* BNF Notations - (Backus Naur form)

→ Notation Technique for CFGs

→ Specify the syntax of the Language.

name ::= expansion

::= → may expand into

Every name in BNF is surrounded by angle brackets < >

<Symbol> ::= exp1 | exp2 | exp3

<Symbol> → Non Terminal

exp1, exp2, exp3 → Sequence of symbols

→ Widely used in Parsing. helps in building the syntax tree of a Program.

→ It provides Concise and readable way

⇒ YACC (Yet Another Compiler Compiler)

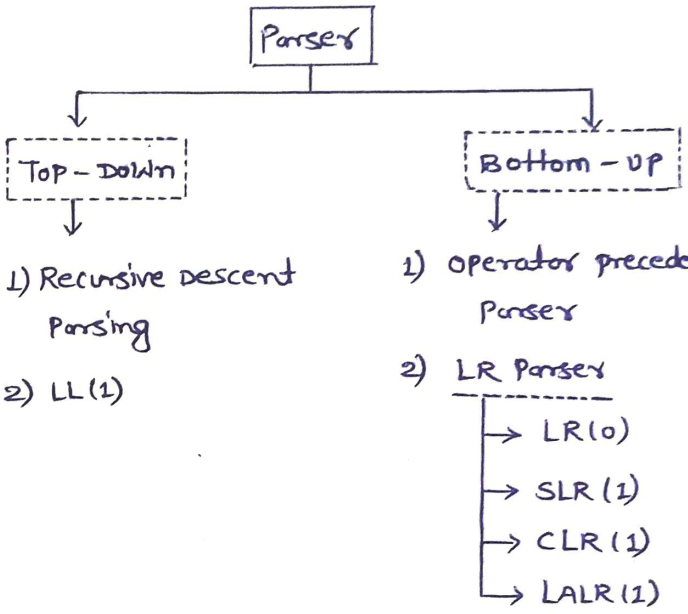
↳ Unix tool that generates ~~LAR~~ LALR Parsers from CFGs (usually in BNF).

* Parser (Syntax Analyzer)

→ main goal is to make sure that the tokens are placed in correct and meaningful order, According to the grammer.

⇓

If the structure is correct, Parser creates a parse Tree / Abstract Syntax tree (AST).



A) Top down Parser

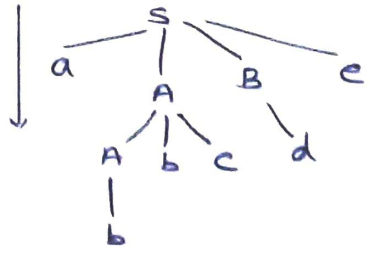
↳ build a parse Tree from the start symbol (root) down to leaves (End symbols)

S → aABe

A → Abc | b

B → d

W = abbcde



(B) Bottom-up Parsing-

→ building a parse tree starting from the leaf nodes (input symbols) and working towards the root node (start symbol).

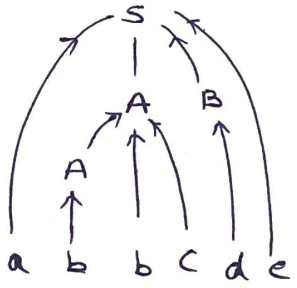
→ Rightmost derivation in Reverse

↳ Parser traces the Right most derivation of the string but work backwards.

→ often called shift-reduce parser.

$$\begin{cases} S \rightarrow aABE \\ A \rightarrow Abc \mid b \\ B \rightarrow d \end{cases}$$

W = abbcde



S → aABe
→ aAde
→ aAbcde
→ abbcde (string match)

(1) Recursive Descent Parser-

- Top-down parser
- Processed input based on recursive functions.

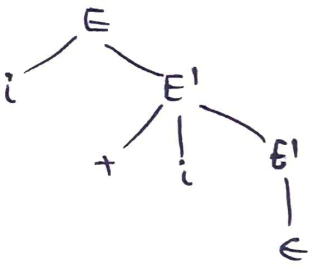
Let Grammer- $\begin{cases} E \rightarrow iE' \\ E' \rightarrow +iE' / \epsilon \end{cases}$

```
main()
{
  E();
if (input == $)
    success
}
```

```
E(){
if (input == i)
    input++
E'()
}
```

```
E'()
{
if (input == +) {
      input++;
      if (input == i) input++
      E'()
}
else return;
```

W = i + i$



- First :-

→ set of terminals that begins in all string derived from A.

