S.THARUN-19MID0031

13)	1)	With the help of neat block diagram explain various phases of compiler ,Also write down the output of each phase for expression a:=b+c*50
	2)	Construct LR(1). S→x|Ay B→ε|z A→Bx

PHASES OF COMPILER:

Lexical Analysis:

Lexical analyser divides the program into tokens (scanning)

eg $a = b + 5 - (c * d)$

Token Type	Value
Identifier	a, b, c, d
operator	+, -, *
constant	5
Delimeter	(,)

Syntax Analysis:

* It takes list of tokens produced by Lexical Analysis.
* then, these tokens are arranged in a tree like structure (syntax tree), which reflects program structure
* Also known as parsing

Semantic Analysis:

* It validates the syntax tree by applying rules & regulations of the target language.
* It does type checking, scope resolution, variable declaration, etc.
* It decorates the syntax tree by putting data types, values, etc.

Intermediate Code Generation:

* The program is translated to a simple machine independent intermediate language
* Register allocation of variables is done in this phase.

Code optimization:

* It aims to reduce process timings of any program
* It produces efficient programming code.
* It is an optional phase.
* Removing unreachable code
* Getting rid of unused variables
* Eliminating multiplication by 1
  addition by 0
* Removing statements that are not modified from the loop.
* Common sub-expression elimination

Code Generation

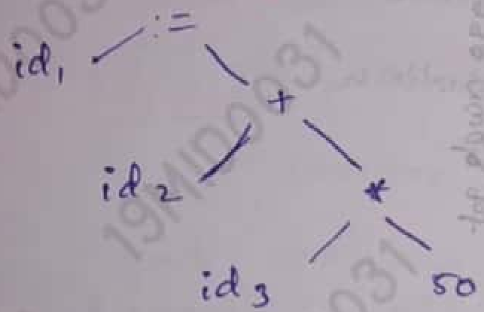
* Target program is generated in the machine language of the target architecture.
* Memory locations are selected for each variable.
* Instructions are chosen for each operation
* Individual tree nodes are translated into sequence of machine language instructions

all others → float   Price := amount + rate * 50 → int

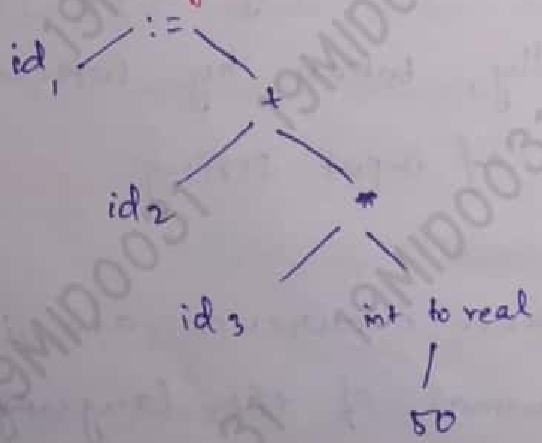
1) Lexical Analyser: removes spaces, comments

$id_1 := id_2 + id_3 * 50$

2) Syntax Analyser:



3) Semantic Analyser:



4) Intermediate Code Generator:

temp1 := int to real (50)
temp2 := id_3 * temp1
temp3 := id_2 + temp2
id_1 := temp3

5) Code optimizer:

temp1 := id_3 * 50.0
id_1 = id_2 + temp1

6) Code Generator:

```
MOVF R2, id3
MULF R2, #50.0
MOVF R1, id2
ADDF R1, R2
MOVF id1, R1
```

2) construct LR(1)

