

## PPL

There are many languages (over 8000), instead of learning each, it is better to learn basic principles of any language.

Why study PPL?

- increased ability to learn new language
- To allow better choice of programming languages

### Tiobe statistics

1) Python

2) C

3) C++

4) Java

5) C#

6) JavaScript

Syntax - structure of language, word order, sentence composition.

Semantics - meaning of words.

Q If a code uses same algorithm on same device, but same different programming language. Can performance be different?

ans. Yes depends on data structures used and internal implementation of data structures.

## Models of programming & language

- 1) Imperative
- 2) Functional
- 3) Logic

### 1) Imperative.

Imperative Languages mimic Von Neumann Architecture

variables  $\leftrightarrow$  memory cells.

Assignment statements  $\leftrightarrow$  data piping between memory and CPU.

Operations and expressions  $\leftrightarrow$  CPU executions

Explicit control of execution flows  $\leftrightarrow$  program counter.

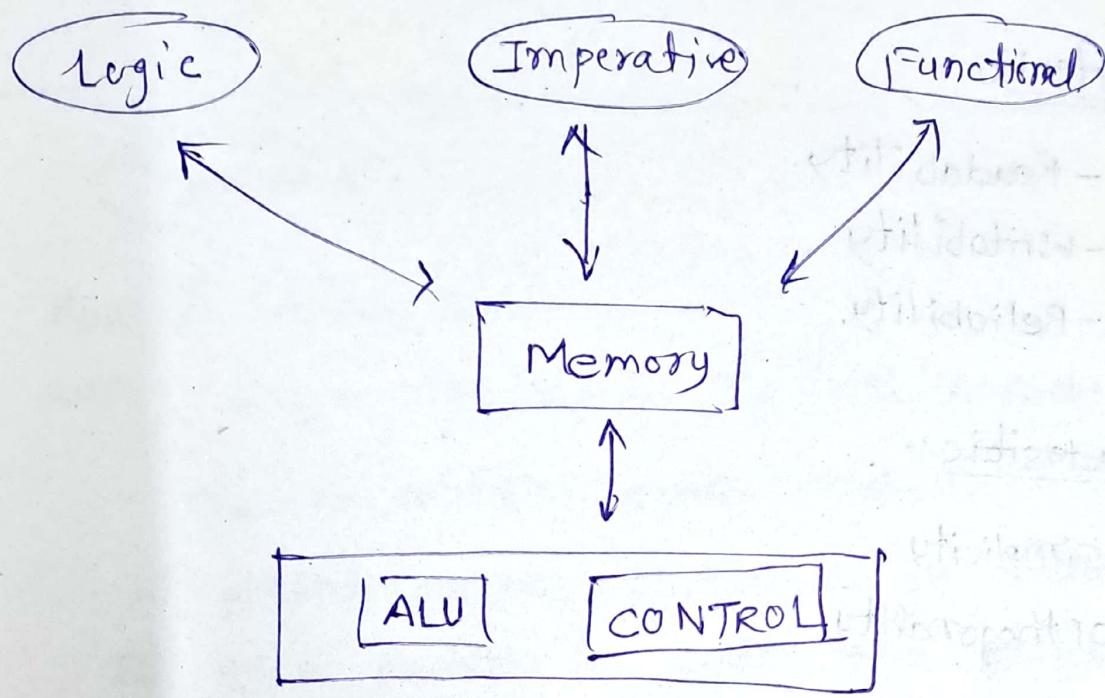
### 2) Functional

programming is like solving mathematical functions

A program, its subprograms are just implementations of mathematical functions

### 3) Logic

- program expressed as rule in formal logic
- Execution by rule resolution
- has facts, rules & goals



Von Neumann Architecture.

## Evaluation Criteria & Characteristic

### • Criteria

- Readability
- Writability
- Reliability.

### • Characteristics

simplicity

orthogonality

MEM

CLOSURE CONVENTION JULIA

INTERFACING ALGORITHMS

download putty.

Q) `int i=22;  
float p=22.3;  
printf("%d", i+p);`

ans:- garbage value.

$i+p \rightarrow 'i'$  will be promoted to float type.

but "%d" leads to demotion, thus loss of data, so we get garbage value

To prevent this, we need to do explicit conversion.

but `printf("%f", i+p);`  
will not give any error

### indentation

How to debug code?

ans:-  $\log_2 n$  complexity,  $n = \text{lines in code}$ .

Go to middle, `cout << "Hello";`

if printed correctly, problem is in code below "Hello".

else problem in code above "Hello"

Repeat this for problematic part also

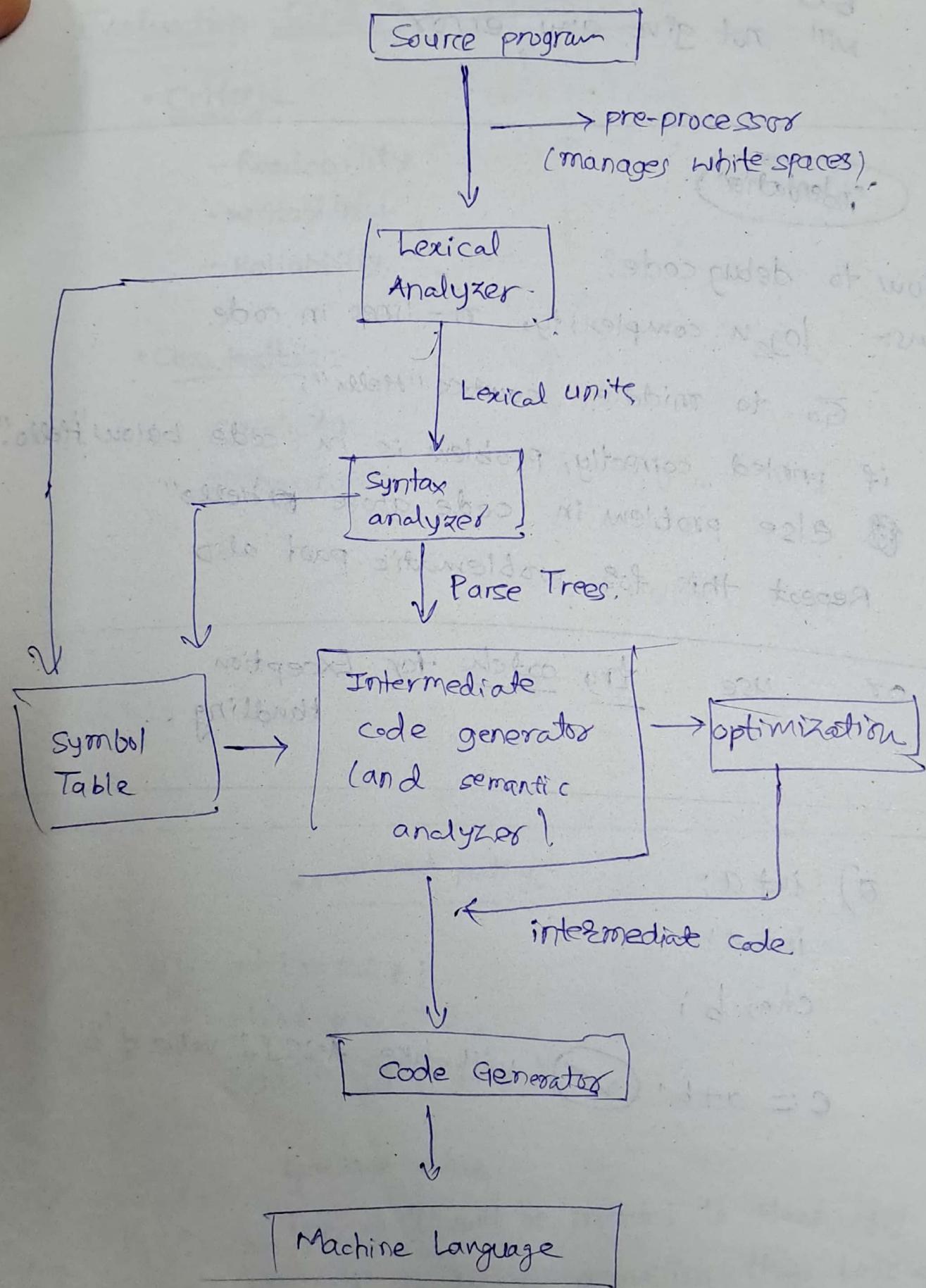
or use try catch for Exception  
Handling.

