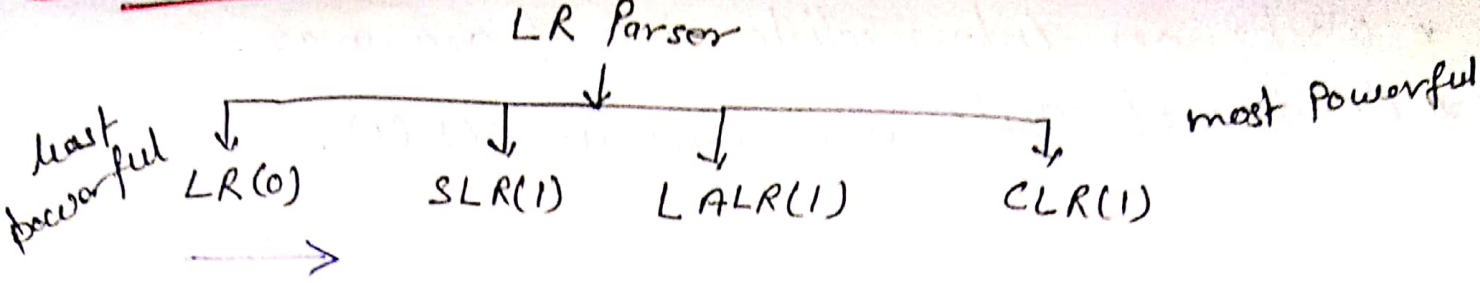
⊛ LR Parsers :-



LR → L- means scanning the string from Left to right.
R- means reverse of right most derivation. (Reverse RMD)

SLR → Simple LR

LALR → Look Ahead LR

CLR → Canonical LR

LR Parser:- Non recursive shift reduce bottom up parser.

- It is also known as LR(K). → look ahead s/b
  - Left to Right scanning i/p string
  - Construction of Right most derivation in reverse.

⊛ SLR(1) :-

- Simple LR parser
- Works on smallest class of grammar.
- Few number of ~~states~~ states.
- Simple & fast Construction

⊛ LR(1) :- LR parser

- Works on Complete set of LR(1) grammar
- Large no. of states.
- Slow Construction.

⊛ LALR(1) :- Look ahead LR parsor.

* Works on intermediate size of grammar.
* Number of states are same as SLR(1).