$S \rightarrow x \mid Ay$

$B \rightarrow \varepsilon \mid z$

$A \rightarrow Bx$

Augmented Grammar

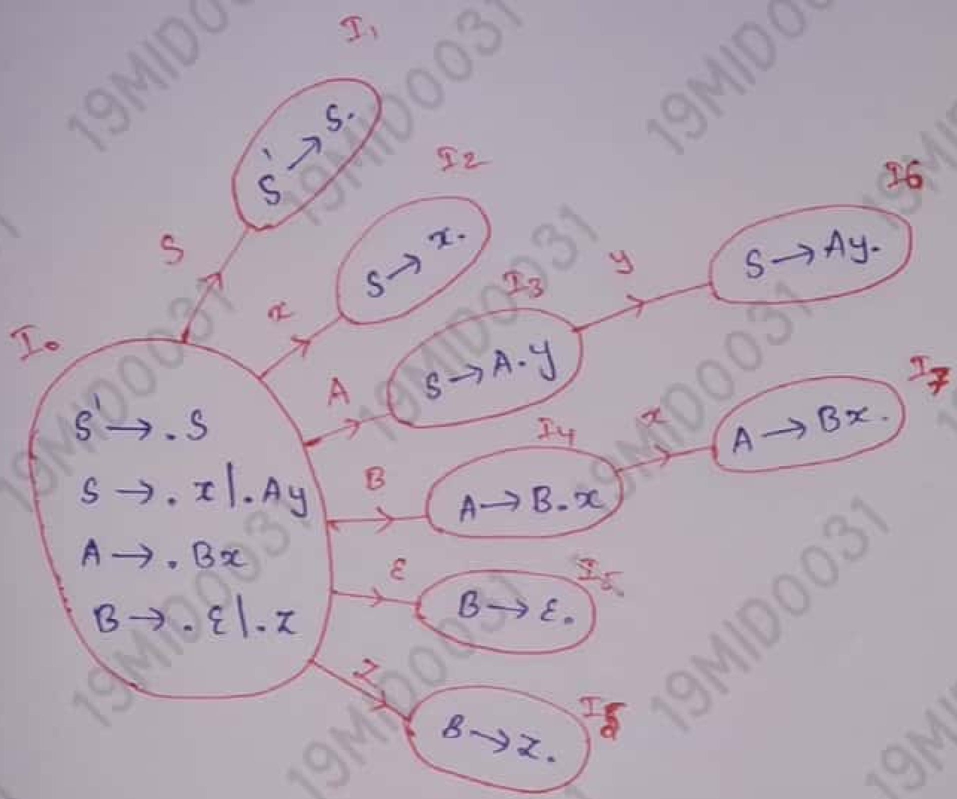
$S' \rightarrow S$

$S \rightarrow x \mid Ay$ = r_1 and r_2

$B \rightarrow \varepsilon \mid z$ = r_3

$A \rightarrow Bx$ = r_4

canonical forms



	ACTION				GOTO		
	x	y	z	$	A	B	S
0	S2		S5		3	4	1
1				accept			
2				r1			
3		S6					
4	S7						
5				r3			
6				r2			
7				r4			

35) 1) Consider the grammar

S→(L) | a

L→L,S | S

f) What are the terminal, non-terminal and start symbol?

g) Find parse tree for the following sentences

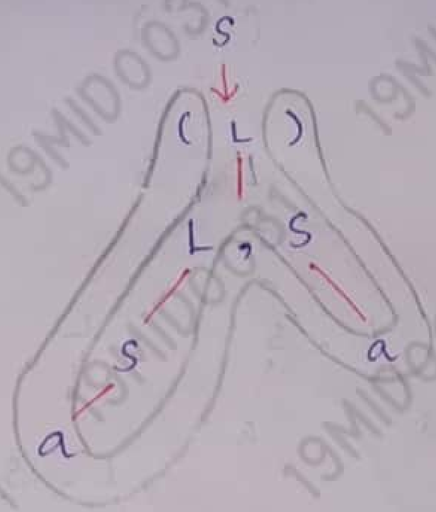
(iv) (a,a)

(v) (a,(a,a))

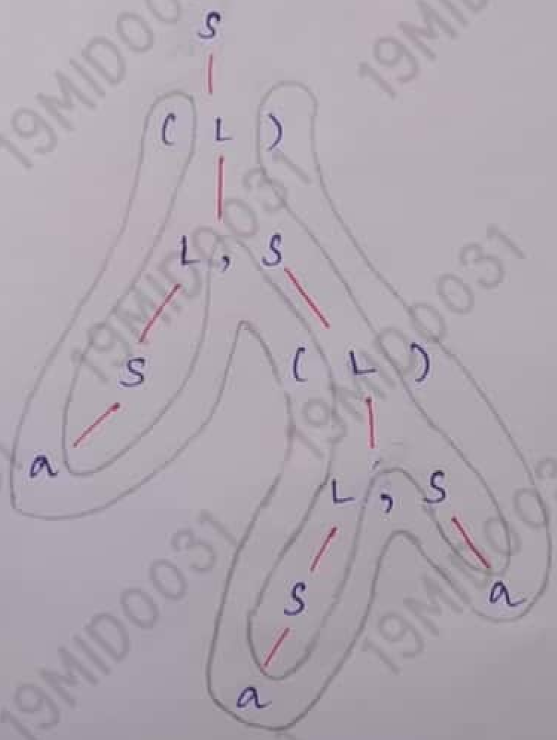
(vi) (a,((a,a),(a,a)))

