9			r_2			

follow(S') = { $ }

follow(S) = { $ }

follow(A) = { a, b }

follow(B) = { b, b }

As Conflicting entries appear in above table. This grammar is not SLR(1).

$E \to E+T \mid T$
$T \to TF \mid F$
$F \to F* \mid 9 \mid b$

Date: / /
Page No.

Q. B

16 $S \to Aa \mid bAc \mid dc \mid bda$

SLR $A \to d.$

→

1 $S \to Aa$
2 $S \to bAc$
3 $S \to dc$
4 $S \to bda$
5 $A \to d$

follow(S') : {$}
follow(S) : {$}
follow(A) : {a, c}

Augmented grammar:

$S' \to .S$
$S \to Aa$
$S \to bAc$
$S \to dc$
$S \to bda$
$A \to d.$

Canonical set of (RCO) Item:

I_0 : closure ({$S' \to .S$})

$S' \to .S$
$S \to .Aa$
$S \to .bAc$
$S \to .dc$
$S \to .bda$
$A \to .d$

I_1 : goto (I_0, S) : {$S' \to S.$} //

I_2 : goto (I_0, A) : {$S \to A.a$}

I_3: goto(I_0, b) : { $S \to b \cdot Ac$
$S \to b \cdot da$
$A \to \cdot d$ }

I_4: goto(I_0, d) : { $S \to d \cdot c$
$A \to d \cdot$ } //

I_5: goto(I_2, a) : { $S \to Aa \cdot$ } //

I_6: goto(I_3, A) : { $S \to bA \cdot c$ }

I_7: goto(I_3, d) : { $S \to bd \cdot a$
$A \to d \cdot$ } //

I_8: goto(I_4, c) : { $S \to dc \cdot$ } //

I_9 goto(I_6, c) : { $S \to bAc \cdot$ } //

I_{10} goto(I_7, a) : { $S \to bda \cdot$ } //

SLR Parsing table :

	a	b	c	d	$	S	A
0		S_3		S_4		1	
1					accept		
2	S_5						
3				S_7			6
4	R_5		R_5/S_8				
5					R_1		
6			S_9				
7	R_5/S_{10}		R_5				
8					R_3		
9					R_2		
10					R_4		

grammar is not SLR.

Two types of Conflicts in LR parsing table (SLR, CLR, LALR, LR(0))

SR → Shift reduce
RR → reduce reduce.

Date: / /
Page No.

Canonical (LR(1)) Parser :

→ The Canonical Set of items is the Passing technique in which a lookahead Symbol is generated while Constructing set of items. Hence the Collection of set of items is refered as LR(1). The value 1 in the bracket indicates that there is one lookahead Symbol in the set of items.

We follow the same steps as in SLR passing technique and those are:

1 Construction of Canonical set of items along with the lookahead.

2 Building Canonical LR parsing table.

3 parsing the Input String using canonical LR Passing table.

LR(1) item : LR(0) item + lookahead.

Construction of Canonical set of items along with the lookahead:

1 for the grammar G initially add $S' \rightarrow \cdot S$ in the set of item C.

2 for each Set of items I_i and c and for each grammar symbol X (may be terminal or non terminal) add closure (I_i, X).

This process should be repeated by applying goto (Ii, X) for each X in Ii such that goto (Ii, X) is not empty and not in C. The set of items has to constructed until no more set of items can be added to C.

3 The closure function can be computed as follows:
for each item $A \to \alpha . X\beta$, a and rule $X \to y$ and $b \in$ first (β, a).
such that $X \to .y$ and b is not in I then add $X \to .y$, b to I.

4 Similarly the goto function can be computed as: for each item $[A \to \alpha . X\beta, a]$ is in I and rule $[A \to \alpha X . \beta, a]$ is not in goto items then add $[A \to \alpha X . \beta, a]$ to goto items.

This process is repeated until no more set of items can be added to the collection C.

ex: $S \to CC$.
$C \to aC \mid d$

CLR

Construct (RCI) Set of items for the above grammar.

