Syllabus

Page No.
Date: / /201

UNIT I

Introduction to compiler:
compilers and translator
Cross Compiler
Phases of compilation and overview
lexical Analysis (scanner):
Regular language, finite Automata, regular expression, scanner generator (LEX, FLEX)

UNIT II

Syntax Analysis:
Syntax Specification of programming language, Design of top-down and bottom-up parsing technique. Design of LL(1) passer. LR passing Design of SLR, CLR, LALR passers. Dealing with ambiguity of Grammar, passer generator (YACC, BISON)

UNIT III

Syntax Directed definitions, implementation SDTS, Intermediate code representation (Postfix Syntax tree, TAC), Intermediate code generation using syntax directed translation schemes for translation of controls structure, declaration, procedure calls, and Array reference

INTRODUCTION To Compiler.

Page No.
Date: / /201

Programming languages are notation for describing computation to people and to machine. The world as we know it depends on programming languages, because all the software running on all the computer was written in some programming language. But before a program can be run, it first must be translated into a form which it can be executed by a computer.

The software systems that do this translation are called COMPILERS.

This COURSE is about how to design and implement compilers. We shall discover that a few basic idea can be used to construct translators for a wide of languages and machines. Besides compilers, the principles and techniques for compiler design are applicable to so many other domains that they are likely to be reused many times in the career of a computer scientis

The study of compiler writing touches upon programming languages, machine architecture, language theory, algorithms, and software engineering. In this preliminary chapter we introduce the different form of language translator, give a high level overview of the structu of a typical compiler, and discuss the trends in programming languages and machine architecture that are shaping compilers. We include some observations o the relationship between compiler design and computer

COURSE OVERVIEW:

Page No.
Date: / /201

We have learnt that any computer system is made of hardware and software. The hardware understands a language, which humans cannot understand. So we write program in high-level language, which is easier for us to understand and remember. These program are then fed into a series of tools and OS components to get the desired code that can be used by the machine. This is known as LANGUAGE PROCESSING SYSTEM, shown in fig.

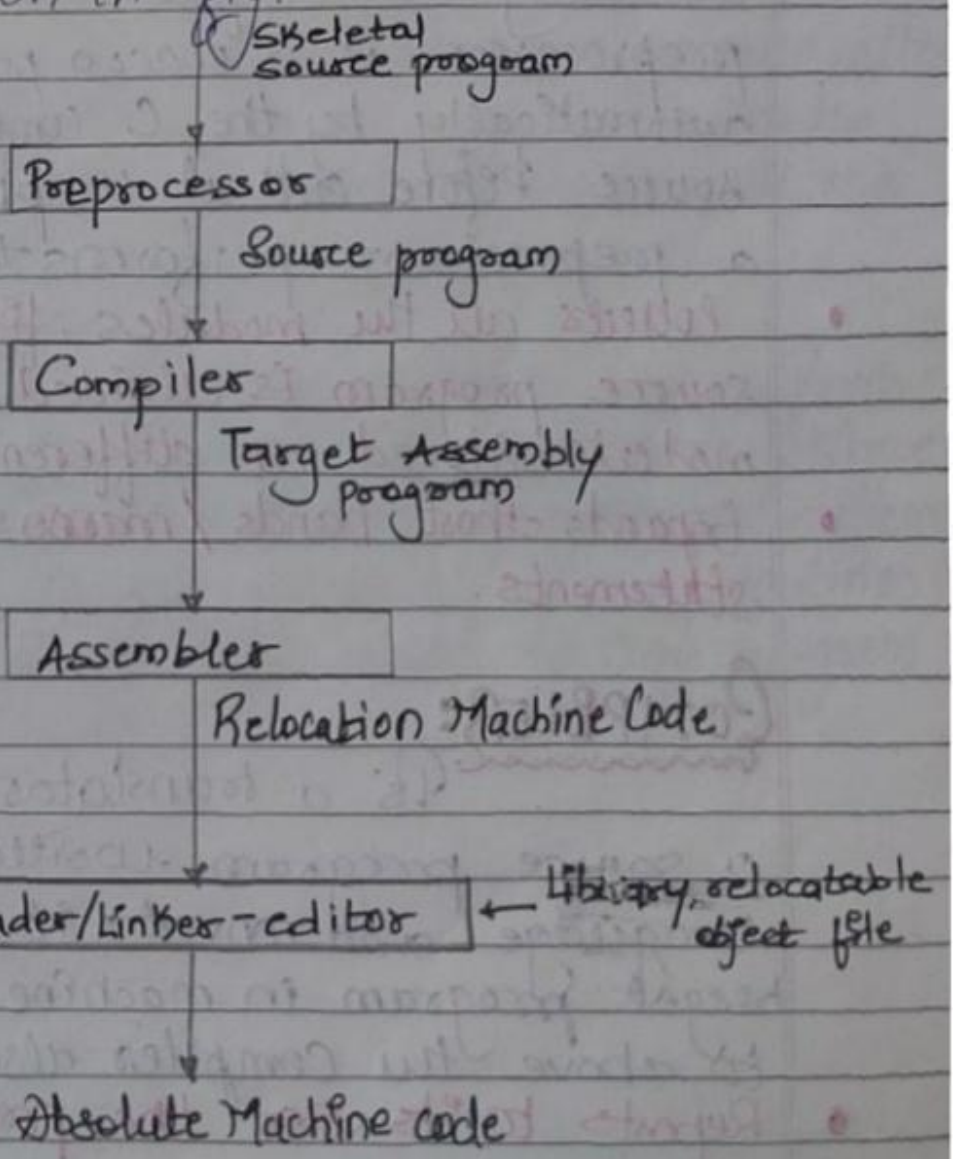


fig. 1.1 Language-processing System.

Language Processing System

Based on the input the translator takes and the output it produces, a language translator can be called as any one of the following.

PREPROCESSOR:

A preprocessor takes the skeletal source program as input and produces an extended version of it, which is the resulted as expanding the macros, manifest constants if any, and including header files etc. For example, the C preprocessor is a macro processor that is used automatically by the C compiler to transform our source before actual compilation. Over and above a preprocessor performs the following activities:

- Collects all the modules, files in case if the source program is divided into different modules stored at different files
- Expands short hands / macros into source language statements.

COMPILER:

Is a translator that takes as input a source program written in high level language and converts it into its equivalent target program in machine language. In addition to above the compiler also

- Reports to its user the presence of errors in the source program
- Facilitates the user in rectifying the errors, and execute the code

ASSEMBLER:

Is a program that takes as input an assembly language program and converts it into its equivalent machine language code.

LOADER / LINKER

This is a program that takes as input A relocatable code and collects the library functions, relocatable object files, and produces its equivalent absolute machine code.

Specifically, the Loading consists of taking the relocatable machine code, altering the relocatable addresses, and placing the altered instructions and data in memory at the proper location

Linking allows us to make a single program from several files of relocatable machine code. These files may have been result of several different compilation, one or more may be library routines provided by the system available to any program that needs them

Date: 26/Jun/2019

Language Processor (LP) [TOUGH COURSE]

UNIT I : (13 M)

Topics.