1) if $A \rightarrow a\alpha$, $\alpha \in (VUT)^*$

$first(A) = a$

2) if $A \rightarrow \epsilon$

$first(A) = \epsilon$

3) if $A \rightarrow BC$

- $first(A) = first(B)$ (if first(B) doesn't Contains ϵ)
- $first(A) = first(B) \cup first(C)$ (if first(B) Contains ϵ)

Exa- $S \rightarrow AaB / BA$

$A \rightarrow a/b$

$B \rightarrow d/e$

$first(S) = \{a,b,d,e\}$

$first(A) = \{a,b\}$

$first(B) = \{d,e\}$

Exa- $S \rightarrow AaB$

$A \rightarrow b/\epsilon$

$B \rightarrow c$

$first(S) = \{b,a\}$

$first(A) = \{b, \epsilon\}$

$first(B) = \{c\}$

Exa- $S \rightarrow AB$

$A \rightarrow a/\epsilon$

$B \rightarrow b/\epsilon$

$first(S) = \{a,b,\epsilon\}$

$first(A) = \{a, \epsilon\}$

$first(B) = \{b, \epsilon\}$

- follow :-

→ follow (A) gives set of all terminals that follow immediately to the right of A.

1) if S is on start symbol.

$follow(S) = \{\$\}$

2) if $A \rightarrow \alpha B\beta$

follow (B) = first (B) if first (B) doesn't Contains ϵ.

3) $A \rightarrow \alpha B$,

$follow(B) = follow(A)$

Exa- $S \rightarrow aA$

$A \rightarrow bA / \epsilon$

$follow(S) = \{\$\}$

$follow(A) = \{\$\}$

$S \rightarrow AaBb / BbAa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

$follow(S) = \{\$\}$

$follow(A) = \{a\}$

$follow(B) = \{b\}$

Exa- $E \rightarrow E+T / T$

$T \rightarrow T*F / F$

$F \rightarrow (E) / id$

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

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

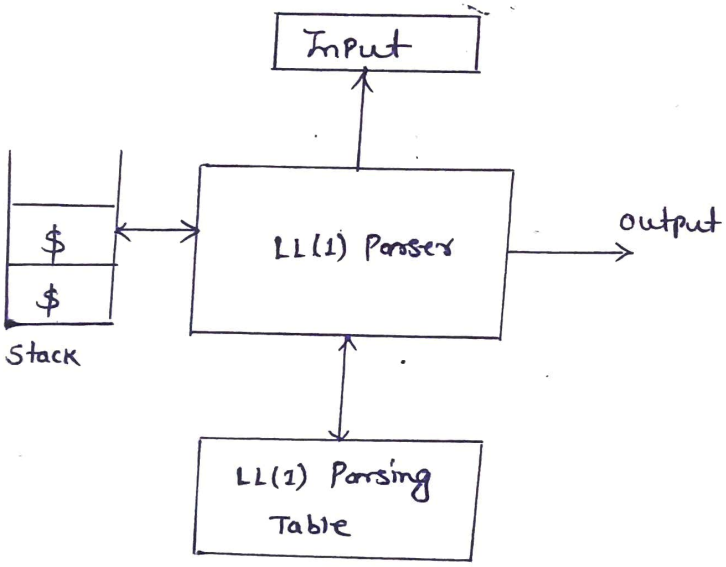
follow (F) = { $, +,), * }

* LL(1) Parser :-

L : Left to Right Scanning

L : Left most derivation

1 : No. of Look ahead symbols.