→ augmented grammar:

$S' \to S$
$S \to CC$
$C \to aC \mid d$

Initially add $S' \to \cdot S, \$$ as the first rule in I_0.

We begin by computing the closure of $\{S' \to \cdot S, \$\}$

We match the item $[S' \to \cdot S, \$]$ with the item $[A \to \alpha \cdot X\beta, a]$ in the Procedure Closure.

→ $A = S'$, $\alpha = \varepsilon$, $X = S$, $\beta = \varepsilon$, $a = \$$

If there is a production $X \to Y, b$
then add $X \to \cdot Y, b$

here, $S \to \cdot CC$, $b \in first(\beta a)$
$b \in first(\varepsilon \$) = \$$
$b \in first(\$)$
$b = \$$

$S \to \cdot CC, \$$ will be added in I_0.

Now match the item $S \to \cdot CC, \$$
with the item $[A \to \alpha \cdot X\beta, a]$

$A = S, \alpha = \varepsilon, X = C, \beta = C, a = \$$

if there is production $X \to Y$ then add
$X \to \cdot Y, b$

$C \to \cdot aC,$ $b \in first(\beta a)$
$C \to \cdot d,$ $b \in first(C\$)$
$\to first(C)$
$\to \{a, d\}$
$\to b = \{a, d\}$

$C \to \cdot aC, a/d$ } will be added in
$C \to \cdot d, a/d$ } I_0.

I_0: $S' \to \cdot S, \$$
$S \to \cdot CC, \$$
$C \to \cdot aC, a/d$
$C \to \cdot d, a/d$

I_1 : goto (I_0, S) : $\{ S' \to S\cdot, \$ \}$

I_2 : goto (I_0, C) : $\{ S \to C \cdot C, \$$ (first(βa))
$C \to \cdot aC, \$$ first($\$$) : $\$$
$C \to \cdot d, \$$

I_3 : goto (I_0, a) : $C \to a \cdot C, a/d$ (first(βa))
$C \to \cdot aC, a/d$ first(a) or
$C \to \cdot d, a/d$ first(d) =
(a/d)

I_4 : goto (I_0, d) : { C → d·, a/d }

I_5 : goto (I_2, C) : { S → CC·, $ }

I_6 : goto (I_2, a) : { C → a·C, $ → first(βa)
C → ·aC, $: first($)
C → ·d, $ } = $

I_7 : goto (I_2, d) = { C → d·, $ }

I_8 : goto (I_3, C) = { C → aC·, a/d }

I_3 : goto (I_3, a) = { C → a·C, a/d
C → ·aC, a/d
C → ·d, a/d }

I_4 : goto (I_3, d) = { C → d·, a/d }

I_9 : goto (I_6, C) = { C → aC·, $ }

I_6 : goto (I_6, a) = { C → a·C, $
C → ·aC, $
C → ·d, $ }

I_7 : goto (I_6, d) = { C → d·, $ }

Date: / /
Page No.

Construction of Canonical LR parsing table:

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'.

2. 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 create an entry in the action table $[I_i, a]$ = shift j.

b) If there is a production $[A \to \alpha., a]$ in I_i then in the action table:
action $[I_i, a]$ = reduce by $A \to \alpha$. Here A should not be S'.

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

3. The goto part of the LR table can be filled as: The goto transitions for state i is considered for nonterminals only. If goto$(I_i, A) = I_j$ then goto$[I_i, A] = j$.

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

CLR Parsing table

	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	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

CLR

Q-B

Q1 Construct Canonical Parsing table for the following grammar.

→ $S' \to S$

$S \to CC$

$C \to cC \mid d$

→ assuming $a = c$.

Q2. Check whether the following grammar is CLR or, not?

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

$A \to d$

$B \to d$

Date: / /
Page No.