1) Introduction to compiler
   - Translator
   - Compiler
   - Phases of Compilation & Overview * (IMP * 7M-13M)
   - Cross Compiler * (IMP TOPIC) (7M)

2) Lexical Analysis (Scanner)
   - Regular Language
   - Finite Automata
   - Regular Expression
   - Scanner Generator (LEX/FLEX)* (7-13M IMP)

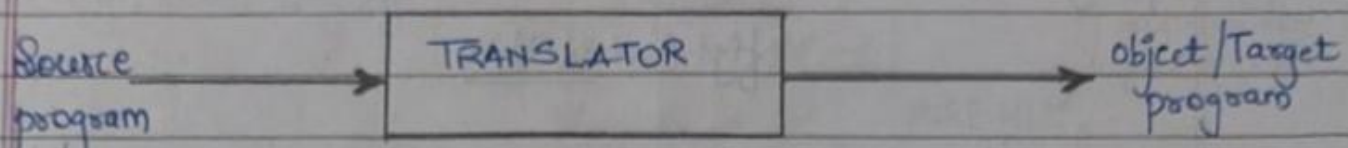
It is a very easy unit.
Total Conceptual.

INTRODUCTION TO COMPILER.

Page No.
Date : / /201

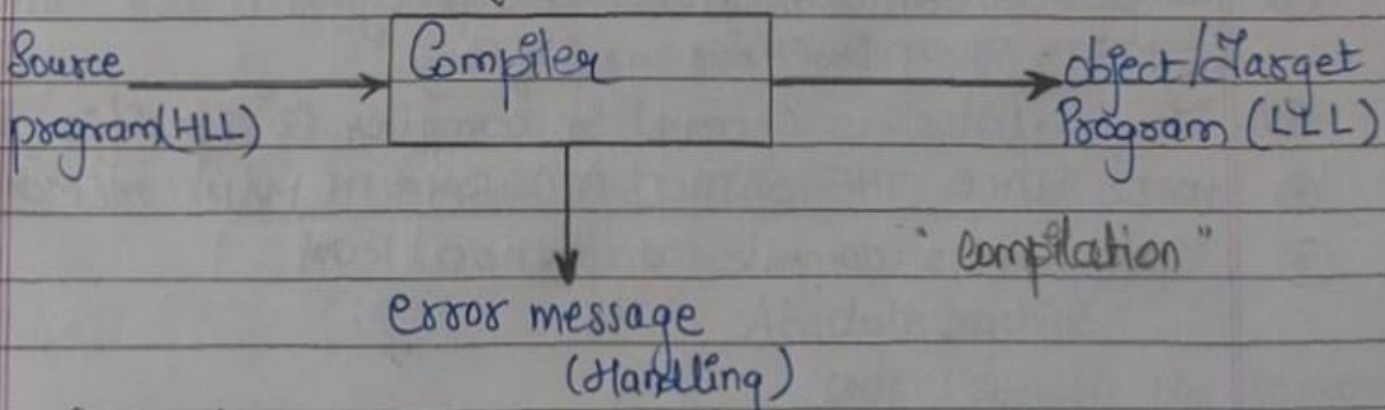
Translator

A program that takes input as program written in one programming language (Source program) and produce output as a program written in another language (object program) is called as Translator



If source program is in high level language such as FORTRAN, COBOL, and object program is in low level language such as Assembly language than such translator is called as COMPILER.

fig. Compiler



Compiler is a program that read the source program written in high level language which is converted into object program written in low level language. Also, an error message is returned if any error(s) are encountered in the source program

A program written in high level language basically executed into TWO steps as shown in the figure below;

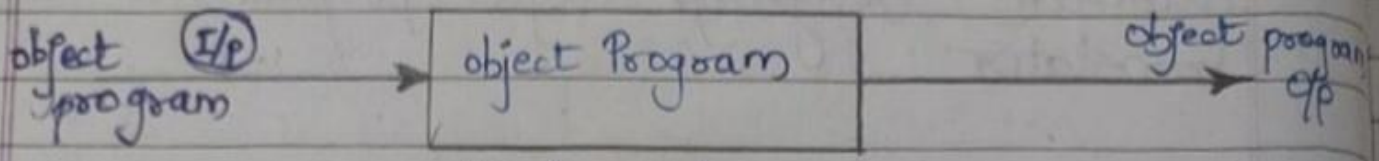


fig: Execution

In first step the source program is compiled ie translated into object program

In second step the resulting source program is loaded into MAIN MEMORY and is executed.

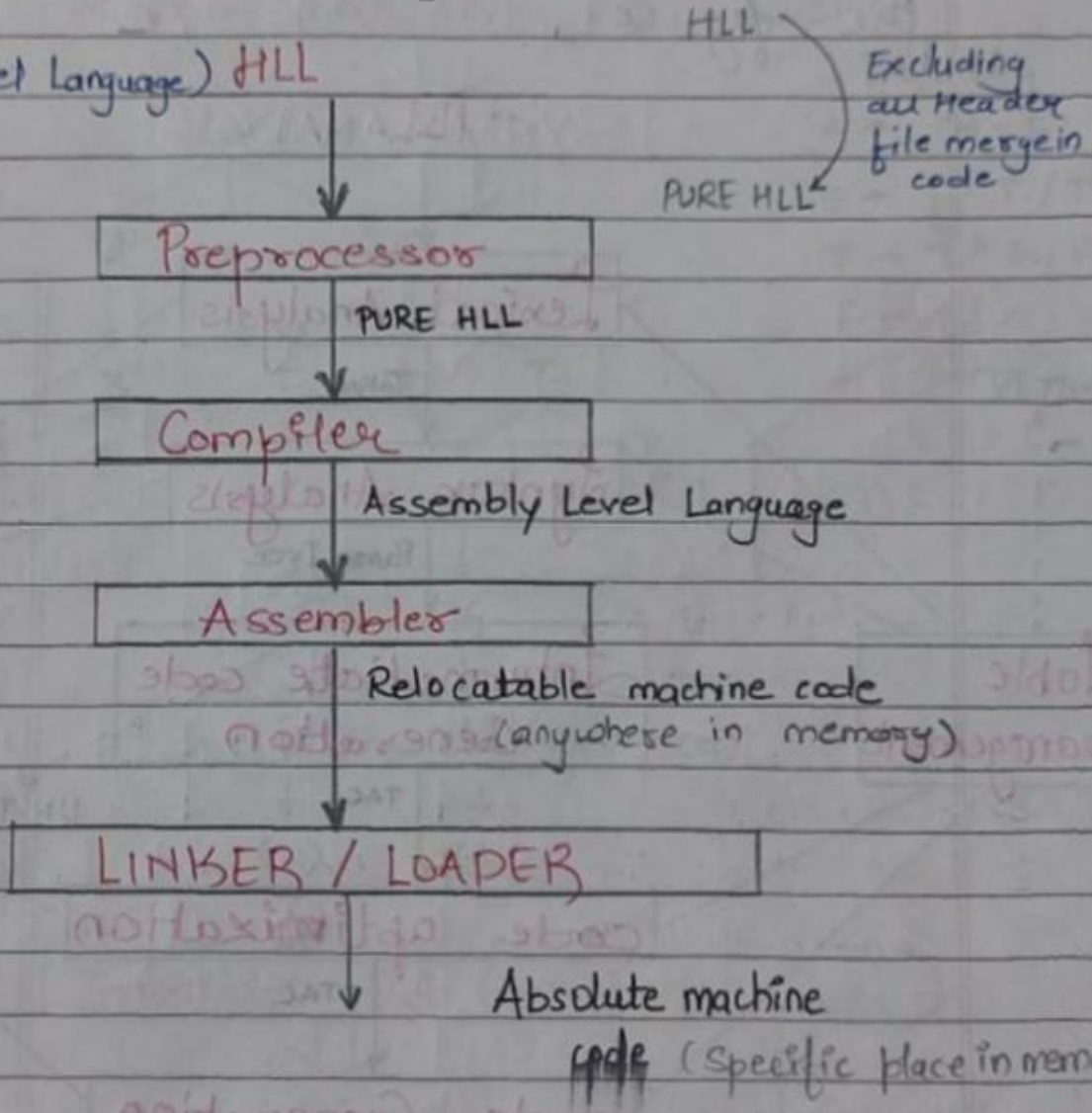
- Interpeter → Read the program line by line
- Compiler → Scans the whole program and than returns Error message if any.
- Translator is General & compiler is specific
- NOTE: PLACE THE OBJECT PROGRAM IN MAIN MEMORY
- Cprogram is compile and than RUN

Phase of Compilation & overview

Page No.
Date : 27 / June 2019

i) Execution Process of program
ii) Phases of compiler

1) Execution Process of program.