LL1 Parser Consist of 3 Components-

- Input Buffer
- Parse Stack
- Parse Table

→ Rule to fill Table

1) Add $A \rightarrow \alpha$ under m[A,a]
Where $a \in$ first (α)

2) Add $A \rightarrow \alpha$ under m[A,a]
Where $a \in$ follow (A)
If first (α) Contains ϵ

Example-

	first	follow
$E \rightarrow TE'$	{id, (}	{$,)}
$E' \rightarrow \epsilon / +TE'$	{ϵ, +}	{$,)}
$T \rightarrow FT'$	{id, (}	{+, $,)}
$T' \rightarrow \epsilon / *FT'$	{ϵ, *}	{+, $,)}
$F \rightarrow id / (E)$	{id, (}	{*, +, $,)}

No. of Non Terminals- 5

No. of Terminals = 5

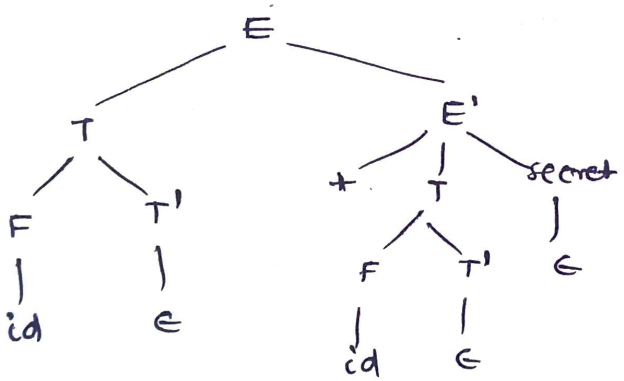
matrix size - 5 * (5+1)

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' = *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Let input str = "id + id"

Stack	input	Production
$ E	id + id $	E → TE'
$ E'T	id + id $	T → FT'
$ E'T'F	id + id $	F → id
$ E'T'id	id + id $	Pop
	Same	
$ E'T'	+ id $	T' → ∈
$ E'	+ id $	E' → +TE'
$ E'T+	+ id $	Pop
$ E'T	id $	T → FT'
$ E'T'F	id $	F → id
$ E'T'id	id $	Pop
$ E'T'	$	T' → ∈
$ E'	$	E' → ∈
$	$	Accept

Parse Tree



Algorithm-

Let x is a grammer symbol (stock symbol) and a is the look ahead symbol.

if x == a == $, then Parsing is success ful.

if x == a != $, then Pop and increment the i/p pointer.

if x != a != $ and m[x, a] Contains the production x → abc, then replace x by abc in reverse order and Continue process.

* Grammer for which LL(1) Parser Can be Constructed is Known as LL(1) Grammers.

if LL(1) Parser Table Contains multiple entries in the Same cell, then the Grammer is Ambiguous and We Can not solve Ambiguous Grammers with LL(1)

Exa-

$S \rightarrow aBDh$

$B \rightarrow cC$

$C \rightarrow bC / \epsilon$

$D \rightarrow EF$

$E \rightarrow g / \epsilon$

$F \rightarrow f / \epsilon$

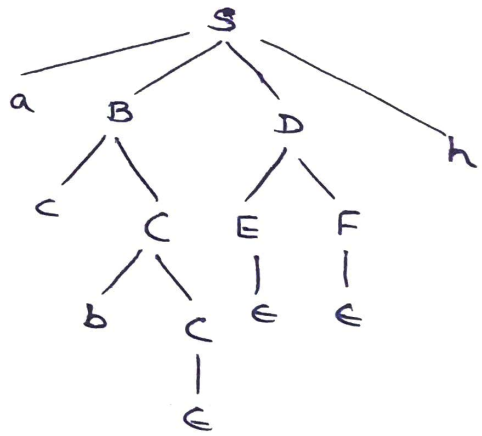
first(S) = {a}
first(B) = {c}
first(C) = {b, ∈}
first(D) = {g, f, ∈}
first(E) = {g, ∈}
first(F) = {f, ∈}

follow(S) = {$}
follow(B) = {g, f, h}
follow(C) = {g, f, h}
follow(D) = {h}
follow(E) = {f, h}
follow(F) = {h}

	a	b	c	f	g	h	$
S	S→aBDh						
B			B→cC				
C		C→bC		C→∈	C→∈	C→∈	
D				D→EF	D→EF	D→∈F	
E				E→∈	E→g	E→∈	
F				F→f		F→∈	