a) `int a;`

`int c;`

`char b;`

`c = a+b;` will take ASCII value of 'b'.



compiler does not stop at errors

It does error recovery. (Just skip, ignore & note  
the errors)

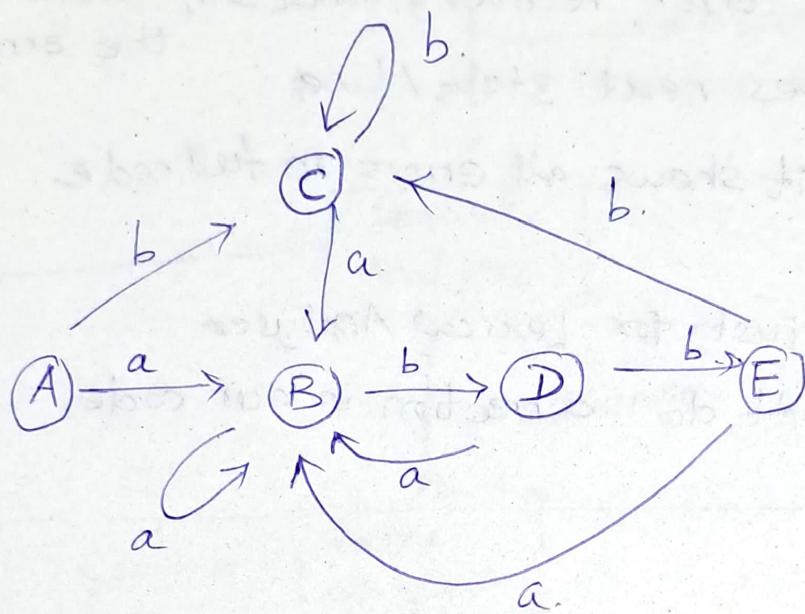
and moves next steps / Lines.

Finally it shows all errors in full code.

Recovery is just for Lexical Analyser

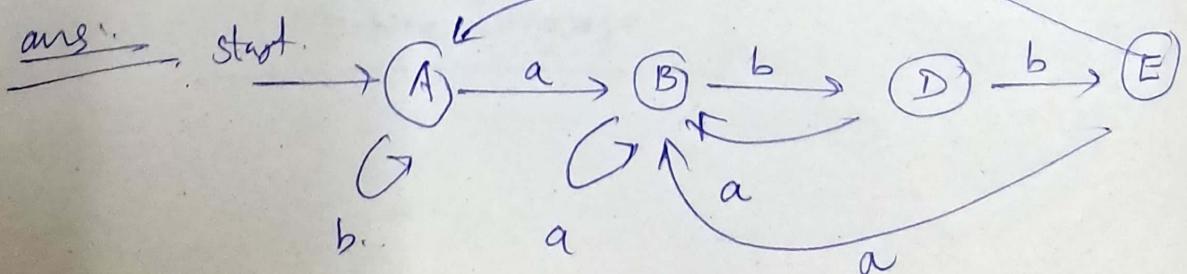
It doesn't do correction in our code.

\* Minimize the following DFA:-



ans:-

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



ans:- start.

~~$\pi_0 = \{A\}, \{B, C, D, E\}$~~

~~$\pi_1 = \{A\}, \{B, C, D, E\}$~~

$\pi_0 = \{A, B, C, D\} \{E\}$

$\pi_1 = \{A, B, C\} \{D\} \{E\}$

$\pi_2 = \{A\} \quad \{B, C\} \{D\} \{E\}$

error

Lex ] C language Lexical  
Analyser

fast ← Flex

JLex ] Java Lexical Analyser

Q] Try to find 5 errors that can arise in Lex phase?

Ans:-

~~Answers~~

## Find the FIRST and FOLLOW

$$S \rightarrow ABCD | \epsilon$$

$$A \rightarrow a | \epsilon$$

$$B \rightarrow bA$$

$$C \rightarrow a | \epsilon$$

$$D \rightarrow d$$

- it is for both terminals & non-terminals

ex:- Q]  $E \rightarrow E + T / T$

$$T \rightarrow F$$

$$F \rightarrow id$$

- For finding First, just see Left Hand side

$E \rightarrow E + T$  recursion - drop.

$E \rightarrow T$  — nonterminal - drop.

$T \rightarrow F$  — nonterminal - drop.

$F \rightarrow id$  — terminal.

Hence

$$\text{First}('E') = \{id\}$$

$$\text{First}('+' ) = \{+\}$$

$$\text{First}('id') = \{id\}$$

$$\text{First}('T') = \{id\}$$

$$\text{First}('F') = \{id\}$$

Note- For any terminal,

$$\text{First}('+') = \{+\}$$

$$\text{First}('terminal') = \{\text{terminal}\}$$

Q] Find First for following

$$E \rightarrow E+T \quad | \quad E-T \quad | \quad T$$

$$T \rightarrow T * F \quad | \quad T/F \quad | \quad F$$

$$F \rightarrow id$$

ans:-

$$E \rightarrow E+T \quad - \text{recursion-loop}$$

$$E \rightarrow E-T \quad - \text{recursion-loop}$$

$$E \rightarrow T \quad \text{nonterminal}$$

$$T \rightarrow F \quad \text{nonterminal}$$

$$F \rightarrow id \quad \boxed{\text{terminal } \checkmark}$$

$$\therefore \text{First}('E') = \{id\}$$

$$\text{First}('T') = ?$$

$$T \rightarrow T * F \quad - \text{recursion-loop}$$

$$T \rightarrow T/F \quad - \text{recursion-loop}$$

$$T \rightarrow F$$

$$F \rightarrow id \quad \boxed{\text{terminal } \checkmark}$$

$$\text{First}('F') = \{id\}$$



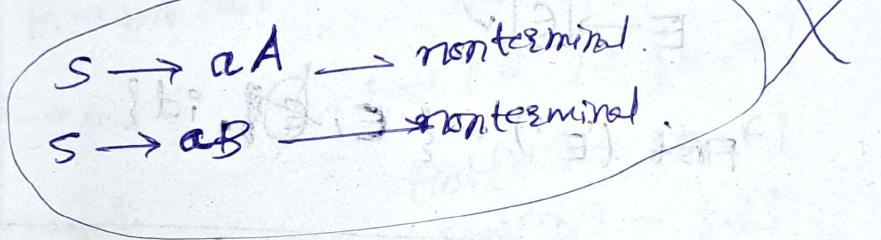
similarly

$$\text{FIRST}(T') = \{\text{id}\}$$

for terminals,

$$\begin{cases} \text{First}(+) = \{+\} \\ \text{First}(-) = \{-\} \\ \text{First}(/) = \{/\\} \\ \text{First}(\ast) = \{\ast\} \end{cases}$$

Q]  $S \rightarrow aA \mid aB$



$S \rightarrow aA$   
 terminal  
 non-terminal

$\text{FIRST}(S) = \{a, a\}$

Ambiguous  
grammar.

will ignore this.