Consider the example of C program.

#include <stdio.h> → file inclusion
#define a 10 → macro

Page No.
Date : / /201

2) Phase of Compiler (***)

Defination

The process of compilation is very complex hence it is position into series of sub-process. The collection of this position is called as Phases of compiler. as shown in the figure.

TAC: Three Address code

OTAC: optimize Three Address code

instruction source, destination

computer can only understand parse tree

SUNDAY

Page No.

Date: 30/06/2019

Consider an Example

$x = a + b * c$

⇓

Lexical Analysis

⇓

id = id + id * id

⇓

Syntax Analysis ⟵ CFG

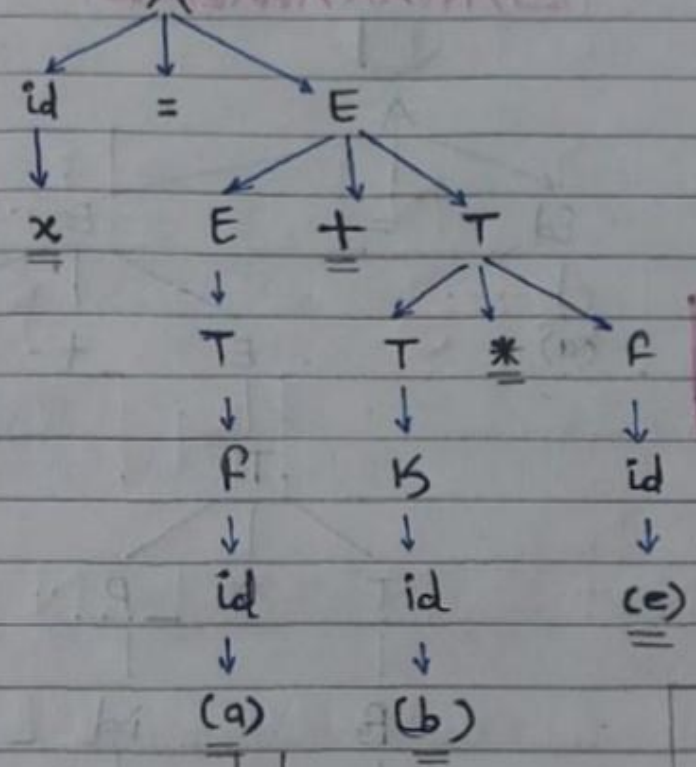
⇓

A → id = E

E → E+T / T

T → T * F / F

F → id



HLL → Lexical Analysis

⇓

Token

Group of character making LOGICAL MEANING

- Keyword
- Identifier
- Operator
- Constant
- Special Symbol

eg: if (a > b)

Keyword op id op id op

Reading is done from LEFT TO RIGHT

⇓

Intermidate Code Generation

$t_1 = b * c$

$t_2 = a + t_1$

$x = t_2$

⇓

Code Optimization

$t_1 = b * c$

$x = a + t_1$

⇓

⋮

⇓

Code Generation

⇓

MOV A, R_0;

MOV B, R_1;

MOV C, R_2;

MUL R_1, R_2;

ADD R_0, R_2;

MOV R_2, x;

8 marks
M 18
Q

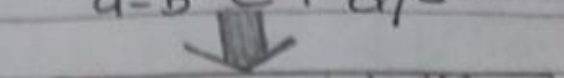
Explain various phases of compiler in brief for the given expression
$a = b * c + d / e$