35) $S \rightarrow (L) \mid a$
$L \rightarrow L, S \mid S$

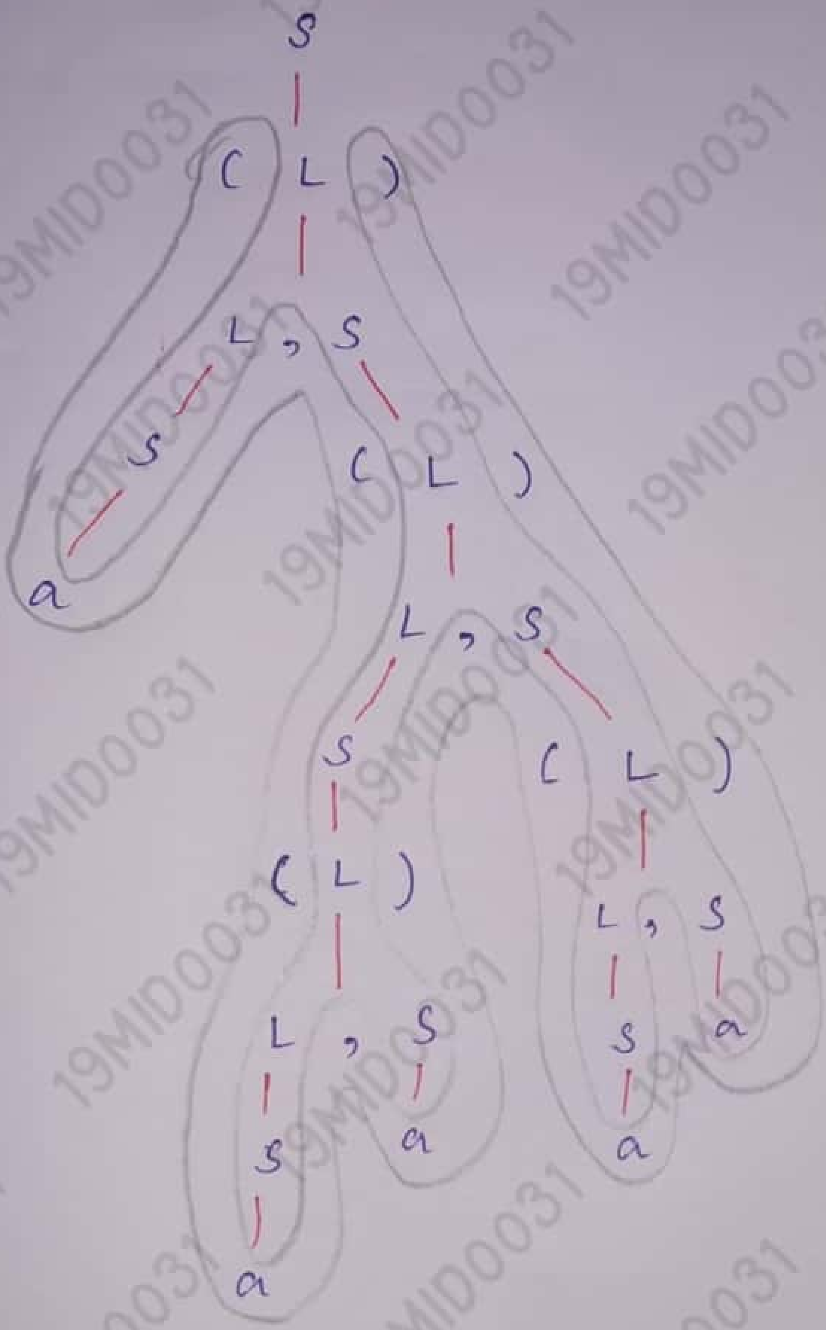
* terminals = { (,), ,, a }
* Non-terminals = { S, L }
* Start symbol = { S }
* parse tree for (a, a)



* parse tree for (a, (a, a))



* parse tree for (a, ((a, a), (a, a)))



40)

1) What are the issues of the Lexical analyser?
2) Eliminate left recursion, perform left factoring and find: FIRST & FOLLOW

E→E+T|T

T→id|id [ ] |id [X]

X→E,E|E

3) Check the given grammar is LL(1) or NOT?

S→(A) | 0

A→SB

B→,SB|ε and also parse the grammar (0,(0,0))

ISSUES IN LEXICAL ANALYSIS

we do seperate the work of Lexical Analysis and syntax Analysis for the following reasons.

* **Simplicity of design**

A parser containing the rules for comments and white space is more complex to make than a parser that can assume that comments & white spaces has been removed.

* **Improved compiler Efficiency:**

Reading source code & classifying it in token is time consuming task. when we separate from parser, it allows us to use specialized technique for lexer, which can speed up scanning.

* **Higher probability:**

Input device specific peculiarities are restricted to lexer

Lexical Errors:

A character sequence which is not possible to scan into any valid token is a lexical error.

It's hard for lexical analyzer without the aid of other components, that there is a source code error.

Eg: If the statement "if" is encountered for the first time in a C program, it can not tell whether fi is misspelling of "if" statement or a undeclared literal.

Probably the parser in this case will be able to handle this

* Also Error Handling is very localized with respect to input source.

Eg: whil (x = 0) do generates no lexical error is PASCAL.

Handling Lexical Errors:

* Panic Mode Recovery

Delete successive characters from the remaining input until the analyzer can find a well formed token.

→ May confuse parser by creating syntactical errors.