Note:- Stop whenever you get terminal.

$S \rightarrow aA$   
 stop, this must be first.

$E \rightarrow E + T \quad / \quad T$   
 $T \rightarrow T * F \quad / \quad F$   
 $F \rightarrow id$

answ-

$E \rightarrow E + T$  — recursive rule

$E \rightarrow T$  — nonterminal,

$T \rightarrow [E] \checkmark$

$T \rightarrow F$

$F \rightarrow [id] \checkmark$

$E \rightarrow [E] \checkmark$

$\text{First}('E') = \{ E, \epsilon, id \}$

$\text{First}('T') = \{ id, \epsilon \}$

$\text{First}('F') = \{ id \}$

$\text{First}('+'') = \{ + \}$

$\text{First}('*'') = \{ * \}$

Q]  $S \rightarrow A B C D \mid \epsilon$

$A \rightarrow a \mid \epsilon$

$B \rightarrow bA \mid aA$

$C \rightarrow a \mid \epsilon$

$D \rightarrow d$

ans:-

$S \rightarrow \boxed{A} B C D$

$S \rightarrow \epsilon$

$\text{First}(S') = \{\epsilon, a\}, b\}$

$A \rightarrow \underline{a} \mid \epsilon$

$\text{First}(A') = \{a, \epsilon\}$

$B \rightarrow bA \mid aA$

$\text{First}(B') = \{b, a\}$

$C \rightarrow a \mid \epsilon$

$\text{First}(C') = \{a, \epsilon\}$

$D \rightarrow d$

$\text{First}(D') = \{d\}$

explanation

$S \rightarrow A B C D, S \rightarrow \epsilon$

$A \rightarrow a$

$S \rightarrow \boxed{a} B C D$

$A \rightarrow \boxed{\epsilon}$

$S \rightarrow B C D$

$b \rightarrow bA \mid aA$

$S \rightarrow \boxed{b} A C D$

$S \rightarrow \boxed{a} A C D$

$B C D$

$\boxed{b} A C D$

$\boxed{a} A C D$

{first, p }

$S \rightarrow \{a, b, \epsilon\}$

}

## FOLLOW

- Look at Right Hand side.
- 1 default symbol (augmented symbol) to add (ie \$).

because in PDA, bottom of stack has \$

Look at terminals after symbol for which you are calculating follow

stop after terminal reached;

Q]

$$E \rightarrow [E+T] / [E-T] / [T^*F] / E$$

$$T \rightarrow [T^*F] / [T-F] / F / E$$

$$F \rightarrow id.$$

ans:-

$$\text{Follow}(E) =$$

$$\{ \$, +, - \}$$

default.

T occurs at 5 positions

$$\text{Follow}(T) = \{ *, +, - \}$$

Note:- Q]  $E_a, T_a$

$$\text{Follow}(E) = \{\alpha\}$$

$$\text{Follow}(T) = \{\alpha\}$$

$$E \rightarrow T.$$

ans

Rule:-

$$E \rightarrow E + T$$

$$\text{follow}(T) = \text{follow}(E).$$

$$E \rightarrow E + T$$

$$E \_ a$$

$$E + T \_ a$$

T

Repeat

$$Q] E \rightarrow E + T / E - T / T / \epsilon$$

$$T \rightarrow T * F / T / F / \epsilon$$

$$F \rightarrow id$$

$$\text{Follow}(E) = \{ \$, +, -, / \}$$

$$\text{Follow}(T) = \{ \$, +, -, *, / \}$$

$$\text{Follow}(F) = \{ \$, +, -, *, / \}$$

ans:-



Scanned with OKEN Scanner

Rule 2

$$S \rightarrow A B C D$$

Follow (A) = first(B)

Follow (B) = first(C)

Follow (C) = first(D).

Repeat

Q]  $S \rightarrow ABCD / \epsilon$

$$A \rightarrow a / \epsilon$$

$$B \rightarrow bA$$

$$C \rightarrow a / \epsilon$$

$$D \rightarrow d$$

Rule 3

' $\epsilon$ ' not come  
in follow

Follow(S) = {} nothing  
hence \$.

Ans:

$$S \rightarrow ABCD$$

Follow(S) = {\$, a, b, } stop

$$A \rightarrow \epsilon$$

then

$$S \rightarrow BCD$$

$$B \rightarrow bA$$

$$S \rightarrow ABCD$$

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

$$B \rightarrow bA$$

A will include B elements

Follow(B) = {first(C)} = {a, \$, d}.

~~Follow(C) = {first(D)}~~

~~Follow(D) = {nothing}~~

Follow(C) = {first(D)} = {d}



Follow D = follow(s) = {} nothing.

= {} \$ {}

Q]  $S \rightarrow a \underline{B} \underline{D} h$

$B \rightarrow cC$

$C \rightarrow bC \mid \epsilon$

$D \rightarrow EF$

$E \rightarrow g \mid \epsilon$

$F \rightarrow f \mid \epsilon$

ans Follow(s) = {} \$ {}

Follow(F) = Follow(D) = {h}

Follow(B) = (first D) = {g, f, h}

Follow(C) = (follow B, first C) = {g, f, b, h}

Follow(D) = h.

Follow(E) = (first F) = f

Rule 4:- Remove useless / unreachable symbols

Rule 5:- If Grammar has ambiguity,  
then back track & ignore it.

$\frac{S \rightarrow aB \quad | \quad aC}{C \rightarrow e}$  will be avoided.  
 $B \rightarrow d$

website:- ANSI C grammar

[www.lysator.liu.se/~c/c-grammar-y.html](http://www.lysator.liu.se/~c/c-grammar-y.html)