Monday
Page No.
Date : 1 / 07 / 2019
BOTTOM TO TOP APPROACH

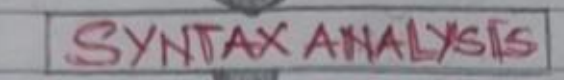
Bottom up approach

Sol^n Consider expression

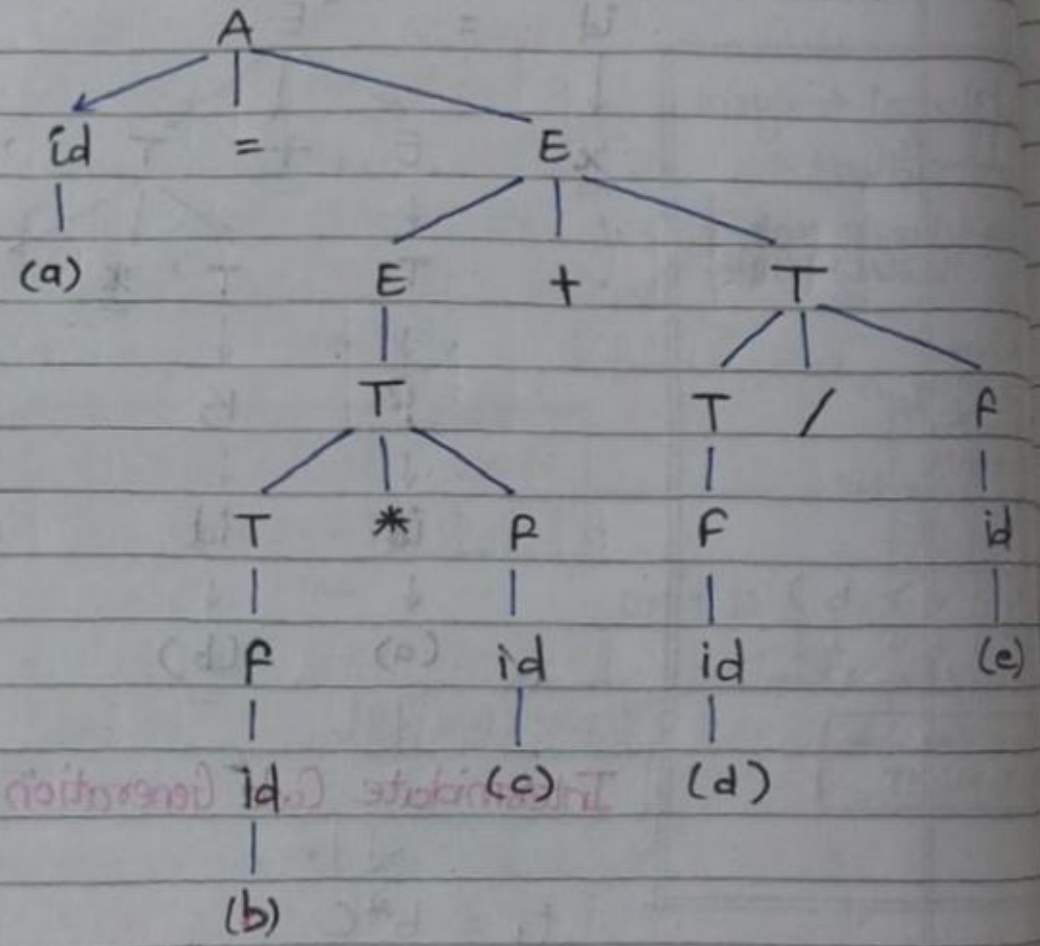
$a = b * c + d / e$

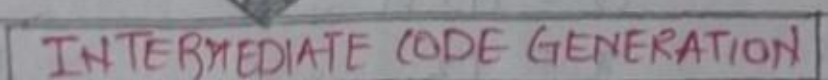


$id = id * id + id / e$



CFG
A → id=E
E → E+T/T
T → T*F/T/F | F
F → id



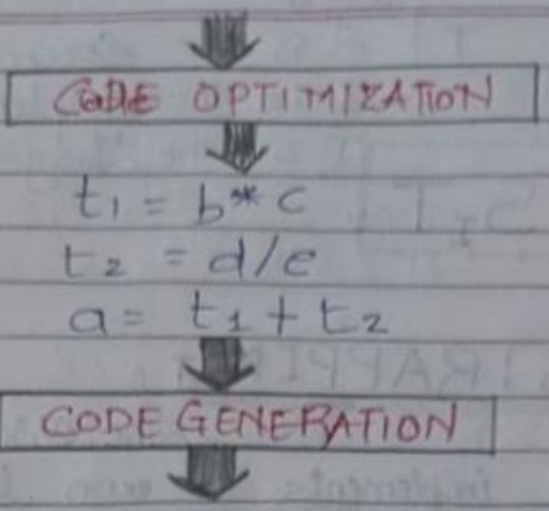


$t_1 = b * c$
$t_2 = d / e$
$t_3 = t_1 + t_2$
$a = t_3$

Left associated operator: If same priority operator in expression give preference to left most operator

Bottom to top approach ↑ $a = \underbrace{b * c} + \underbrace{d / e}$ ↑

Page No.
Date: / /201




```
MOV  b, R0
MOV  c, R1
MOV  d, R2
MOV  e, R3
MUL  R0, R1
DIV  R2, R3
ADD  R1, R3
MOV  R3, a
```

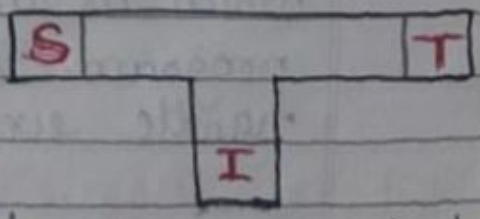
* CROSS COMPILER

Cross compiler is a compiler which runs on one machine and produces object code for another machine

Thus by using Cross compilation technique, platform independancy can be achieved.

Basically a compiler is catagorized by THREE languages

1) Source Language
2) Implementation Language
3) Target Language



Compiler is represented by using T diagram shown above

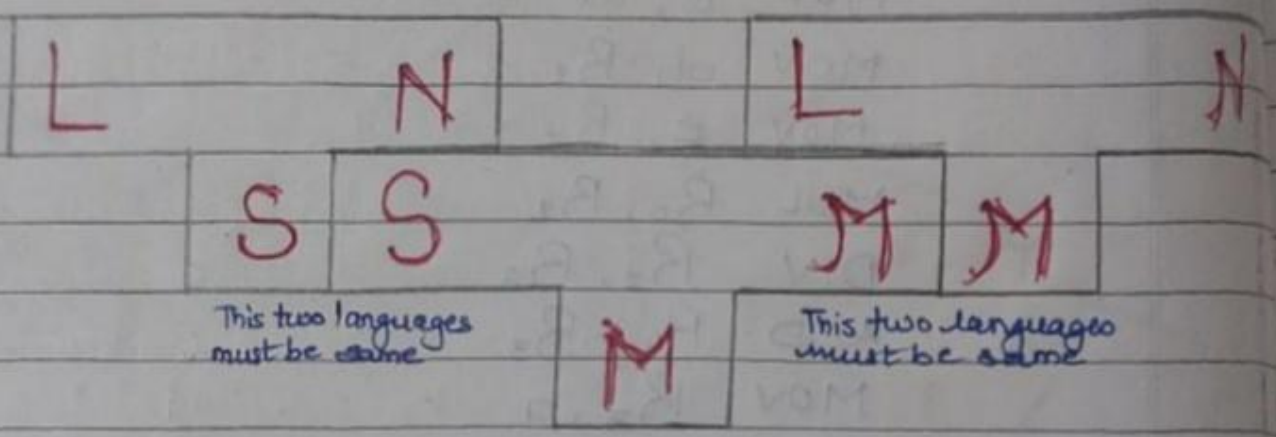
where

S is the source language
I is the Implementation language
T is the Target language

NOTATION $S_I T$

BOOTSTRAPPING

The arrangement in which compiler implements its own language is called as BOOTSTRAPPING, shown in the figure



$$L_S N + S_M M = L_M N$$

for the source language L and the target language N is generated which runs on machine M.

Bootstrapping is a process in which simple language is used to translate more complecated program which in turn may handle for more complicated program. This complicated program can further handle even more complicated program. & so on

2 LEXICAL ANALYSIS. (SCANNER)

English REGULAR LANGUAGE:

Symbol **Symbol:** Any predefine entity (alphabet, digit, symbol)
e.g. $\lambda, +, 0, 1$

Alphabet **Alphabet (Σ):** Set of symbols $\Sigma = \{0, 1\}$
$\Sigma = \{a, b\}$

Word **String (ω):** finite sequence of symbol over given Σ
$\omega = ababa$; $|\omega| = 5$

Language **Language (L):** Collection / set of string over given Σ

e.g. $\Sigma = \{a, b\}$

$|\Sigma|^n = 2^2 = 4$
$|\Sigma|^n = 2^3 = 8$

L_1 = set of all strings of Length '2' over Σ
$L_1 = \{aa, ab, ba, aa\}$

L_2 = set of all strings of length '3' over Σ
$L_2 = \{a, b\}\{aa, ab, ba, bb\}$ $\because L_1.L_1$
$L_2 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

L_3 = set of all strings starting with 'a'
$L_3 = \{a, aa, ab, aaa, aab, abb, \ldots\ldots\}$

NOTE: Language may be finite or infinite

Because of language being infinite you are learning TOC & language processor

Page No.
Date: / /201

Regular Language:

The language accepted by the finite automata is called as Regular language

Operations on a language(s):

There are following operation that can be performed on a language

1. CONCATINATION:

$$L_1 \cdot L_2 = \{uv \mid u \in L_1 \ \& \ v \in L_2\}$$

eg: $L_1 = \{a, b\}$

$L_2 = \{aa, ab, ba, bb\}$

$L_1 \cdot L_2 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

($u \in L_1$, $v \in L_2$)

2. CLOSURE OPERATION

i) Star | Kleen closure (L^*):

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots\ldots$

$= \{\epsilon\} \cup \{a, b\} \cup \{aa, ab, ba, bb\} \cup \ldots\ldots$

$= \{\epsilon, a, b, aa, ab, ba, bb, \ldots\ldots\ldots\}$

ii) Positive | Union closure (L^+):

$L^+ = L^1 \cup L^2 \cup L^3 \ldots\ldots\ldots$

$= \{a, b\}, \{aa, ab, ba, bb\}, \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

$= \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb \ldots\}$

WHERE $L = \{a, b\}$

$L^0 = \{\phi\} = \{\epsilon\}$ where $|\omega| = 0$

$L^1 = \{a, b\}$

$(L_1 \cdot L_1) = L^2 = \{a \cdot b\} \cdot \{a \cdot b\} = \{aa, ab, ba, bb\}$

$(L_1 \cdot L_2) = L^3 = \{a \cdot b\} \cdot \{aa, ab, ba, bb\}$

$= \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

FINITE AUTOMATA.