* possible error Recovery Actions:

→ Deleting extra irrelavent character

→ Inserting missing input character

→ Replacing an incorrect character by a correct character

→ Transposing two adjacent characters

Input Buffering:

The amount of time taken to process characters of a large source program. Lexical analyzer may need to look at least a character ahead to make a token decision

Sentinels:

During buffering for each character.

→ Check the end of buffer

→ determine what character is read

40) 2) $E \rightarrow E + T \mid T$ — ①

$T \rightarrow id \mid id[\,] \mid id[X]$ — ②

$X \rightarrow E, E \mid E$ — ③

* First production has Left Recursion.

$E \rightarrow E + T \mid T$

Removing Left Recursion.

$E \rightarrow TE'$

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

* Second production has Left factoring

$T \rightarrow id \mid id[\,] \mid id[x]$

Removing left factoring

$T \rightarrow id\,T'$

$T' \rightarrow \varepsilon \mid [\,] \mid [x]$

Now the productions are

$E \rightarrow TE'$ —— ①

$E' \rightarrow \varepsilon \mid TE'$ —— ②

$T \rightarrow idT'$ —— ③

$T' \rightarrow \varepsilon \mid [\,] \mid [X]$ —— ④

$X \rightarrow E,E \mid E$ —— ⑤

FIRST(E) = { id }

FIRST(E') = { ε, id }

FIRST(T) = { id }

FIRST(T') = { ε, [}

FIRST(X) = { id }

FOLLOW(E) = { $, ,,] }

FOLLOW(E') = { $, ,,] }

FOLLOW(T) = { id, $, ,,] }

FOLLOW(T') = { id, $, ,,] }

FOLLOW(X) = {] }

A0) 3) $S \rightarrow (A) \mid 0$

$A \rightarrow SB$

$B \rightarrow , SB \mid \varepsilon$

No left recursion nor left factoring

FIRST(S) = { (, 0 } FOLLOW(S) = { , ,), 0, $ }

FIRST(A) = { (, 0 } FOLLOW(A) = {) }

FIRST(B) = { , } FOLLOW(B) = {) }

	S	A	B
(	S→(A)	A→SB	
)			B→ε
,			B→,SB
0	S→0	A→SB	

Stack	Input	production
S $	(0,(0,0))	
(A) $	(0,(0,0))	S→(A)
SB) $	0,(0,0))	A→SB
~~0~~B) $	~~0~~,(0,0))	S→0
~~,~~SB) $	~~,~~(0,0))	B→,SB
~~(~~A)B) $	~~(~~0,0))	S→(A)
SB)B) $	0,0))	A→SB
~~0~~B)B) $	~~0~~,0))	S→0
~~,~~SB)B) $	~~,~~0))	B→,SB
~~0~~B)B) $	~~0~~))	S→0
~~)~~B) $	~~)~~)	B→ε
~~)~~ $	~~)~~	B→ε

Now stack is empty with $, so accepted

23) 1) Define handle and handle pruning?

2) Construct LR parsing table

$E \rightarrow E+T$
$E \rightarrow T$
$T \rightarrow T*F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow id$ *id*id+id* using stack implementation.

23) Handles

1) * Formally, Handle of a right sentential form γ is $<A \rightarrow \beta$ location of β in $\gamma>$

* i.e) $A \rightarrow B$ is a handle of $\alpha\beta\gamma$ at the location immediately after the end of α, if $S \Rightarrow \alpha A\gamma \Rightarrow \alpha\beta\gamma$

* A certain sentential form may have many different handles.

* Right sentential forms of a non-ambiguos grammar have one unique handle.

Handle pruning

* the process of discovering a handle & reducing it to the appropriate left hand side is called handle pruning

* Handle pruning forms the basis for a bottom-up parsing method.

23) 2) $E \rightarrow E+T$

$E \rightarrow T$

$T \rightarrow T*F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$ R.

state	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
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	Input	production
0	id*id+id $	s5
0 id 5	* id +id $	r6
0 F 3	* id +id $	r4
0 T 2	* id +id $	s7
0 T 2 * 7	id +id $	s5
0 T 2 * 7 id 5	+ id $	r6
0 T 2 * 7 F 10	+id $	r3
0 T 2	+ id $	r2
0 E 1	+id $	s6
0 E 1 + 6	id $	s5
0 E 1 + 6 id 5	$	r6
0 E 1 + 6 F 3	$	r4
0 E 1 + 6 T 9	$	r1
0 E 1	$	accept

24)

1) Differentiate between final states in a NFA and a DFA

2) Table:

Remove left recursion	Remove left Factoring
A→Aα |β	S→iEtS|iEtSeS|a E→b
S→Aa|b A→Ac|Sd|ε	Stmt→if expr then Stmt else Stmt|ifexpr then Stmt
S→aBDh S→Bb|C D→EF E→g|ε F→f|ε	S→aSb|aTc T→dTU|ε U→f
S→SA|SB|a|b|c	

24)

1) FINAL STATE OF DFA

$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q \rightarrow$ set of states

$\Sigma \rightarrow$ alphabets

$\delta \rightarrow$ transition function

$q_0 \rightarrow$ Initial state.