⊛ Structure of LR Parser :-

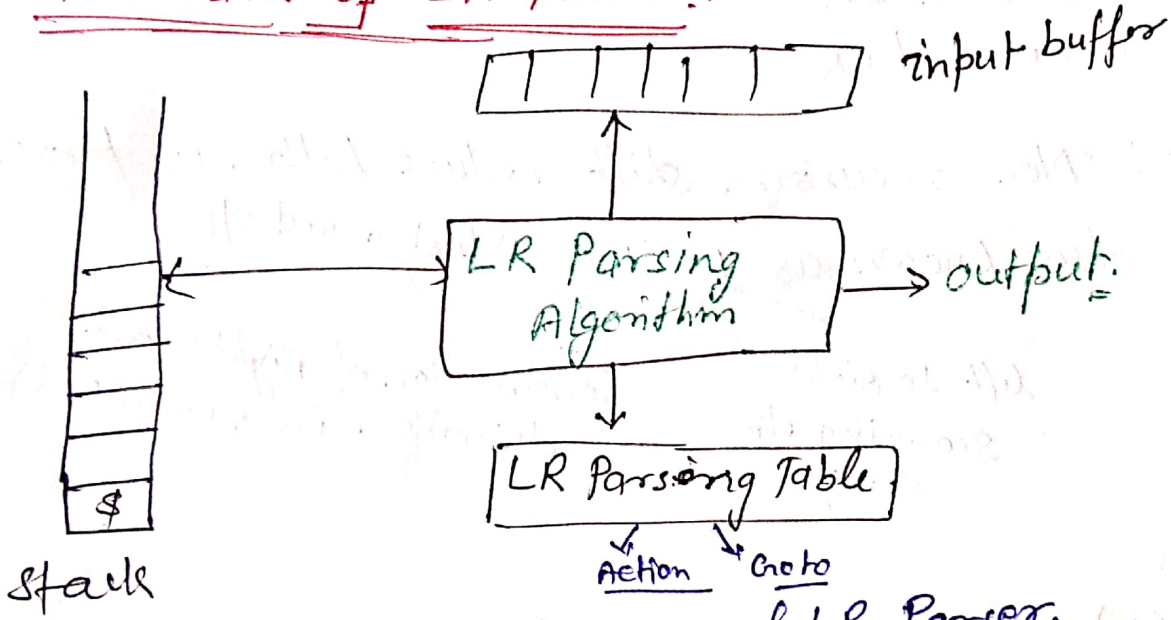


fig :- Block diagram of LR Parser.

⊛ To construct LR(0) & SLR(1) tables we use canonical collection of LR(0) items.

⊛ To construct LALR(1) & CLR(1) tables we use Can. canonical collection of LR(1) items.

⊛ LR(0) Parsing :- / Construction of LR(0) Parser :-

Various steps involved in parsing -

- for the given i/p string write CFG (context free grammar)
- Check ambiguity of the grammar.
- Add augment production in the given grammar
- Create canonical collection of LR(0) items.
- Draw a data flow diagram (DFA)
- Construct LR(0) parsing table.

eg :- $S \to AA$
$A \to aA/b$ } G.

→ Add one more production

$S' \to S$
$S \to aA/b$ } Augment Production
$S \to AA$

Example:-

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

Soln:- the given grammar is CFG & unambiguous.

Step ① the given grammar is CFG.

Step ② " " " is unambiguous.

Step ③ Add Augment production.

$S' \rightarrow S$
$S \rightarrow AA$
$A \rightarrow aA/b$ } given grammar.

Step ④ Create canonical Collection of LR(0) items:-

- An LR(0) item is a production G with dot(.) at some position the right hand side of the production.
- LR(0) items is useful to indicate that how much of the i/p has been scanned up to a given point in the process of parsing.
- An LR(0), we place the reduce node in entire row.

$S' \rightarrow .S$
$S \rightarrow .AA$
$A \rightarrow .aA$
$A \rightarrow .b$

Step ⑤:- Drawing DFA

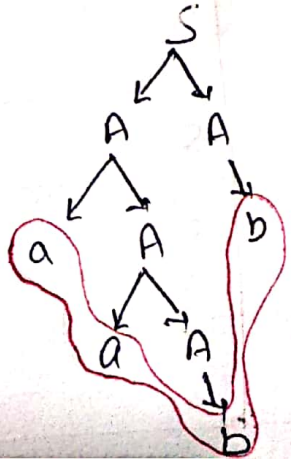
Contain 7 states (I_0 to I_6)

Step ⑥ :- LR(0) Table

- if a state is going to some other state on a terminal it is correspond to a shift move.
- ⊛ if a state is going to some other state on a variable it is correspond to go to move.
- ⊛ if a state contains the final item in the particular row the write the reduce node completed.

Parsing i/p string :- i/p ⇒ aabb$

Steps	Parsing stack	i/p	Action
1.	$0	aabb$	shift 3/shift a3
2.	$0a3	abb$	shift a3
3.	$0a3a3	bb$	shift b4
4.	$0a3a3b4	b$	Reduce r_3 (A→b)
5.	$0a3a3A6	b$	Reduce r_2 (A→aA)
6.	$0a3A6	b$	Reduce r_2 (A→aA)
7.	$0A2	b$	shift b4
8.	$0A2b4	$	Reduce R_3 (A→b)
9.	$0A2A5	$	Reduce R_1 (S→AA)
10.	$0S1	$	Accept



= aabb Ans

⊛ SLR(1) Parsing :-

- Simple LR.
- Works on smallest class of grammar
- few no. of states requires.
- Simple and fast to construct.

- In SLR we place the reduce move only in the fallow of left hand side not to Entire Row.

eg:- G: $A \to (A) / a$

Soln: ① Grammar is CFG.

② " is unambiguous.

③ Add Augumented grammar.