Page No.
Date : 03/07/2019

→ It is a basic mathematical or computational model which is use to recogized the string

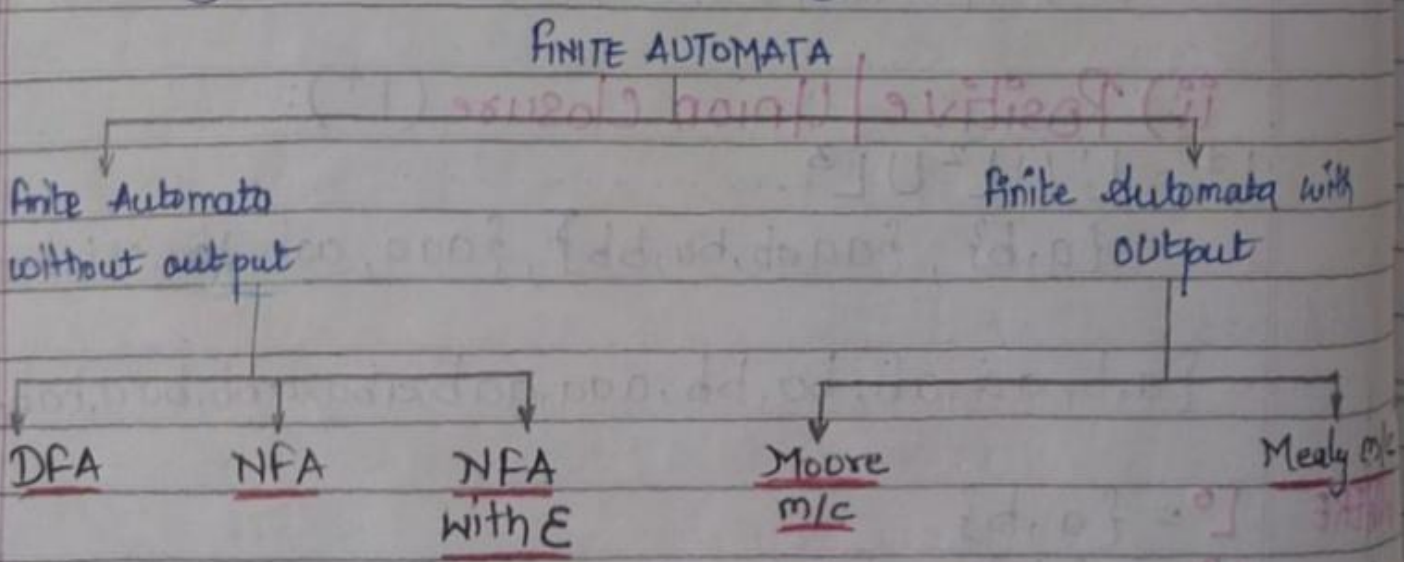
→ In finite automata number of possible states and input alphabet(s), both are finite

→ It is a finite representation of finite or infinite language(s)

→ finite automata contain limited amount of memory, hence it can not perform any mathematical operation like addition, substraction, multiplication, etc

TYPE OF FINITE AUTOMATA.

finite automata are classified into following types as shown in figure



FINITE AUTOMATA WITHOUT OUTPUT

Deterministic finite Automata (DFA).

In DFA, each and every state should contain exactly one out degree for each input alphabet symbol.

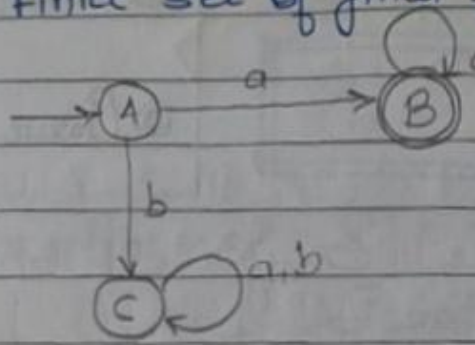
It is represented by using 5 tuples.

$M = \langle Q, \Sigma, \delta, q_0, f \rangle$

where

Q : finite set of states
Σ : finite set of input symbol
δ : Transition/Mapping function $[\delta: Q \times \Sigma \to Q]$
q_0 : Initial state
f : finite set of final symbol

e.g.



$Q = \{A, B, C\}$
$\Sigma = \{a, b\}$
$q_0 = A$
$f = \{B\}$

$\delta = Q \times \Sigma \to Q$

$\delta = \{A, B, C\} \times \{a, b\} \longrightarrow \{A, B, C\}$

Q\Σ	a	b
A	B	C
B	B	B
C	C	C

* REGULAR EXPRESSION.

Page No.
Date: 7 / 07 / 2019

The language accepted by the finite automata (i.e. Regular language) is represented using simple expression is called as REGULAR EXPRESSION.

Regular Language $\xrightarrow{\text{Represent}}$ Regular Expression

Regular Expression

operand

① $\phi = \{\}$

② $\epsilon = \{\epsilon\}$

③ $a = \{a\}$, $a \in \Sigma$

operator

① UNION $(+, |)$, or

② Concatination $(\cdot)$

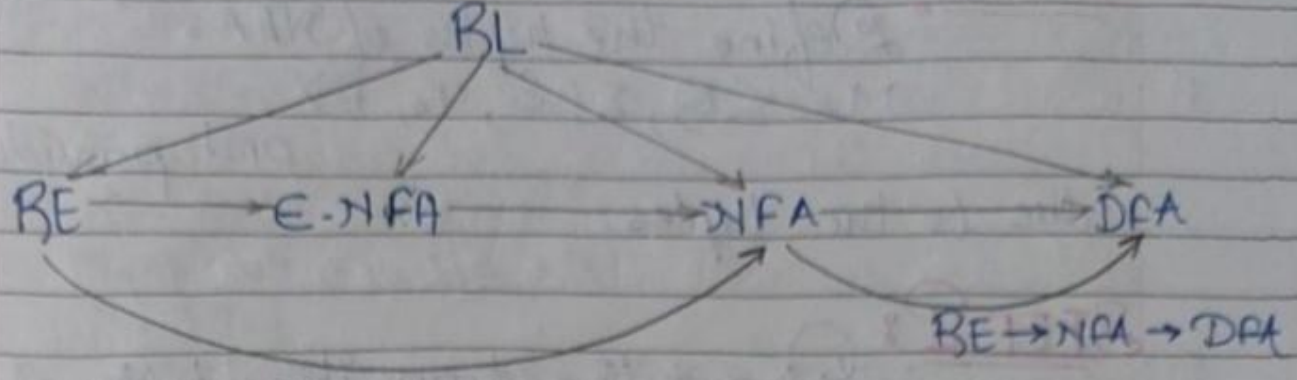
③ Closure (R^*, R^+)

Regular Expression (R.E.)