Augmented grammar:

$S' \to S$

1. $S \to Aa$
2. $S \to bAC$
3. $S \to BC$
4. $S \to bBa$
5. $A \to d$
6. $B \to d$

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

Closure $\{I_0\}$

$\{ S' \to .S, \$$

$S \to .Aa, \$$

$S \to .bAc, \$$

$S \to .Bc, \$$

$S \to .bBa, \$$

$A \to .d, a$

$B \to .d, c$

for each item $A \to \alpha . X \beta, a$

first(βa)

first($\$$) = $\$$

$\to$ first(βa) = first(a) $\div a$

$\to$ first(βa) = first(c) $= c$

I_1: goto(I_0, S) : $\{ S' \to S., \$ \}$ //

I_2: goto(I_0, A) = $\{ S \to A.a, \$ \}$

I_3: goto(I_0, b) = $\{ S \to b.Ac, \$$ $\to$ first(c) = c

$A \to .d, c$

$S \to b.Ba, \$$ $\to$ first(a) = a

$B \to .d, a$

I_4: goto(I_0, B) = $\{ S \to B.c, \$ \}$

I_5 : goto(I_0, d) : $\{ A \to d., a$; $B \to d., c\$$ // $\}$

I_6: goto (I_2, a) : { $S \to Aa.$, \$ } //

I_7: goto (I_3, A) = { $S \to bA.c$, \$ }

I_9: goto (I_3, d) : { $A \to d.$, c // $= I_4$
$B \to d.$, a } //

I_8: goto (I_3, B) = { $S \to bB.a$, \$ } $= I_{12}$

I_{10}: goto (I_4, c) : { $S \to Bc.$, \$ } //

I_{11}: goto (I_7, c) : { $S \to bAc.$, \$ } //

I_{12}: goto (I_8, a) : { $S \to bBa.$, \$ } //

CLR(1) parsing table :

	a	b	c	d	\$	S	A	B
0		S_3		S_5		1	2	4
1					Accept			
2	S_6							
3				S_9			7	8
4			S_{10}					
5	R_5		R_6					
6					R_1			
7			S_{11}					
8	S_{12}							
9	R_6		R_5					
10					R_3			
11					R_2			
12					R_4			

Grammar is CLR(1).

I_3 : goto (I_2, A)

{

$S \to cA.$ //

I_4 : goto (I_2, C)

{

$S \to cc.B$

$A \to c.A$

$B \to .ccB$

$B \to .b$

$A \to .cA$

$A \to .a$

I_5 : goto (I_2, a)

{

$A \to a.$ //

I_6 : goto (I_4, B)

{

$S \to ccB.$ //

}

I_7 : goto (I_4, A)

{

$A \to cA.$ //

}

I_8 : goto (I_4, c)

{

$B \to c.cB$

$A \to c.A$

$A \to .cA$

$A \to .a$

I_9 : goto (I_4, b)

{

$B \to b.$ //

}

I_{10} : goto (I_4, a)

{

$A \to a.$

}

I_{10} : goto (I_8, c)

$B \to cc.B$

$A \to c.A$

$B \to .ccB$

$B \to .b$

$A \to .cA$

$A \to .a$

~~I_{11} goto (I_{10},~~

I_5 : goto (I_8, a)

{

$A \to a.$ //

}

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

{

$B \to ccB.$

}

(I_7) : goto (I_{10}, A)

{

$A \to cA.$

}

(I4) : goto (I_{10}, c)

$B \to c \cdot CB$

$A \to c \cdot A$

$A \to \cdot cA$

$A \to \cdot a$

(I9) : goto (I_{10}, b)

$B \to b \cdot$

(I5) . goto (I_{10} , a)

$A \to a \cdot$

Parsing table :

	a	b	c	$	S	A	B
0			S_2		1		
1							
2	S_5		S_4			3	
3							
4	S_5	S_9	S_8			7	6
5				r_4			
6				r_2			
7				r_3			
8	S_5		S_{10}			7	
9				r_6			
10	S_5	S_9	S_8			7	11
11				r_5			

Parse the input string "ccb"

Stack	i/p buffer	Action
$ 0	ccb $	Shift
$ 0c2	cb $	Shift
$ 0c2c4	b $	Shift
$ 0c2c4b9	$	reduce B→b
$ 0c2c4B6	$	reduce S→ccB
$ 0 S1	$	accept

Date: / /
Page No.