Let input = acbh

Stack	input	Production
$S	acbh$	S→aBDh
$hDBa	acbh$	Pop
$hDB	cbh$	B→cC
$hDCc	cbh$	Pop
$hDC	bh$	C→bC
$hDCb	bh$	Pop
$hDC	h$	C→∈
$hD	h$	D→EF
$hFE	h$	E→∈
$hF	h$	F→∈
$h	h$	Pop
$	$	Accept



* Bottom - up Parser -

→ closure (I) :- 1) Add LR(0) item I to closure (I)

2) if $A \longrightarrow \alpha \cdot B\beta$ is LR(0) item and $B \rightarrow \gamma$ is in G then -

Add $B \rightarrow \cdot\gamma$ to closure (I) Also.

3) Repeat Above 2 steps for Every newly Added LR(0) item.

Exa- $S \longrightarrow AA$

$A \longrightarrow aA / b$

Closure $(S \longrightarrow \cdot AA) \Rightarrow$

$S \longrightarrow \cdot AA$

$A \longrightarrow \cdot aA$

$A \longrightarrow \cdot b$

→ Goto (I, X) :- 1) Add LR(0) item I by moving dot after X.

2) Apply closure to the result obtained in step 1.

Exa- Goto $(S \longrightarrow \cdot AA, A) \Rightarrow$

$S \longrightarrow A \cdot A$

$A \longrightarrow \cdot aA$

$A \longrightarrow \cdot b$

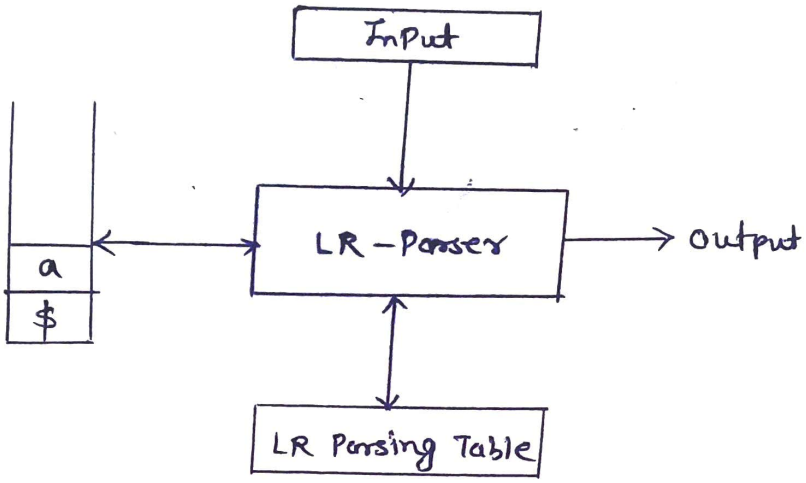
Goto $(S \longrightarrow A \cdot A, A) \Rightarrow S \longrightarrow AA \cdot$

• Handle - Sub string of Input string that matches with RHS of any Production, is called as Handle.

The process of finding the handle & Replacing that handle by it's LHS Variable is called Handle Pruning.

→ Bottom-up Parser is also known as shift Reduce Parser.

→ Bottom up Parser is faster than Top down Parser.



Components

1) I/p buffer
2) Stack
3) Table

→ operation in shift Reduce Parser-

1) shift :- shift operation can be used when handle does not occurs from the top most symbol of the stack. using shift operation, will moving a look ahead symbol in stack.

2) Reduce - performs when handle occurs from the topmost symbol from the stack.

3) Accept - After scanning the complete I/p string, if the stack Contains only the start symbol of the grammer as top most symbol, then the i/p string is Accepted and Parsing is successful.

4) Error - After the Complete I/p string, if the stack Contains any symbol which is different from start symbol as a topmost symbol, then parsing is unsuccessful and hence Error.