# additive-expression

2024-even-venkat-iiita

= (1) wallet

= (2) wallet

= (3) wallet

= (4) wallet

= (5) wallet

= (6) wallet

= (7) wallet

= (8) wallet

= (9) wallet

= (10) wallet

= (11) wallet

= (12) wallet

= (13) wallet

= (14) wallet

= (15) wallet

= (16) wallet

= (17) wallet

= (18) wallet

= (19) wallet

= (20) wallet

= (21) wallet

= (22) wallet

= (23) wallet

= (24) wallet

= (25) wallet

= (26) wallet

= (27) wallet

= (28) wallet

= (29) wallet

= (30) wallet

= (31) wallet

= (32) wallet

= (33) wallet

= (34) wallet

= (35) wallet

= (36) wallet

= (37) wallet

= (38) wallet

= (39) wallet

= (40) wallet

= (41) wallet

= (42) wallet

= (43) wallet

= (44) wallet

= (45) wallet

= (46) wallet

= (47) wallet

= (48) wallet

= (49) wallet

= (50) wallet

= (51) wallet

= (52) wallet

= (53) wallet

= (54) wallet

= (55) wallet

= (56) wallet

= (57) wallet

= (58) wallet

= (59) wallet

= (60) wallet

= (61) wallet

= (62) wallet

= (63) wallet

= (64) wallet

= (65) wallet

= (66) wallet

= (67) wallet

= (68) wallet

= (69) wallet

= (70) wallet

= (71) wallet

= (72) wallet

= (73) wallet

= (74) wallet

= (75) wallet

= (76) wallet

= (77) wallet

= (78) wallet

= (79) wallet

= (80) wallet

= (81) wallet

= (82) wallet

= (83) wallet

= (84) wallet

= (85) wallet

= (86) wallet

= (87) wallet

= (88) wallet

= (89) wallet

= (90) wallet

= (91) wallet

= (92) wallet

= (93) wallet

= (94) wallet

= (95) wallet

= (96) wallet

= (97) wallet

= (98) wallet

= (99) wallet

= (100) wallet

= (101) wallet

= (102) wallet

= (103) wallet

= (104) wallet

= (105) wallet

= (106) wallet

= (107) wallet

= (108) wallet

= (109) wallet

= (110) wallet

= (111) wallet

= (112) wallet

= (113) wallet

= (114) wallet

= (115) wallet

= (116) wallet

= (117) wallet

= (118) wallet

= (119) wallet

= (120) wallet

= (121) wallet

= (122) wallet

= (123) wallet

= (124) wallet

= (125) wallet

= (126) wallet

= (127) wallet

= (128) wallet

= (129) wallet

= (130) wallet

= (131) wallet

= (132) wallet

= (133) wallet

= (134) wallet

= (135) wallet

= (136) wallet

= (137) wallet

= (138) wallet

= (139) wallet

= (140) wallet

= (141) wallet

= (142) wallet

= (143) wallet

= (144) wallet

= (145) wallet

= (146) wallet

= (147) wallet

= (148) wallet

= (149) wallet

= (150) wallet

= (151) wallet

= (152) wallet

= (153) wallet

= (154) wallet

= (155) wallet

= (156) wallet

= (157) wallet

= (158) wallet

= (159) wallet

= (160) wallet

= (161) wallet

= (162) wallet

= (163) wallet

= (164) wallet

= (165) wallet

= (166) wallet

= (167) wallet

= (168) wallet

= (169) wallet

= (170) wallet

= (171) wallet

= (172) wallet

= (173) wallet

= (174) wallet

= (175) wallet

= (176) wallet

= (177) wallet

= (178) wallet

= (179) wallet

= (180) wallet

= (181) wallet

= (182) wallet

= (183) wallet

= (184) wallet

= (185) wallet

= (186) wallet

= (187) wallet

= (188) wallet

= (189) wallet

= (190) wallet

= (191) wallet

= (192) wallet

= (193) wallet

= (194) wallet

= (195) wallet

= (196) wallet

= (197) wallet

= (198) wallet

= (199) wallet

= (200) wallet

= (201) wallet

= (202) wallet

= (203) wallet

= (204) wallet

= (205) wallet

= (206) wallet

= (207) wallet

= (208) wallet

= (209) wallet

= (210) wallet

= (211) wallet

= (212) wallet

= (213) wallet

= (214) wallet

= (215) wallet

= (216) wallet

= (217) wallet

= (218) wallet

= (219) wallet

= (220) wallet

= (221) wallet

= (222) wallet

= (223) wallet

= (224) wallet

= (225) wallet

= (226) wallet

= (227) wallet

= (228) wallet

= (229) wallet

= (230) wallet

= (231) wallet

= (232) wallet

= (233) wallet

= (234) wallet

= (235) wallet

= (236) wallet

= (237) wallet

= (238) wallet

= (239) wallet

= (240) wallet

= (241) wallet

= (242) wallet

= (243) wallet

= (244) wallet

= (245) wallet

= (246) wallet

= (247) wallet

= (248) wallet

= (249) wallet

= (250) wallet

= (251) wallet

= (252) wallet

= (253) wallet

= (254) wallet

= (255) wallet

= (256) wallet

= (257) wallet

= (258) wallet

= (259) wallet

= (260) wallet

= (261) wallet

= (262) wallet

= (263) wallet

= (264) wallet

= (265) wallet

= (266) wallet

= (267) wallet

&lt;

## Methods of parsing

Topdown

Bottom-up

- (i) If we go left  $\rightarrow$  right execution.  
will need to remove Left Recursion.

### Elimination of Left-Recursion :-

$$\begin{array}{l} A \rightarrow A\alpha \\ \text{ans:- } A \rightarrow \alpha A \mid E \end{array}$$

$$\begin{array}{l} A \rightarrow Aa \mid B \\ \text{ans:- } A \rightarrow BA' \\ A' \rightarrow \alpha A' \mid E \end{array}$$

Q

$$\begin{array}{c} A \qquad A \qquad \alpha \qquad B \\ \sqcap \qquad \sqcap \qquad \sqcap \qquad \sqcap \\ E \rightarrow E + T \qquad \qquad \qquad T \\ T \rightarrow T * F \qquad \qquad \qquad F \\ F \rightarrow (E) \qquad \qquad \qquad id \end{array}$$

ans:-

$$\begin{array}{l} E \rightarrow E + T \\ E \rightarrow TE' \\ E' \rightarrow +TE' \mid \epsilon \\ T \rightarrow FT' \\ T' \rightarrow *FT' \mid \epsilon \\ F \rightarrow (\neg E') \mid id \end{array}$$

Q2)  $E \rightarrow E + T \mid T \mid F$

$E \rightarrow TE' \mid FE'$

$E' \rightarrow +TE' \mid \epsilon$

Q3)  $E \rightarrow E + T \mid E + F \mid T \mid F$

$E \rightarrow TE' \mid FE'$

$E' \rightarrow +TE' \mid +FE' \mid \epsilon$

~~Q3~~

~~Q~~

- We will be seeing predictive parsing.
- We will not consider ambiguity and non-determinism

Ex:-

$$A \rightarrow a\beta \mid a\gamma$$

X

left factoring  
present

$$A \rightarrow aA'$$

$$A' \rightarrow \beta \mid \gamma$$



Left Factoring  
Removed.

Q]

~~Grammar~~

~~Q2~~

Note:-  $\epsilon, \in, \ddot{\epsilon}$  are same

will use  $\ddot{\epsilon}$  in this course.

TABLE MAKING

$$E \rightarrow T E'$$

$$E' \rightarrow + TE' | \ddot{e}$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' | \ddot{e}$$

$$F \rightarrow (E) | id.$$

$$E \rightarrow TE'$$

$$E' \rightarrow + TE'$$

$$E' \rightarrow \ddot{e}$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT'$$

$$T' \rightarrow \ddot{e}$$

$$F \rightarrow (E)$$

$$F \rightarrow id.$$

<u>Symbol</u>	<u>First</u>	<u>Follow</u>
---------------	--------------	---------------

E	(, id	\$, )
---	-------	-------

E'	+ , $\ddot{e}$	\$, )
----	----------------	-------

T	(, id	+ , \$, )
---	-------	-----------

T'	* , $\ddot{e}$	+ , \$, )
----	----------------	-----------

F	(, id	* , + , \$, )
---	-------	---------------

Based on first, follow we will create a table that parser will follow.

$E$  gets id, C

because of  $E \rightarrow TE'$  production

don't go in depth, i.e.  $(E \rightarrow TE')$ ,  $T' \rightarrow F \rightarrow id$ .

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

Note: If a cell has more than 1 entry, that means Grammar has Non-determinism.

Whenever  $A \rightarrow \epsilon$

write  $(A \rightarrow \epsilon)$  rule in follow of A.

ex:  $T' \rightarrow E$ ,  $\text{Follow}(T') = \{+, \$, )\}$

- Empty spaces are located as Error telling us that our syntax is wrong.