LALR Parser:

→ In this type of parser the lookahead symbol Symbol is generated for each set of item The table obtained by this method are ~~Similar~~ smaller in size than LR(k) parsers In fact the states of SLR and LALR parsing are always same. Most of the programing languages use LALR Parsers.

We follow same steps as in SLR and Canonical LR parsing and those are:

1. Construction of Canonical set of items along with lookahead
2. Building LALR parsing table.
3. Parsing the input string using Canonical LR Parsing table.

Construction Set of LR(1) Items along With the look ahead.

→ The Construction of LR(1) item is same as in CLR. parser. But the only difference is that: in Construction of LR(1) items for LR parser. we have ~~the~~ differed the two states if the second Component is different but in this case we will merge the two States by merging of first and second Components from both the states.

for example
we have got I_3 and I_6 because of different second components but for LALR parser we will consider these two states as same by merging these states i.e.

$I_3 + I_6 : I_{36}$

Hence

I_{36} : goto(I_0, a) (I_2, a)

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

Ex (LALR)

$S \to CC$
$C \to aC$
$C \to d$

Construct set of (R.C.I) items for LALR parser

I_0 :
$S' \to \cdot S$, $
$S \to \cdot CC$, $
$C \to \cdot aC$, a/d
$C \to \cdot d$, a/d

I_1 : goto(I_0, S)
$S' \to S \cdot$, $

I_2: goto (I_0, C)
$S \to C \cdot C, \$$
$C \to \cdot aC, \$$
$C \to \cdot d, \$$

I_{36}: goto (I_0, a) & goto (I_2, a)
$C \to a \cdot C, a/d/\$$
$C \to \cdot aC, a/d/\$$
$C \to \cdot d, a/d/\$$

I_{47}: goto (I_0, d) goto (I_2, d)
$C \to d \cdot, a/d/\$$

I_5: goto (I_2, C)
$S \to CC \cdot, \$$

I_{89}: goto (I_3, C) goto (I_6, C)
$C \to aC \cdot, a/d/\$$

Construction of LALR parsing table:

Step: 1 Construct the (LR(1)) set of items.

Step: 2 Merge the two States I_i and I_j if the first Component (i.e. the production rules with dots) are matching and create new state replacing one of the older state such as $I_{ij}: I_i \cup I_j$

Step: 3 The parsing actions are based on each item If. The actions are as given below.

a) If $[A \to \alpha\beta, b]$ is in I_i and goto $(I_i, a) = I_j$ then create an entry in the action table action $[I, a]$ = Shift j.

b) If there is a production $[A \to \alpha., a]$ in I_i then in the action table action $[I_i, a]$ = reduce by $A \to \alpha$. Here A should not be S'.

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

Step: 4 The goto part of the LR table can be filled as: The goto transitions for state i is considered for non terminals only. If goto $(I_i, A) = I_j$ then goto $[I_i, A] = j$.

Step: 5 If the parsing action conflict then the algorithm fails to produce LALR Parser and grammer is not LALR (1). All the entries not defined by rule 3 and 4 are considered to be "error".

LALR parsing table:

	Action			Goto	
	a	d	$	S	C
0	S36	S47		1	2
1			Accept		
2	S36	S47			5
36	S36	S47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

QB-16. Check that following grammar is LALR or not.

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

Augmented grammar :→

I_0 : $S' \to .S$, $
$S \to .L=R$, $
$S \to .R$, $
$L \to .*R$, =
$L \to .id$, =
$R \to .L$, $
$L \to .*R$, $
$L \to .id$, $

merge.