→ Algorithm -

1) find Augmented Grammer
2) I_0 = Closure (Augmented LR(0) Item)
3) Apply Closure and Goto function and find all Collections of LR(0) item using DFA
4) Reduce DFA = LR(0) Parsing table.

Ex4 -

$S \rightarrow AA$

$A \rightarrow aA / b$

$\Rightarrow$

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA / b$

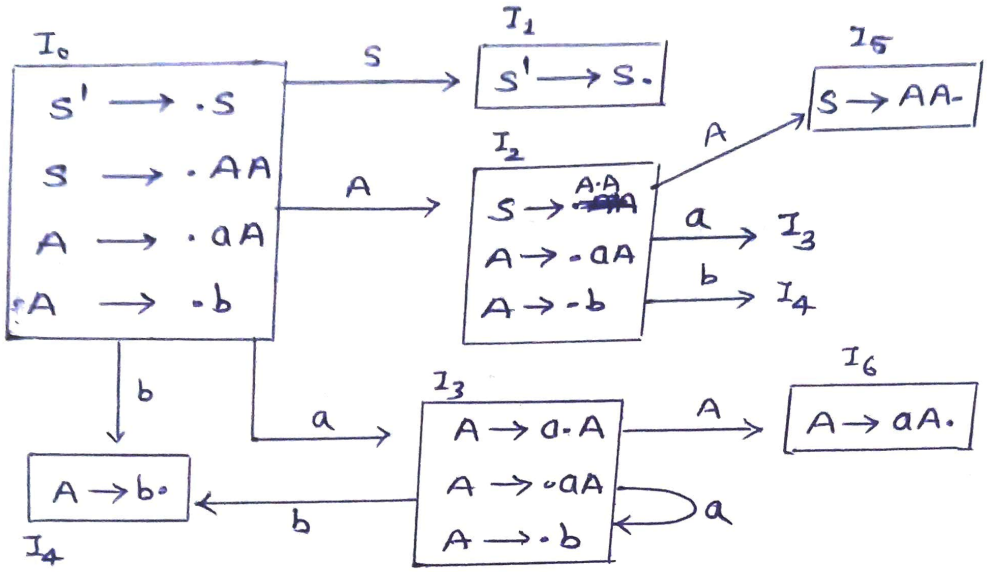


Table-

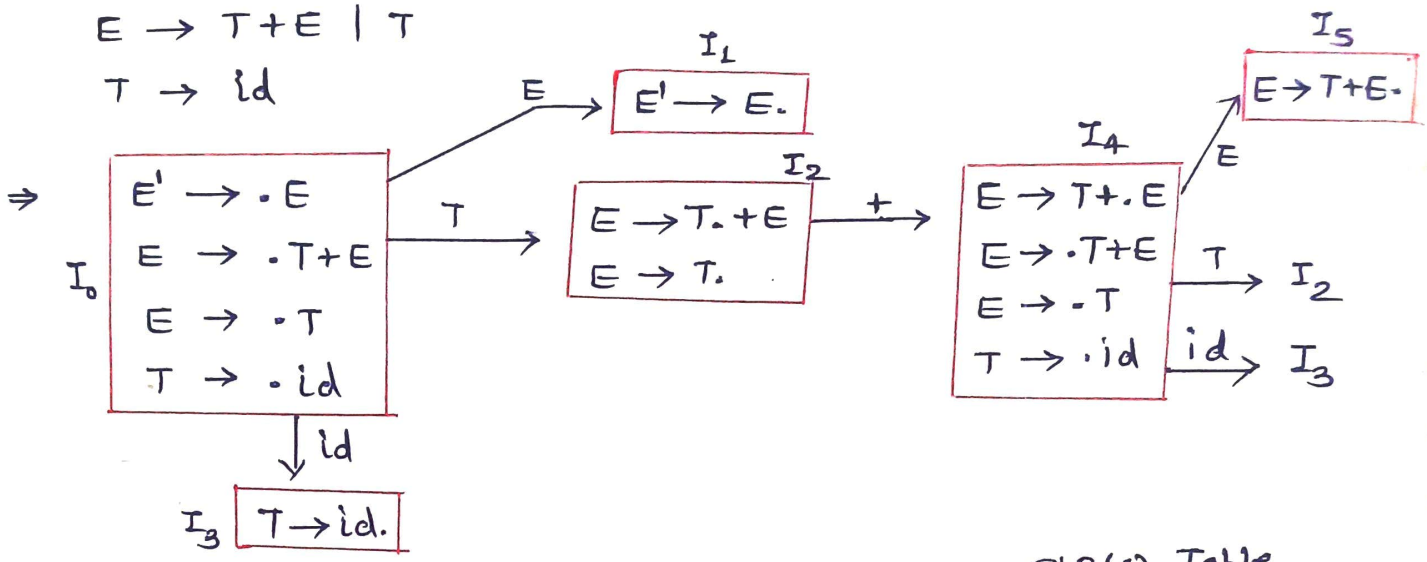
1) $S \rightarrow AA$
2) $A \rightarrow aA$
3) $A \rightarrow b$