Conflicts :- cell with more than 1 entry !-

- ) First | First.
- ) First | Follow

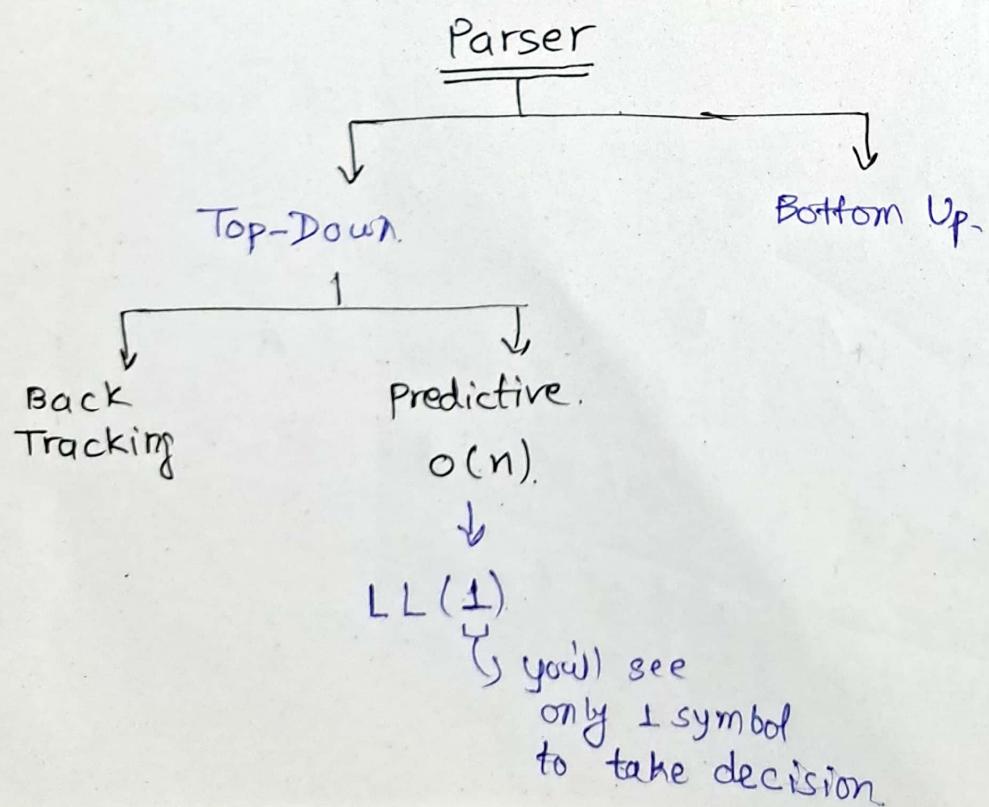
•) First | First       $A \rightarrow a\alpha | a\beta$   
ex:-                   $\text{first}(A) = \{a\}$

- ) First | Follow

$$\text{First}(\tau') = \{\ast, \epsilon\} \quad \text{Follow}(\tau') = \{\ast, +, \$, \}\}$$

$$\tau' \rightarrow \ast F \tau' | \epsilon$$

- Parser will only use table :  
ie. only Table passed in gcc.



\* Grammar is LL, when every cell has no multiple entry.

\* Parser uses PDA (push-down-automata).  
uses stack

Question :- id + id \* id \$

- When input == Top of stack  $\Rightarrow$  (Pop)
- '\$' is bottom of stack.

$E \rightarrow AB$ put in stack as Load in stack in Reverse Format.	$\$ E$ $\hookleftarrow \$ B A$
--	-----------------------------------

<u>Stack</u>	<u>Input</u>	<u>Output</u>
$\$ E$	id + id * id \$	$E \rightarrow TE'$
$\$ E' T$	id + id * id \$	$T \rightarrow FT'$
$\$ E' T' F$	id + id * id \$	$F \rightarrow id$
$\$ E' T' id$	id + id * id \$	matching, pop stack move input pointers
$\$ E' T'$	+ id * id \$	$T' \rightarrow E$
$\$ E'$	+ id * id \$	$E' \rightarrow + TE'$
$\$ E' T +$	+ id * id \$	match, pop

$\$ E' T$

↑

$\$ E' T' F$

↑

$\$ E' T' id$

↑

$\$ E' T'$

↑

$\$ E' T' F *$

↑

$\$ E' T' F$

↑

$\$ E' T' id$

↑

$\$ E' T'$

↑

$\$ E'$

↑

$\$$

bag two

$\uparrow$   $BT \leftarrow B$

$\uparrow$   $T = e$

$id * id \$$

↑

$id * id \$$

↑

$id * id \$$

↑

$* id \$$

↑

$* id \$$

↑

$id \$$

↑

$id \$$

↑

$\$$

↑

$\$$

↑

$\$$

↑

$\$$

↑

$T \rightarrow FT'$

$F \rightarrow id$

match, pop.

$T' \rightarrow *FT'$

match, pop.

$F \rightarrow id$

match, pop.

$T' \rightarrow e$

$E' \rightarrow e$

Accepted.

Q. If the input was ) id + id \* id \$

cang: seems invalid.

) id + id \* id \$

error valid. (prev question)

Recover and report at the end.

ie E, )  $\Rightarrow$  error or skip.

No compiler uses this parser.

All modern compilers use Bottom-up parser.

In C language, there are many productions with Left-Recursion. Hence using Bottom-up parser is useful.

LEX  $\rightarrow$  C/C++ Homework  
FLEX  
JLEX  $\rightarrow$  Java.

Do practical.  
TA will take viva.

If dot comes after a letter  
Means the production is recognised.

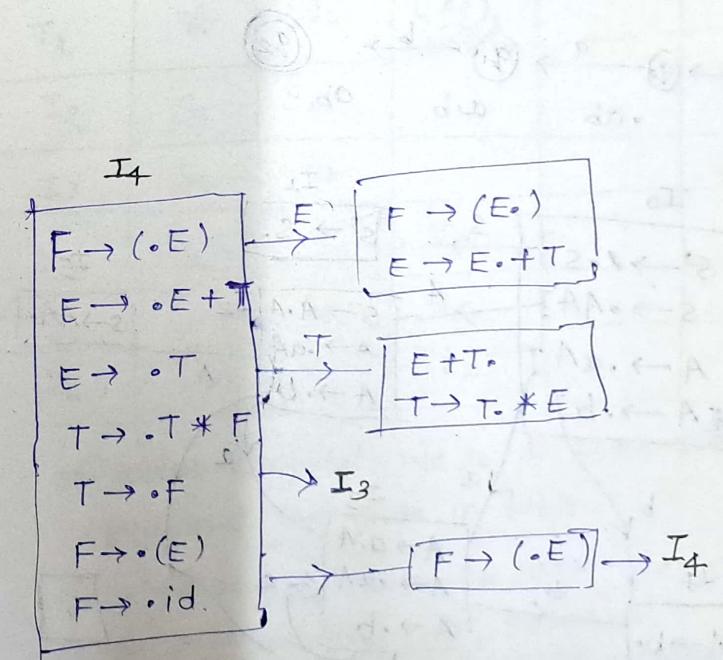
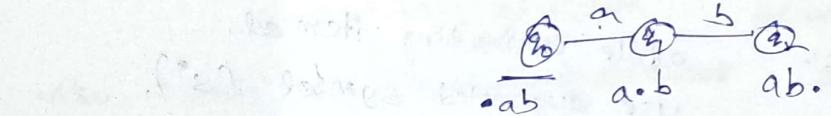
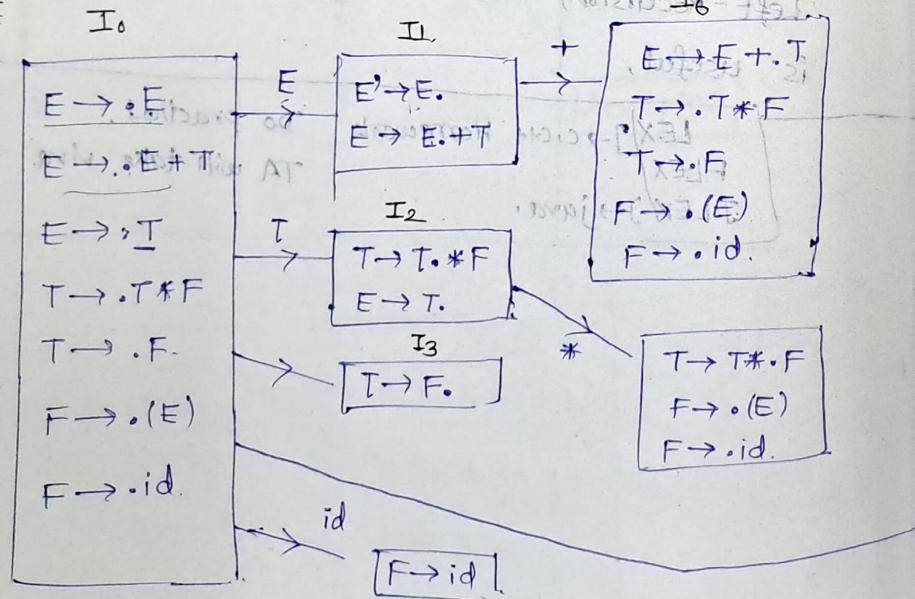
$\overbrace{E+}$ . T  
 $E+$  has been recognized.