For example

Regular Expression	Regular Language
ϕ	$\{\}$
ϵ	$\{\epsilon\}$
a	$\{a\}$
b	$\{b\}$
$a+b$	$\{a, b\}$
$a.b$	$\{a.b\}$
a^*	$\{\epsilon, a, aa, aaa, aaaa, \ldots\ldots\}$
a^+	$\{a, aa, aaa, aaaa, \ldots\ldots\}$
$(a.b)^*$	$\{\epsilon, ab, abab, ababab, \ldots\ldots\}$
$(a+b)^*$	$\{\epsilon, a, b, aa, bb, ab, ba, \ldots\ldots\}$

CONVERSION OF REGULAR EXPRESSION TO DFA.



STEP I Initial state

There is no change in initial state of NFA and DFA it means that initial state of NFA and DFA is same

$$q_0' = q_0$$

STEP II Construction of δ' [Transition function of DFA]

Starts the construction of δ' from initial state and continue the process for every new state which appears in the input column and terminate the process whenever no new state appears in a i/p column.

STEP III Final state

Every subset which contain final state of NFA is final state in DFA.

STEPS FOR SOLVING THE PROBLEM.

Date : / /201

STEP I: Define the tuples of NFA.

$M = \langle Q, \Sigma, \delta, q_0, F \rangle$

and provide the value to the tuples.

STEP II: Draw the transition table of NFA

STEP III: Define the tuples of DFA i.e.

$$M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$$

where

Q' = finite set of all state in DFA
Σ = finite set of input symbol
δ' = Transition function of DFA
q_0' = Initial state of DFA
F' = Set of all final state of DFA.

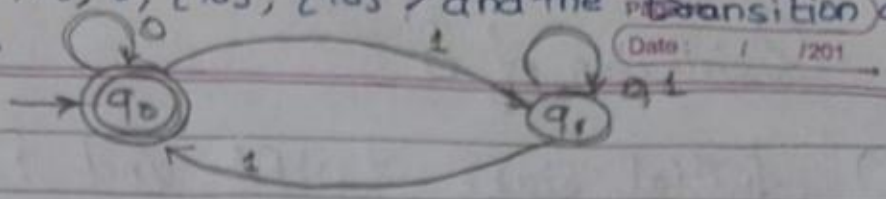
STEP IV: Construct the transition table of DFA by exploiting the logic

STEP V: Provides the value to the tuples of DFA

$Q' \times \Sigma \rightarrow Q'$

STEP VI: Draw the transition Diagram of DFA.

$M = \langle \{q_0, q_1\}, \{0,1\}, \delta, \{q_0\}, \{q_0\} \rangle$ and the transition diagram is given below.



Ans. Let $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ are the tuples of NFA.

$Q = \{q_0, q_1\}$

$\Sigma = \{0, 1\}$

$\delta : Q \times \Sigma \longrightarrow 2^Q$

$q_0 = \{q_0\}$

$F = \{q_0\}$

Transition Table of NFA

$Q \backslash \Sigma$	0	1
$\rightarrow (q_0)$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_1\}$

Let $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ are the tuples of DFA. where

Q' = Finite set of all states in D.F.A.

Σ = Finite set of input symbol

δ' = Transistion function of D.F.A.

q_0' = Initial state of D.F.A.

F' = Set of all final state in D.F.A.

Transition Table of D.F.A.

$Q \backslash \Sigma$	0	1
$\rightarrow [q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0 q_1]$
$[q_0 q_1]$	$[q_0 q_1]$	$[q_0 q_1]$

construction of δ'

i) Initial state of NFA and DFA is equal i.e $q_0 = q_0'$

ii) Start the construction of δ' from initial state & continue the process for every new state which appear in the input table and terminate the process whenever no new state appear in the input table

iii) Every subset which contain final state of NFA is the final state in D.F.A.

$$\delta'([q_0], 0) = \delta(\{q_0\}, 0)$$
$$= [q_0]$$

$$\delta'([q_0], 1) = \delta(\{q_0\}, 1)$$
$$= [q_1]$$

$$\delta'([q_1], 0) = \delta(\{q_1\}, 0)$$
$$= [q_1]$$

$$\delta'([q_1], 1) = \delta(\{q_1\}, 1)$$
$$= \{q_0\} \cup \{q_1\}$$
$$= [q_0 q_1]$$

$$\delta'([q_0 q_1], 0) = \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0)$$
$$= [q_0] \cup [q_1]$$
$$= [q_0 q_1]$$

$$\delta'([q_0 q_1], 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1)$$
$$= [q_1] \cup [q_0 q_1]$$
$$= [q_0 q_1]$$

$Q' = \{[q_0], [q_1], [q_0q_1]\}$

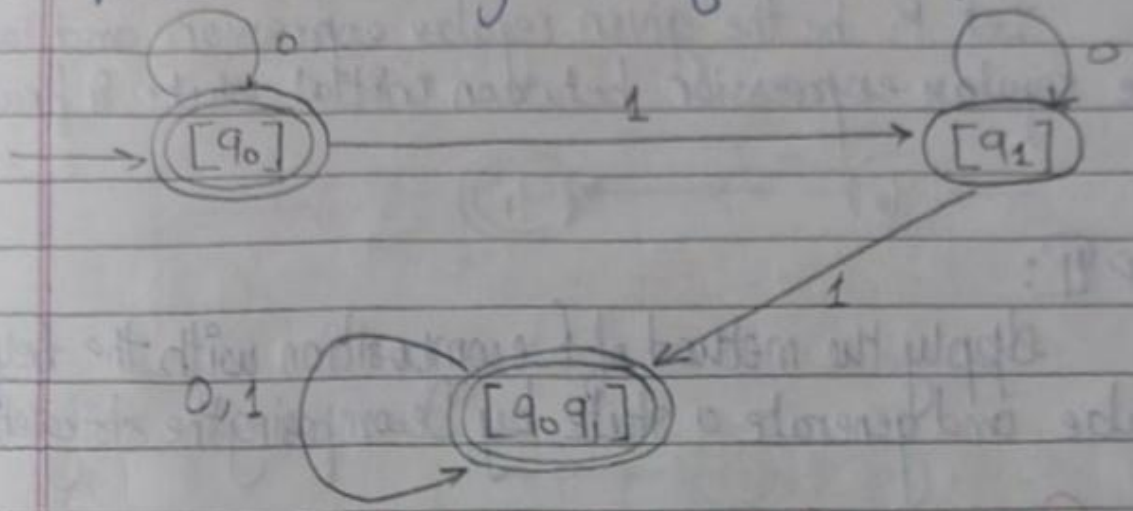
$\Sigma = \{0, 1\}$

$\delta' = Q' \times \Sigma \longrightarrow Q'$

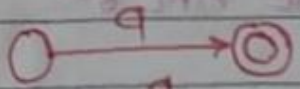
$q_0' = [q_0]$

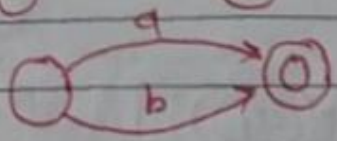
$F' = \{[q_0], [q_0q_1]\}$

Transition Diagram of D.F.A.