	Action			Goto	
	a	b	$	S	A
0	S_3	S_4		1	2
1			Accept		
2	S_3	S_4			5
3	S_3	S_4			6
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

* SLR Parser :- SLR(1)

$E \to T+E \mid T$

$T \to id$



	Action			GoTo	
	id	+	$	E	T
0	S_3			1	2
1			Accept		
2	r_2	S_4/r_2	r_2		
3	r_3	r_3	r_3		
4	S_3			5	2
5	r_1	r_1	r_1		

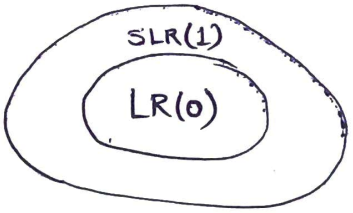
LR(0) Table

Here S-R Conflict Exists, so Grammer is not LR(0) Grammer

SLR(1) Table

	Action			Goto	
	id	+	$	E	T
0	S_3			1	2
1			Accept		
2		S_4	r_2		
3		r_3	r_3		
4	S_3			5	2
5			r_1		

In SLR :-
final item → A → B.
then fill reduce entry in follow(A)



States that have S-R Conflict or R-R Conflict are called Inadequate states.

for I_2 : $E \to T.$

follow(E) = $

for I_3 : $T \to id.$

follow(T) = {+, $}

Let Expr = id + id $

0	id	3	T	2	+	4	id	3	T	2

E	5	E	1

* CLR (1) Parser – (Canonical LR Parser)

→ It uses LookAhead Symbols.

LR(1) items = LR(0) items + LookAhead

if Production is $\begin{cases} A \to \alpha . B C , a \\ B \to . D \end{cases}$, then LookAhead of $B \to . D$ is find by prev. production