$E+T$ . means the production is over now

- We can introduce new products also.  $E^9$  is new product for below example.

Q]  $E \rightarrow E + T \mid T$   
 $T \rightarrow T * F \mid F$

answ

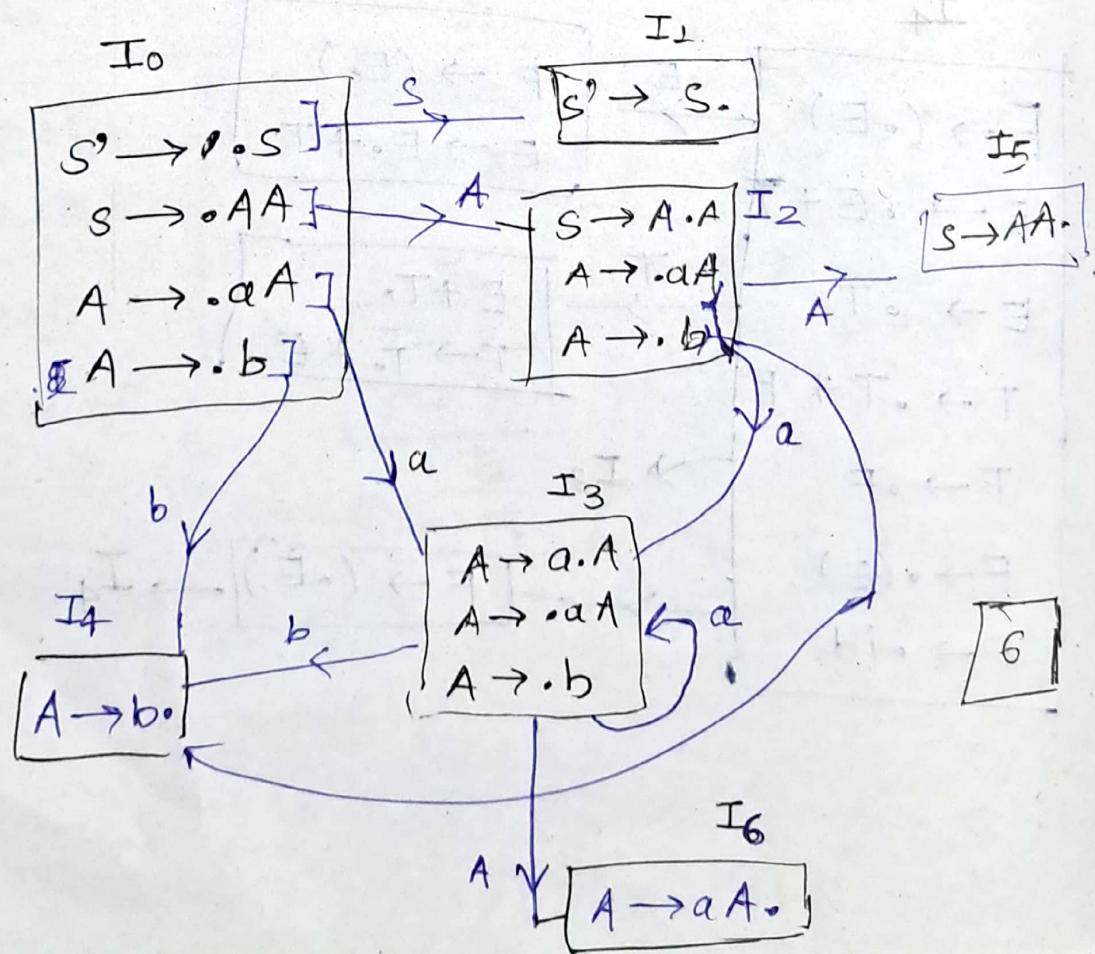
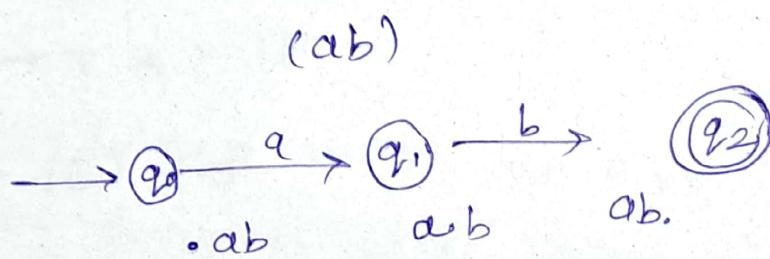


$$Q] \quad S \rightarrow A A$$

$$A \rightarrow a A \mid b$$

ans:- while constructing item set, use augmented symbol ( $S'$ ).

(•) is before non-terminal, then take closure of non-terminals until terminals occur / dot moves forward.



## Shift and Reduce

Instead of transition, we say it as shift as reduce.

Moving to new state

Taking dot forward.

- Also include \$, non-terminals & terminals.

LR(0) Table

	a	b	\$	S	A
I <sub>0</sub>	S <sub>3</sub>	S <sub>4</sub>		1	2
I <sub>1</sub>	γ <sub>0</sub>	γ <sub>0</sub>	γ <sub>0</sub>	Accepted	
I <sub>2</sub>	S <sub>3</sub>	S <sub>4</sub>			5
I <sub>3</sub>	S <sub>3</sub>	S <sub>4</sub>			6
I <sub>4</sub>	γ <sub>3</sub>	γ <sub>3</sub>	γ <sub>3</sub>		
I <sub>5</sub>	γ <sub>1</sub>	γ <sub>1</sub>	γ <sub>1</sub>		
I <sub>6</sub>	γ <sub>2</sub>	γ <sub>2</sub>	γ <sub>2</sub>		

on reading a symbol, we go to some state.