$F \rightarrow$ Final state

Final state $\Rightarrow$ It is non empty set of final states / accepting states from the set belonging to Q

FINAL STATE OF NFA

$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q \rightarrow$ set of states

$\Sigma \rightarrow$ alphabets

$\delta \rightarrow$ transition function

$q_0 \rightarrow$ Initial state

$F \rightarrow$ Final state

Final state $\Rightarrow$ A non empty set of final states and member of Q.

24)

2) REMOVE LEFT RECURSION

(i) $A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

2) $S \rightarrow Aa \mid b \Rightarrow S \rightarrow sda \mid b$

$A \rightarrow Ac \mid sd \mid \varepsilon \Rightarrow A \rightarrow Ac \mid Aad \mid \varepsilon \mid bd$

$S \rightarrow sda \mid b \Rightarrow \boxed{\begin{array}{l} S \rightarrow bS' \\ S' \rightarrow daS' \mid \varepsilon \end{array}}$

$A \rightarrow Ac \mid Aad \mid \varepsilon \mid bd \Rightarrow A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$

$A \quad A\alpha_1 \quad A\alpha_2 \quad \beta_1 \quad \beta_2$

$\Rightarrow \boxed{\begin{array}{l} A \rightarrow bdA' \mid \varepsilon A' \\ A' \rightarrow cA' \mid adA' \mid \varepsilon \end{array}}$

$\Rightarrow$
$S \rightarrow bS'$
$S' \rightarrow daS' \mid \varepsilon$
$A \rightarrow bdA' \mid \varepsilon A'$
$A' \rightarrow cA' \mid \varepsilon \mid adA'$

1/1

(iii) $S \rightarrow aBDh$

$S \rightarrow Bb \mid c$

$D \rightarrow EF$

$E \rightarrow g \mid \epsilon$

$F \rightarrow f \mid \epsilon$

No Left Recursion in this example

(iv) $S \rightarrow SA \mid SB \mid a \mid b \mid c$

$S \rightarrow aS' \mid bS' \mid c$

$S' \rightarrow AS' \mid BS' \mid \epsilon$

REMOVE LEFT FACTORING

(i) $S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow \epsilon \mid eS$

$E \rightarrow b$

(ii) stmt → if expr then stmt else stmt | if expr then stmt

(Same as above)

Stmt → if expr then Stmt Stmt'

Stmt' → ε | else stmt

(iii) $S \rightarrow aSb \mid aTc$

$T \rightarrow dTU \mid \epsilon$

$U \rightarrow f$

$S \rightarrow aS'$

$S' \rightarrow \epsilon \mid Sb \mid Tc$

$T \rightarrow dTU \mid \epsilon$

$U \rightarrow f$

3)

1) Construct DAG for

a) (a-b)+c*(d/e)

b) x=x+x*y

c) (x+5)*(x+5+y)

d) a=(a+a)+a(a+a+a)+a

2) Check the given grammer is LL(1) or NOT?

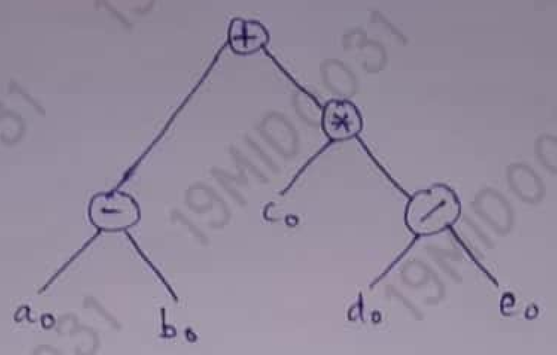
S→(A) | 0

A→SB

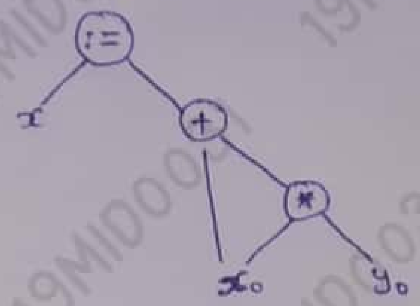
B→,SB|ε

and also parse the grammar (0,(0,0))