$A' \to A$

$A \to (A) / a$

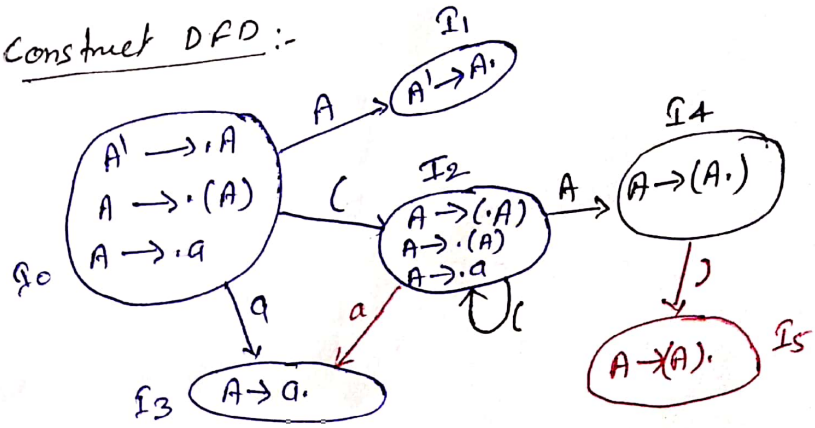
④ Canonical Collection of LR(0) items.

$A' \to \cdot A$

$A \to \cdot (A)$

$A \to \cdot a$

⑤ Construct DFD :-



$A \to \cdot (A)$ — (1)

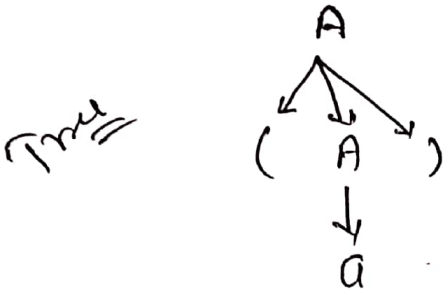
$A \to \cdot a$ — (2)

(6) Parsing Table:-

State	Action (Terminal s/b.)				Goto (Non-terminal symbol)
↓	a	(	)	$	A
I_0	S_3	S_2			1
I_1				Accept	
I_2	S_3	S_2			4
I_3		~~S_2~~	r_2	r_2	
I_4			S_5		
I_5			r_1	r_1	

⑦ Input String Parsing:- (a)$

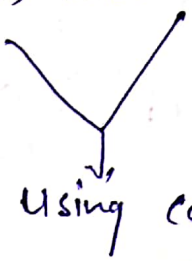
Step	Parsing Stack	Input	Action
1.	$0	(a)$	shift 2 / shift C2
2.	$0(2	a)$	shift_3 / shift 93
3.	$0(2a3	)$	Reduce r3, A→a
4.	$0(2A4	)$	shift_5 / shift 15
5.	$0(2A4)5	$	Reduce r1, A→(A)
6.	$0A1	$	Accept



⇒ (a) Ans

⊛ CLR(1) & LALR(1) Parsing :-

CLR(1) and LALR(1)