Fill no. of that state in table.

γ - reduce, will come for terminals, never for non-terminal.

Si - state denotion for terminals.

Si - state denotion for non-terminals.

1, 2, 3... - state denotion for augmented production.  
we will write Accept, only for augmented production.

For reduce, write the reduction rules  
and number them according to your wish

- ①  $S' \rightarrow S$
- ②  $S \rightarrow AA$
- ③  $A \rightarrow aA$
- ④  $A \rightarrow b$

Another Table for same parser SLR Table

	a	b	\$	S	A
I <sub>0</sub>	$S_3$	$S_4$		1.	
I <sub>1</sub>			Accept.		
I <sub>2</sub>	$S_5$	$S_4$			5
I <sub>3</sub>	$S_3$	$S_4$			6
I <sub>4</sub>	$\gamma_3$	$\gamma_3$	$\gamma_3$		
I <sub>5</sub>			$\gamma_1$		
I <sub>6</sub>	$\gamma_2$	$\gamma_2$	$\gamma_2$		

- We will only follow follows of particular symbol
- shift is same, for reduction difference comes

If any cell has more than 1 entry,  
The grammar is ambiguous.

We have already created  $LR(0)$ , itemset.

Now, Lets create  $LR(1)$ .

$$\underbrace{LR(1)}_{\text{Item set}} = \underbrace{LR(0)}_{\text{Item set}} + \underbrace{\text{Lookahead.}}_{\downarrow}$$

means we will also add follow of  
respective symbols in  $LR(0)$   
conversion.

E	$\rightarrow E + T$	<u>LR(0)</u>
	$E \rightarrow T$	$E^* \rightarrow .E$
T	$\rightarrow T * F$	$E \rightarrow .E + T$
T	$\rightarrow F$	$E \rightarrow .T$
F	$\rightarrow (E)$	$T \rightarrow .T * F$
F	$\rightarrow id$	$T \rightarrow .F$
		$F \rightarrow .(E)$
		$F \rightarrow .id$ .

for above example

$$\underbrace{LR(1)}_{\substack{\text{more items than} \\ \text{LR}(0)}} = \underbrace{LR(0)}_{\substack{\text{as seen} \\ \text{from} \\ LR(0) \text{ table}}} + \underbrace{\text{Lookahead}}_{g}$$

## LR(1)

$$E \rightarrow \cdot E, \$$$

$$E \rightarrow \cdot E + T, \$$$

$$E \rightarrow \cdot T, \$$$

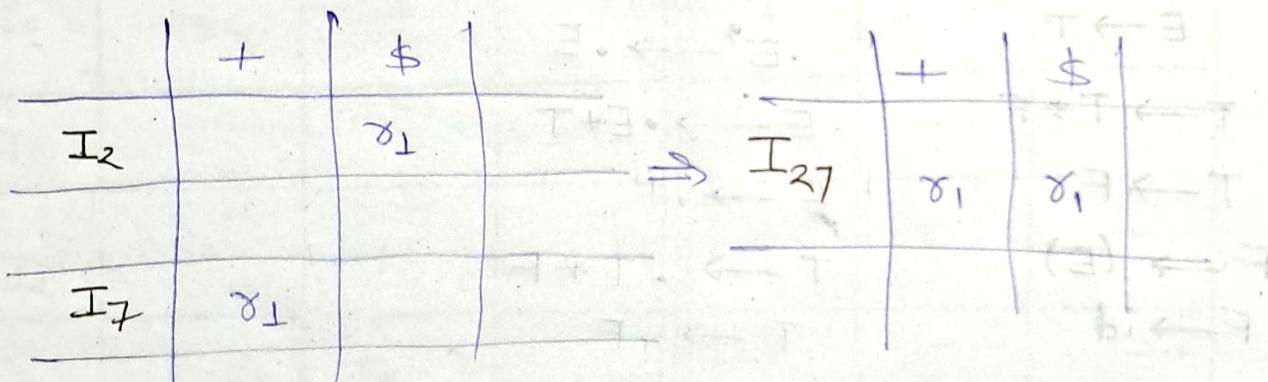
$$T \rightarrow \cdot T * F, \$$$

$$T \rightarrow \cdot F, \$$$

$$F \rightarrow \cdot (E), \$$$

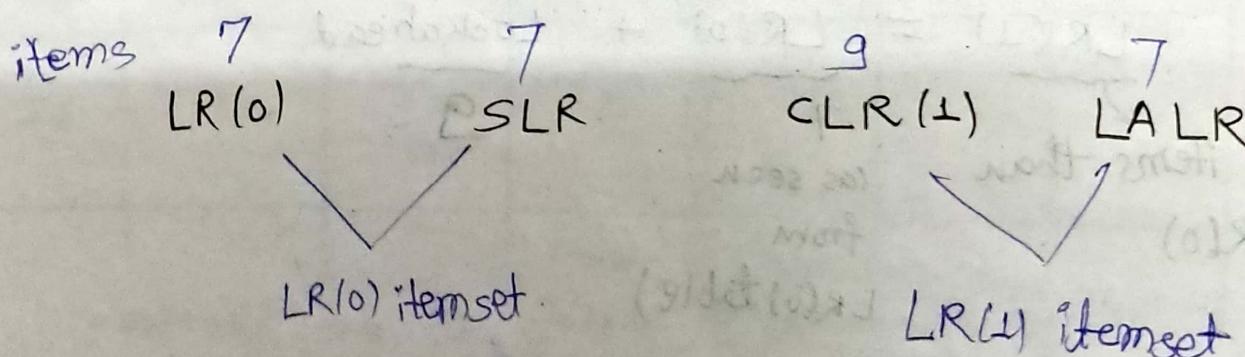
$$F \rightarrow \cdot \text{id}, \$$$

We can merge reductions.



Hence we will reduce no. of states so that space complexity reduces.  
( $SLR \rightarrow LALR$ )

Idea:-



Note - LALR is used by modern compiler.

Sometimes grammar accepted by CLR is rejected by LALR.

$I_2$	$S_1$	$\gamma_1$
$I_{27}$	$S_1, \gamma_1$	
$I_7$	$\gamma_1$	

$S_1, \gamma_1$



shift & reduce conflict.

What is Allowed?

ans:- Reduce & Reduce

What creates conflict?

ans:- shift-shift

shift-reduce

In exam, you should point out such conflicts directly to show that grammar will not accept it.

Instead of writing all grammar derivations.

If input is wrong, Apply Error Recovery Mechanism.

Input :- abb \$

(Look in LR(0) Table.