$A \to \alpha . B C, a$

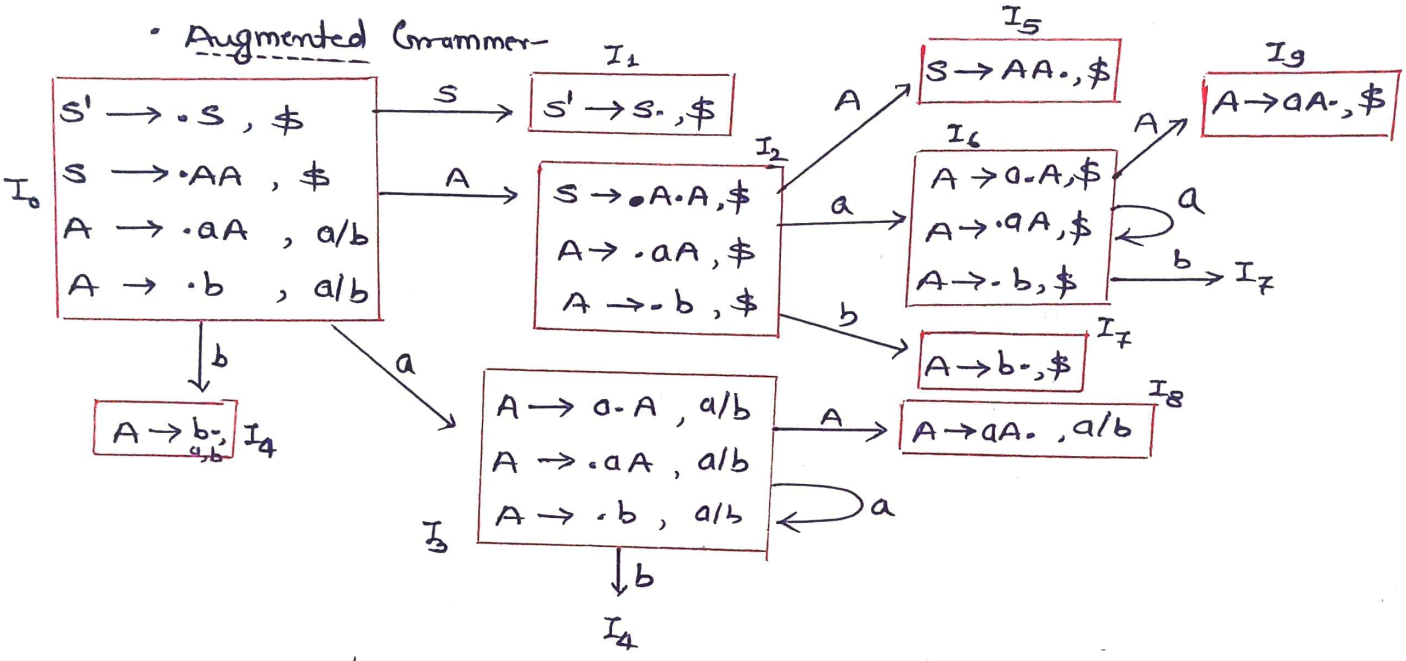
first (C, a)