Using canonical collection of LR(1) items

⇒ LR(1) items ⟹ LR(0) item + Look ahead

⇒ In CLR & LALR ⟹ reduce is written only on look ahead

⇒ How to find Lookahead items :-

Let me take one grammar & then find the Lookahead

eg: G: $E \rightarrow BB$

$B \rightarrow cB / d$

⇒ Add the Augment grammar & LR(1) items

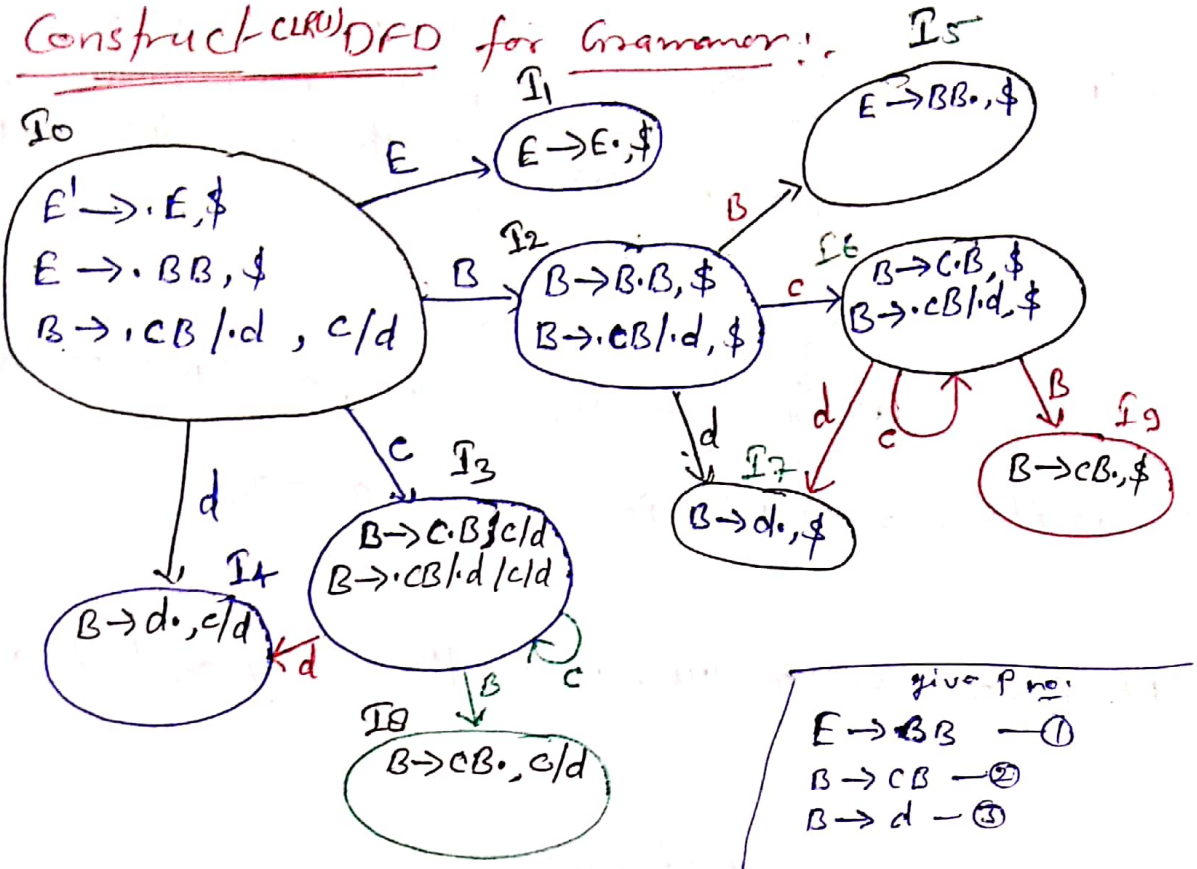
$E' \rightarrow .E$

Now $E' \rightarrow .E, \$$ → Add $ is called lookahead item.

$E \rightarrow .BB, \$$ → Look ahead items.

$B \rightarrow .cB / .d, c/d$ → Look ahead item

✱ Construct CLR(1) DFD for Grammar:-



✱ CLR(1) Parsing Table:-

State	Action (Terminal s/b)			Goto (Non Terminal s/b)	
	c	d	$	E	B
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_6	S_7			5
I_3	S_3	S_4			8
I_4	r_3	r_3			
I_5			r_1		
I_6	S_6	S_7			9
I_7			r_3		
I_8	r_2	r_2			
I_9			r_2		

⑤ LALR(1) :-

CLR(1)
LALR(1) > Both uses the look ahead value

- LALR(1) uses the canonical collection of LR(1) items.

LR(1) items = LR(0) items + Look ahead value

Eg:- G:

$E \rightarrow BB$
$B \rightarrow cB / d$ $\Rightarrow$

$E' \rightarrow .E, \$$
$E \rightarrow .BB, \$$
$B \rightarrow .cB / .d, c/d$

Look ahead value.

} These are the LR(1) items.

$\Rightarrow$ Construct the LAL(1) DFD