Some books write both number & symbol :- \$ 0 a3b4

while some books write only :- \$ 034

<u>Stack</u>	<u>Input</u>	<u>Action</u>	<u>Reduce Statement</u>
\$ 0	abb \$	shift	None
\$ 03	bb \$	shift	$\gamma_1 S \rightarrow AA$
\$ 034	b \$	Reduce	$\gamma_2 A \rightarrow aA$
\$ 03	b \$		$\gamma_3 A \rightarrow b$
pop / delete characters			
double of characters in front			
of \$			
\$ 034	b \$		
\$ 036	b \$		
\$ 02	b \$		
\$ 024	\$		
\$ 025	\$		
\$ 01	\$		
\$	\$		

## LR (0)

$E' \rightarrow \cdot E, \$$

$E \rightarrow \cdot E + T, \$$

$E \rightarrow \cdot T, \$$

$E \rightarrow \cdot E + T, +$

$E \rightarrow \cdot T, +$

$T \rightarrow \cdot T * F, \$$

$T \rightarrow \cdot F, \$$

$T \rightarrow \cdot T * F, +$

$T \rightarrow \cdot F, +$

$T \rightarrow \cdot T * F, *$

$T \rightarrow \cdot F, *$

$F \rightarrow \cdot (E), \$$

$F \rightarrow \cdot id, \$$

$F \rightarrow \cdot (E), +$

$F \rightarrow \cdot id, +$

$F \rightarrow \cdot (E), *$

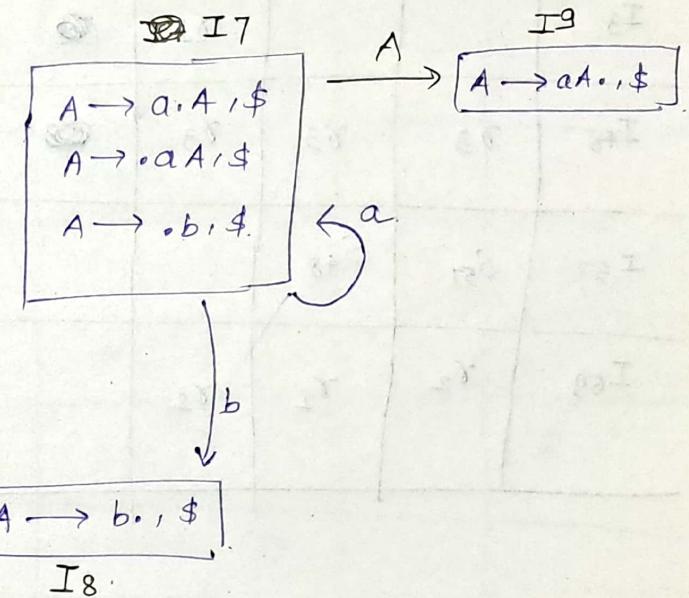
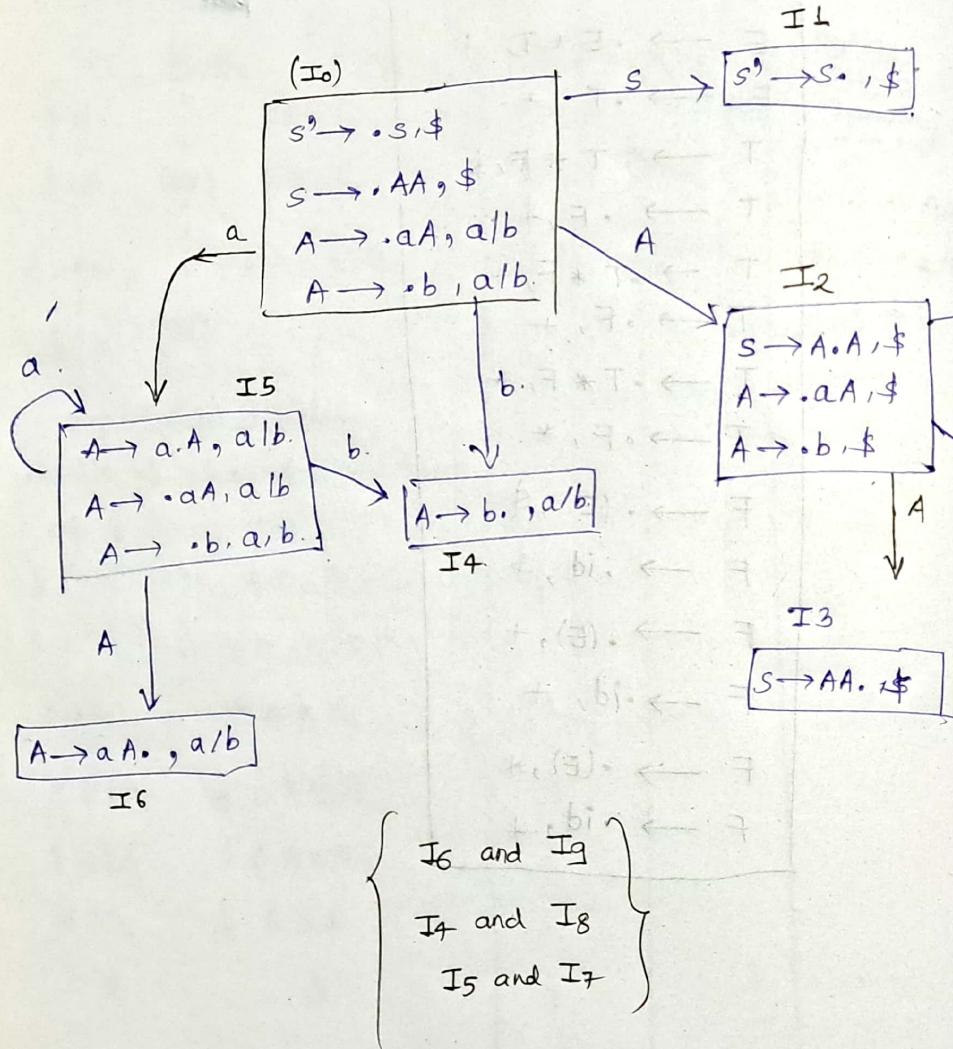
$F \rightarrow \cdot id, *$

$$\begin{array}{l} S \rightarrow AA \\ A \rightarrow aA \\ A \rightarrow b \end{array}$$

$$\begin{array}{l} 0 \rightarrow S' \rightarrow S \\ 1 \rightarrow S \rightarrow AA \\ 2 \rightarrow A \rightarrow aA \\ 3 \rightarrow A \rightarrow b \end{array}$$

CLR(↓)

bottom up parsing  
reached starting symbol.



# without seeing hollow part combine them to one who are same

$$6-9 \Rightarrow 69$$

$$4-8 \Rightarrow 48$$

$$5-7 \Rightarrow 57$$



LALR parser

	a	b	\$	S	A	
I <sub>0</sub>	$S_{57}$	$S_{48}$		1	2	
I <sub>1</sub>			accepted.			
I <sub>2</sub>	$S_{57}$	$S_{69}$			3	
I <sub>3</sub>			$\gamma_1$	$\gamma_1$		
I <sub>48</sub>	$\gamma_3$	$\gamma_3$	$\gamma_3$	$\gamma_3$	$\gamma_3$	
I <sub>57</sub>	$S_{57}$	$S_{48}$			69	
I <sub>69</sub>	$\gamma_2$	$\gamma_2$	$\gamma_2$			

End  $\leftarrow A$

8I

at most six times from well-formed function #  
since 910 only 900

$$P_2 \leftarrow P \rightarrow$$

$$S_4 \leftarrow S - P$$

$$T_2 \leftarrow T - e$$

CLR(1)

	a	b	\$	s	A
I <sub>0</sub>	S <sub>5</sub>	S <sub>4</sub>		1	2
I <sub>1</sub>			accepted		
I <sub>2</sub>	S <sub>7</sub>	S <sub>6</sub>			3.
I <sub>3</sub>			τ <sub>1</sub>		
I <sub>4</sub>	τ <sub>3</sub>	τ <sub>3</sub>			
I <sub>5</sub>	S <sub>5</sub>	S <sub>4</sub>			6
I <sub>6</sub>	τ <sub>2</sub>	τ <sub>2</sub>			
I <sub>7</sub>	S <sub>1</sub>	S <sub>8</sub>			g
I <sub>8</sub>			τ <sub>3</sub>		
I <sub>9</sub>			τ <sub>2</sub>		