3) 1) (a) $(a-b)+c*(d/e)$

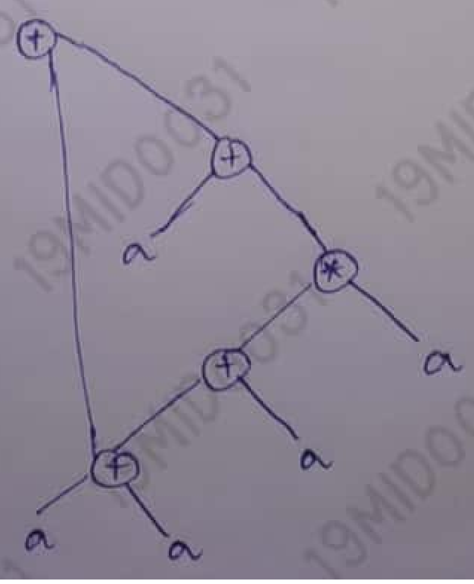


(b) $x = x + x * y$



(c) $(x+5) * (x+5+y)$

(d) $a = (a+a) + a(a+a+a) + a$



3) $S \to (A) \mid 0$

2) $A \to SB$

$B \to , SB \mid \varepsilon$

No Left recursion nor Left factoring

FIRST(S) = { (, 0 } FOLLOW(S) = { , ,), 0, $ }

FIRST(A) = { (, 0 } FOLLOW(A) = {) }

FIRST(B) = { , }, ε FOLLOW(B) = {) }

	S	A	B
(	S→(A)	A→SB	
)			B→ε
,			B→,SB
0	S→0	A→SB	

Stack	Input	production
S $	(0,(0,0))	
(A) $	(0,(0,0))	S→(A)
SB) $	0,(0,0))	A→SB
~~0~~B) $	~~0~~,(0,0))	S→0
~~,~~SB) $	~~,~~(0,0))	B→,SB
~~(~~A)B) $	~~(~~0,0))	S→(A)
SB)B) $	0,0))	A→SB
~~0~~B)B) $	~~0~~,0))	S→0
~~,~~SB)B) $	~~,~~0))	B→,SB
~~0~~B)B) $	~~0~~))	S→0
~~)~~B) $	~~)~~)	B→ε
~~)~~ $	~~)~~	B→ε

Now stack is empty with $, so accepted

1)

1) Check the given Grammar is ambiguous/Unambiguous

$S \rightarrow S(S)S \mid \varepsilon$

2) Prove the given grammar is LR(1),LALR(1),Not SLR(1).

$S \rightarrow Aa \mid bAc \mid dc \mid bda$

$A \rightarrow d$

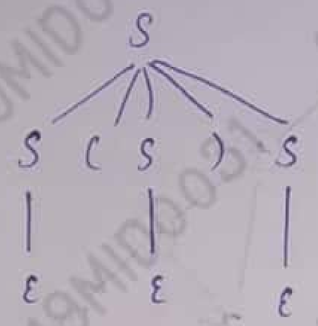
1) (i) $S \to S(S)S \mid \varepsilon$

Let us consider the string => ( )

LMD

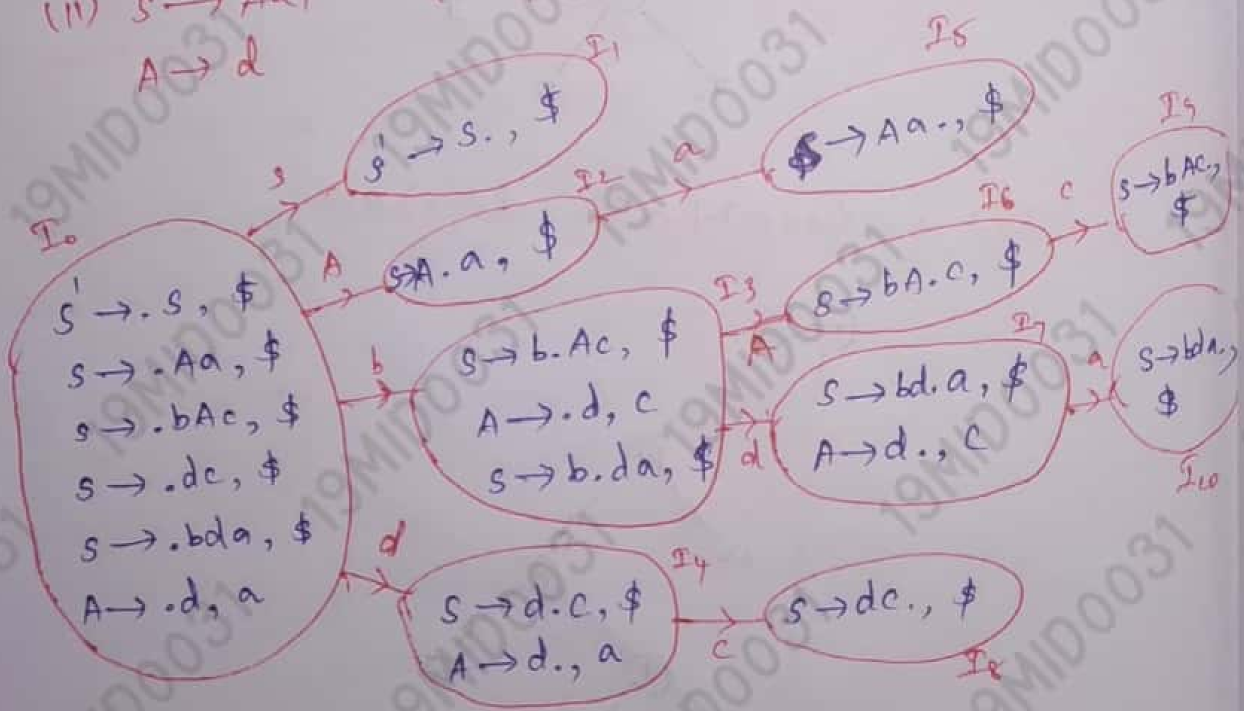
$S \to S(S)S \to \varepsilon(S)S \to (S)S$
$\to (\varepsilon)S \to ()S$
$\to ()\varepsilon \to ()$