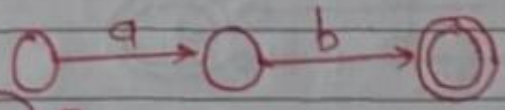


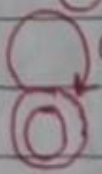
PROBLEM 01

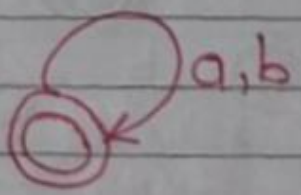
$a =$ 

$a+b =$ 

Using METHOD OF DECOMPOSITION

$a.b \,/\, ab =$ 

$a^* =$ 

$(a+b)^* =$ 

METHOD OF DECOMPOSITION

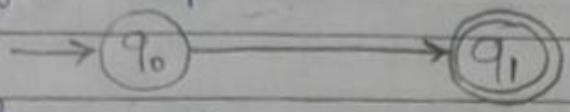
Date : / /201

1. By using the method of Decomposition we are converting the given regular expression into NFA with ε - transition

Step for Solving the problem.

STEP I:

Let R be the given regular expression and take the entire regular expression between initial state & final state.

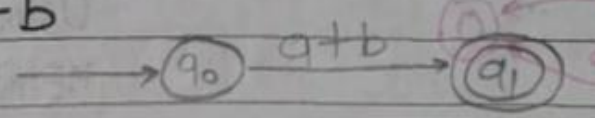


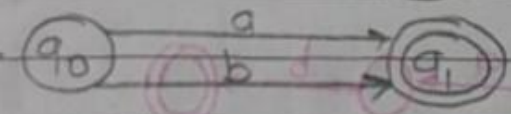
STEP II:

Apply the method of Decomposition with the help of formulae and generate a state by Decomposing the string given.

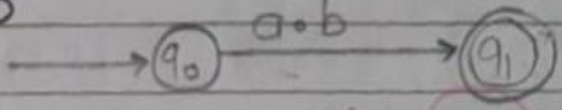
SOME IMPORTANT FORMULAE ARE :→

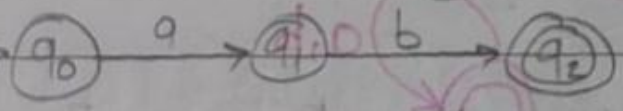
i) R = a+b



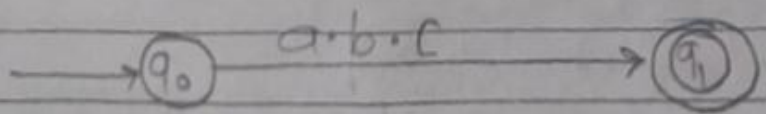


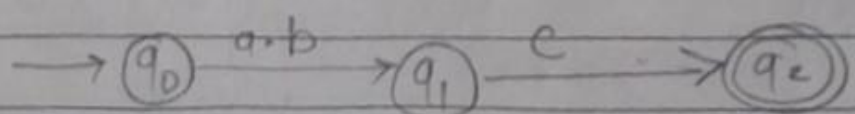
ii) R = a·b

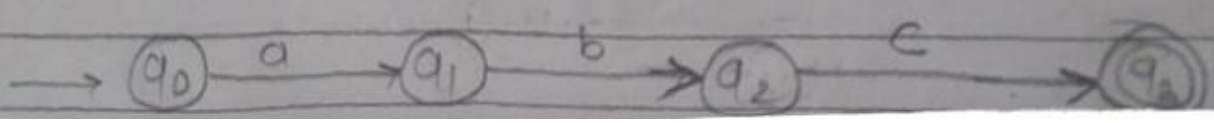




Ex. R = a·b·c

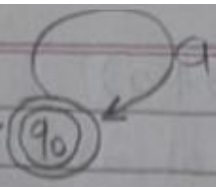




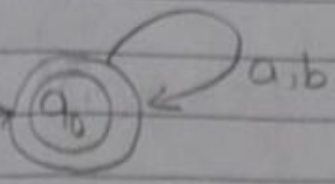


Date : / /201

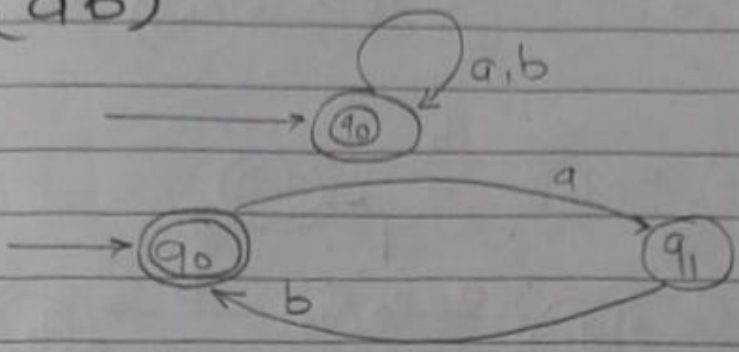
a) $R = a^*$



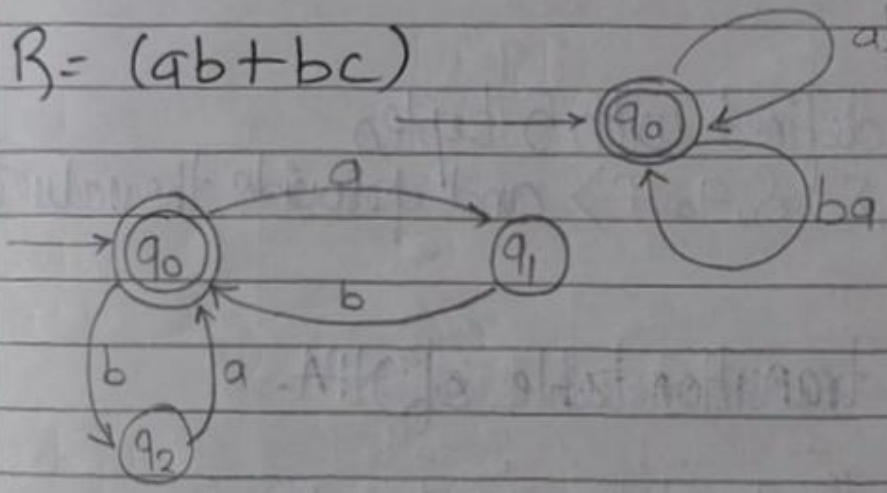
b) $R = (a+b)^*$



c) $R = (ab)^*$

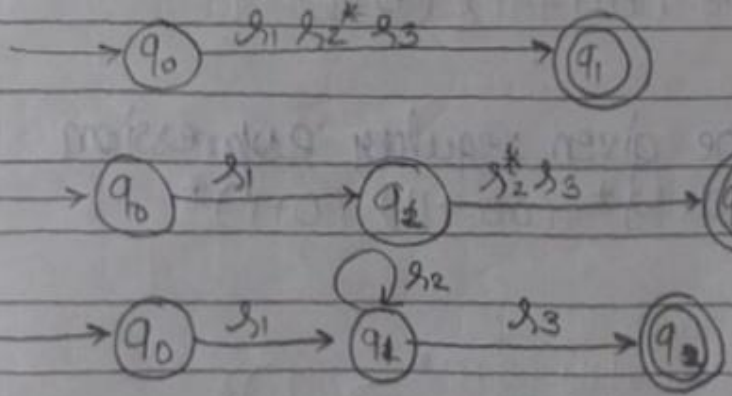


d) $R = (ab+bc)$

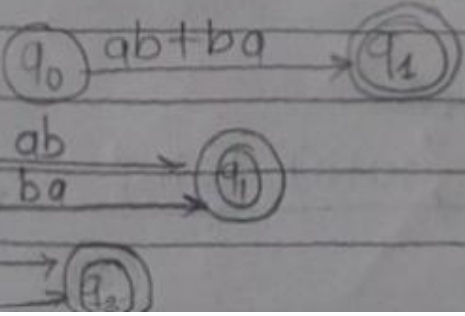


Ex.