I_0 : $S' \to .S$, $
$S \to .L=R$, $
$S \to .R$, $
$L \to .*R$, =/$
$L \to .id$, =/$
$R \to .L$, $

I_1 : goto (I_0, S)
$S' \to S.$, $ //

I_2 : goto (I_0, L)
$S \to L.=R$, $
$R \to L.$, $ //

I_3 : goto (I_0, R)
$S \to R.$, $ //

I_4 : goto (I_0, *)
$L \to *.R$, =/$
$R \to .L$, =/$
$L \to .*R$, =/$
$L \to .id$, =/$

I_5 : goto (I_0 id)
$L \to id.$, =/$ //

I_6 : goto (I_2, =)
$S \to L=.R$, $
$R \to .L$, $
$L \to .*R$, $
$L \to .id$, $

I_7 : goto (I_4, R)
$L \to *R.$, =/$ //

I_8 : goto (I_4, L)
$R \to L.$, =/$ //

(I_4) : goto (I_4, *)
$L \to *.R$, =/$
$R \to .L$ =/$
$L \to *R$, =/$
$L \to id$, =/$

I_5 : goto (I_4, id)

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

I_9 : goto (I_6, R)

$S \rightarrow L = R.$, $\$$

I_{10} : goto (I_6, L)

$R \rightarrow L.$, $\$$

I_{11} : goto (I_6, *)

$L \rightarrow *.R$, $\$$

$R \rightarrow .L$, $\$$

$L \rightarrow .*R$, $\$$

$L \rightarrow .id$, $\$$

I_{12} : goto (I_6, id)

$L \rightarrow id.$, $\$$ //

I_{13} : goto (I_{11}, R)

$L \rightarrow *R.$, $\$$ //

I_{10} : goto (I_{11}, L)

$R \rightarrow L.$, $\$$

I_{11} : goto (I_{11}, *)

$L \rightarrow *.R$, $\$$

$R \rightarrow .L$, $\$$

$L \rightarrow .*R$, $\$$

$L \rightarrow .id$, $\$$

I_{12} : goto (I_{11}, id)

$L \rightarrow id.$, $\$$.

CLR parsing table :

	=	*	id	$	S	L	R
0		S_4	S_5		1	2	3
1				Accept			
2	S_6			γ_5			
3				γ_2			
4		S_4	S_5			8	7
5	γ_4			γ_4			
6		11	12			10	9
7	γ_3			γ_3			
8	γ_5			γ_5			
9				γ_1			
10				γ_5			
11		S_{11}	S_{12}			10	13
12				γ_4			
13				γ_3			

for LALR :

I_{5} : (I_0, id) (I_6, id)

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

I_{7} : (I_4, R) (I_{11}, R)

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

I_{8} : (I_4, L) (I_6, L)

I_{4} : $(I_0, *)$ $(I_6, *)$

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

$R \to \cdot L, \$$

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

$L \to \cdot id, \$$

LALR Parsing table:

	=	*	id	$	S	L	R
0		S411	S512		1	2	3
1				Accept			
2	S6			r5			
3				r2			
4 11		S411	S512			810	713
512	r4			r4			
6							
713	r3			r3			
810	r5			r5			
9							

Date: / /
Page No.

Comparision of LR parsers:

1 SLR Parser:
1 SLR Parser is Smallest in Size
2 LALR Parser:
2 LALR and SLR have the same size
3 Canonical LR parser:
3 LR Parser is largest in size.

1 It is an easiest method based on follow function.
2 This method is appliable to wider class than SLR
3 This method is most powerful than SLR and LALR.

1 This method exposes less Syntactic features than that of LR parsers.
2 Most of the syntactic features of a language are exposed in LALR.

3 This method exposes less Syntactic features than that of LR parsers.

1 Error detection is not immediate in SLR.
2 Error detection is not immediate in LALR
3 Immediate error detection is done by LR parsers.

1. It requires less time and space complexity.

2. The time and space complexity is more in LALR but efficient methods exist for constructing LALR parsers directly.

3. The time and space complexity is more for canonical LR parser.

Q-B Justify the statement " A class of grammar that can be parsed using LR methods is a proper subset of the class of grammars that can be parsed with predictive parser."

→ let us understand the term proper subset. Set A is a proper subset of B if and there exists at least one element in B that is not in A.

→ The languages that can be parsed using predictive parser can also be parsed using LR methods. But there are some languages that can be parsed by LR methods and predictive parsers can not parse them. Hence it is said that the class of grammar that can be parsed using LR methods is a proper subset of the class of grammars that can be passed with predictive parser.