Parse tree



We are able derive atmost one left most derivation for the string ( ). Also only one way of parse tree is available. Hence the given grammar is unambiguous.

(ii) $S \to Aa \mid bAc \mid dc \mid bda$
$A \to d$



state	Action					Goto	
	a	b	c	d	$	S	A
0		S_3		S_4		1	2
1					accept		
2	S_5						
3				S_7			6
4			S_8				
5							
6			S_9				
7	S_{10}		r_5				
8				r_3			
9				r_2			
10				r_4			

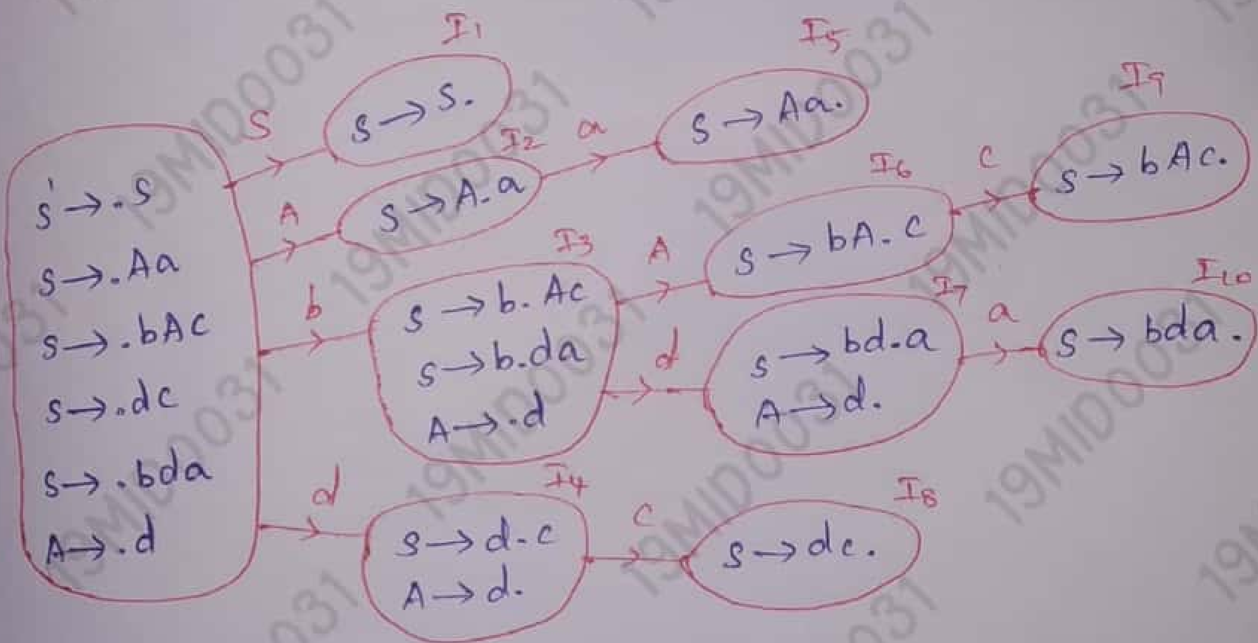
LALR(1) Table

Here there is no SR conflict and RR conflict

∴ The grammar is LALR(1) parsable.

If it is LALR(1) then for sure it is LR(1)

For SLR(1)



Consider 7th state, symbol 'a', Follow(A) = {a, c}, it moves and makes shift reduce conflict, so it is not SLR(1) parser.