Exa –

$S \to AA$

$A \to aA / b$

• Augmented Grammer –

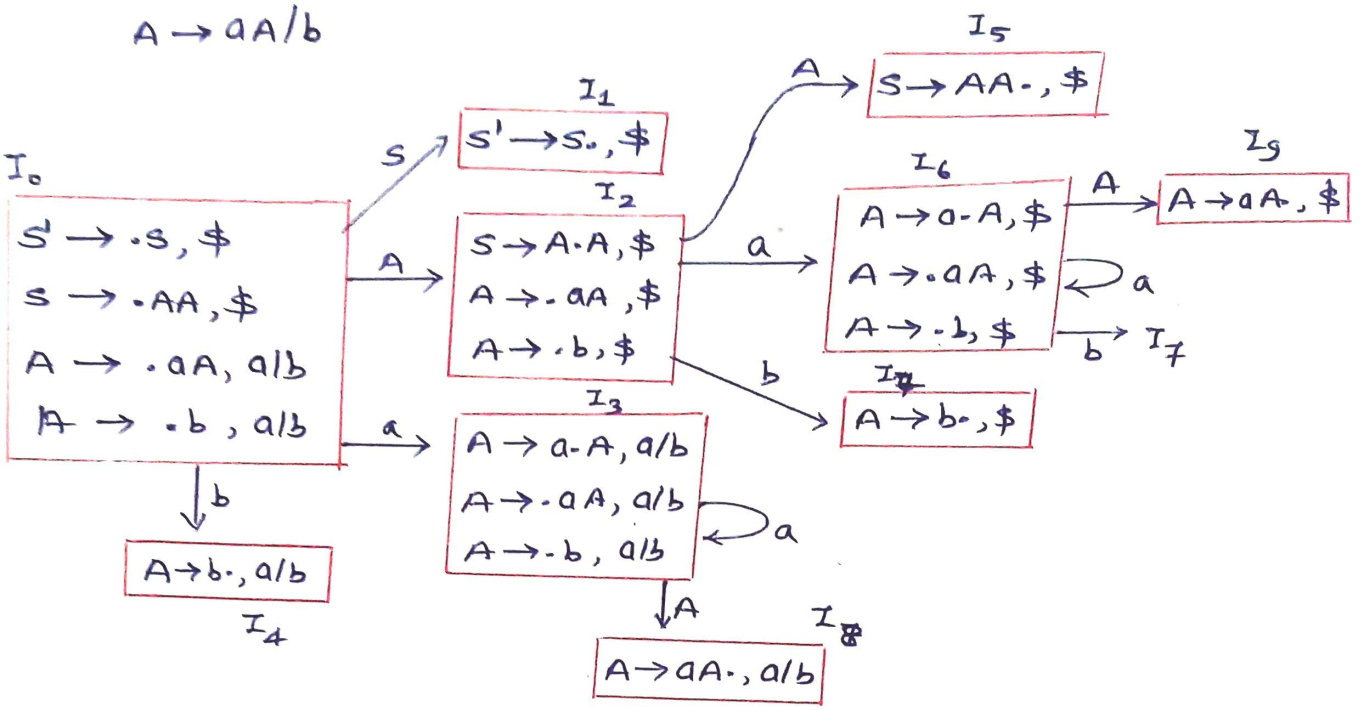


	Action			Goto	
	a	b	$	S	A
0	S_3	S_4		1	2
1			Accept		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

✱ LALR(1) :-

- Similar to the CLR, It reduced the size of parsing Table of CLR, Combining the similar states of CLR parsing Table.

Eg- $S \to AA$
$A \to aA/b$



⇒ I_3 and I_6 are similar, (I_4 and I_7, I_8 and I_9) are also similar

	Action			Goto	
	a	b	$	S	A
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 Table

	a	b	$	S	A
36	S_{36}	S_{47}			89
47	r_3	r_3	r_3		
89	r_2	r_2	r_2		

(Replace 3, 6, 4, 7, 8, 9 by these 3 Rows)

* operator precedence Parser-

- **operator Grammer-** A Grammer G does not Contain:
  1.) NULL Production
  2) 2 Adjacent variable on RHS of Production

$A \to \in X$
$A \to BC X$

$A \to BCD X$
$E \to E+E \mid E*E \mid id$ ✓

Exa- $P \to SR \mid S$
$R \to bSR \mid bS$
$S \to WbS \mid W$
$W \to L*W \mid L$
$L \to id$

$\Rightarrow P \to SbSR \mid SbS \mid S$
$R \to bP \mid bS$ ← (useless)
$S \to WbS \mid W$
$W \to L*W \mid L$
$L \to id$

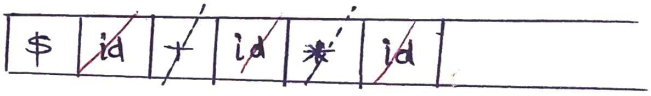
(operators Grammer)

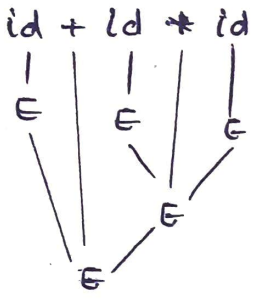
Let Grammer- $E \to E+E \mid E*E \mid id$

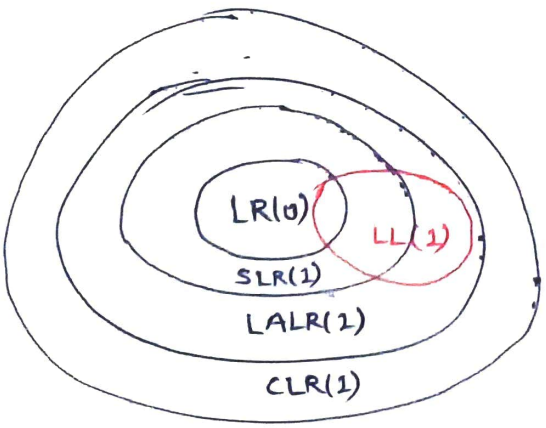
m	id	*	+	$
id	–	>	>	>
*	<	>	>	>
+	<	<	>	>
$	<	<	<	–

Operator Relation Table
id > (*, +, $)
* > * (Associativity

str = id + id * id $







LR(0) SLR(1) LALR(1) CLR(1)

$n_1 = n_2 = n_3 \leq n_4$