2) 1) Find the following grammar is LL(1) ,LR(1)

$S \rightarrow AaAb \mid BbBa$

2) Check whether the following grammar is LR(0), SLR(1),LALR and LR(1)

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

2) $S \rightarrow AaAb \mid BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

FIRST(S) = {a, b}

FIRST(A) = {ε}

FIRST(B) = {ε}

FOLLOW(S) = {\$}

FOLLOW(A) = {a, b}

FOLLOW(B) = {b, a}

Parsing table

	a	b	\$
S	$S \rightarrow AaAb$	$S \rightarrow BbBa$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	
B	$B \rightarrow \varepsilon$	$B \rightarrow \varepsilon$	

No Repetitions

∴ LL(1)

$S \to AaAb \mid BbBa$ LR(0), SLR(1)

$A \to \varepsilon$

$B \to \varepsilon$

1) Augmented Grammar

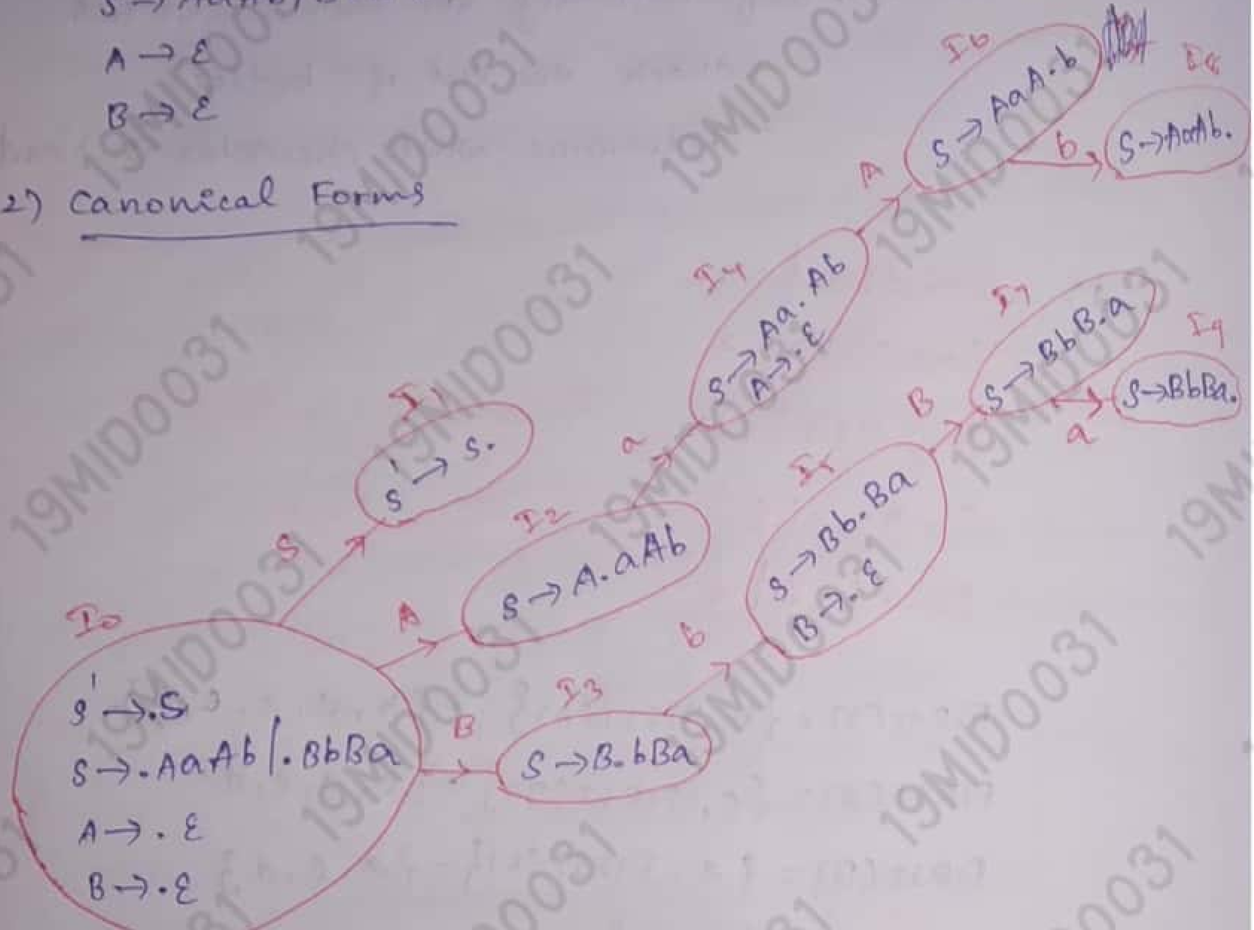
$S' \to S$

$S \to AaAb \mid BbBa$

$A \to \varepsilon$

$B \to \varepsilon$

2) Canonical Forms



3) parsing table

LR(0)

	ACTION			GOTO		
	a	b	$	A	B	S
0				2	3	1
1			accept			
2	S4					
3		S5				
4				6		
5					7	
6		S8				
7	S9					
8	r1	r1	r1			
9	r2	r2	r2			

* There is no SR and RR Conflict. Therefore it is LR(0)

* If it is LR(0), definitely it is SLR(1)