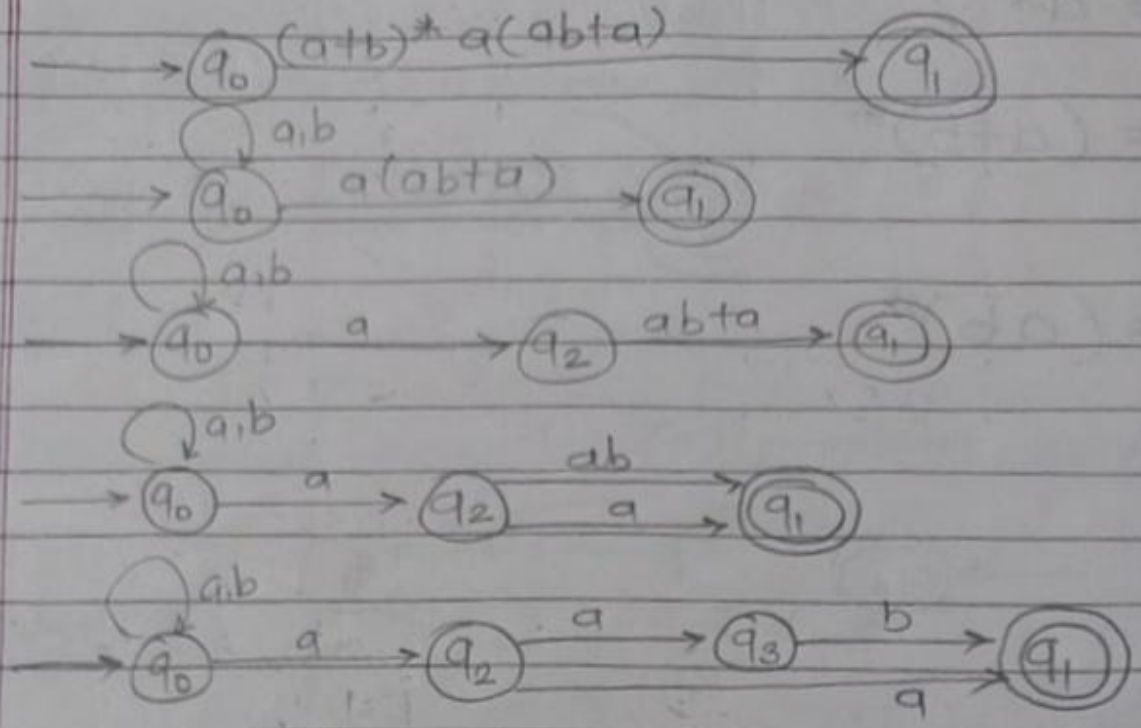
1) $R = r_1 r_2^* r_3$



2) $R = ab+ba$



3) $R = (a+b)^* a(ab+a)$



STEP III

NFA is defined with 5 tuples
$\gamma 1 = \langle Q, \Sigma, \delta, q_0, f \rangle$ and provide the value to tuples

STEP IV

Draw the transition table of NFA.

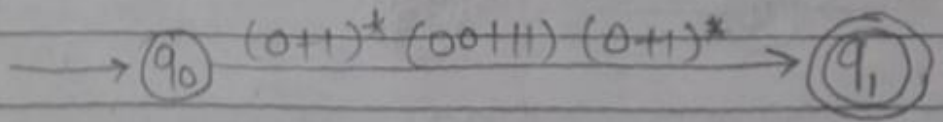
Q1. Construct a finite automata for the given regular expression.

$(0+1)^* (00+11) (0+1)^*$

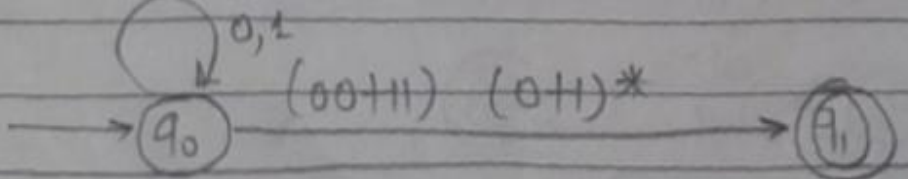
Sol Let R be the given regular expression

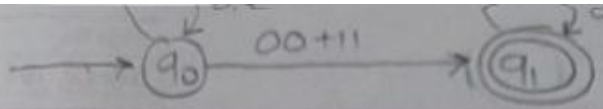
$R = (0+1)^* (00+11) (0+1)^*$

STEP I:

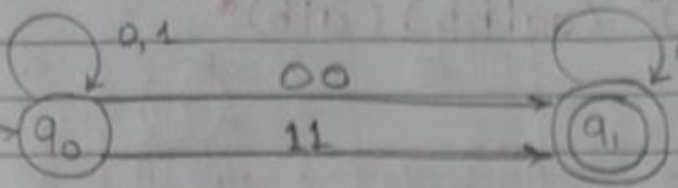


STEP II:

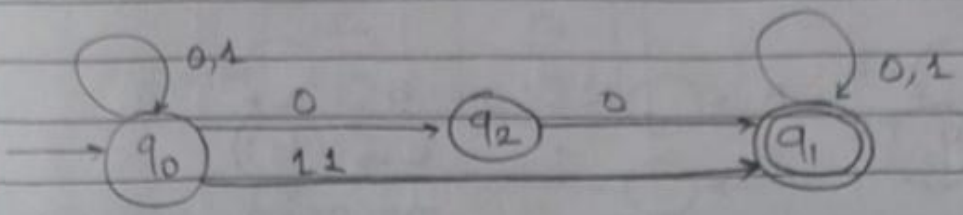




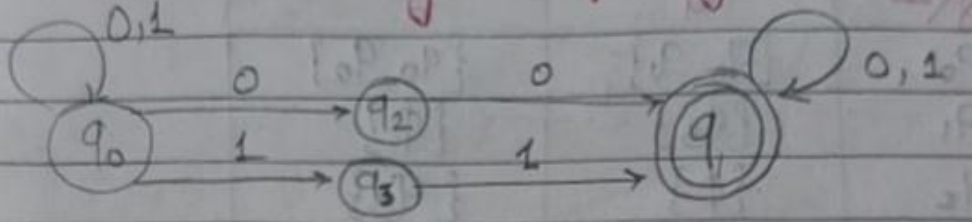
Step II



Step IV



Step VI final transition diagram of the given R.E.



NFA is defined with 5 tuples

$M = \langle Q, \Sigma, \delta, q_0, f \rangle$

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$\delta = Q \times \Sigma \longrightarrow 2^Q$

$q_0 = \{q_0\}$

$f = \{q_1\}$

Transition table of NFA.

$Q \backslash \Sigma$	0	1
$\rightarrow q_0$	$[q_0 q_2]$	$[q_0 q_3]$
(q_1)	$[q_1]$	$[q_1]$
q_2	$[q_1]$	ϕ
q_3	ϕ	$[q_3]$

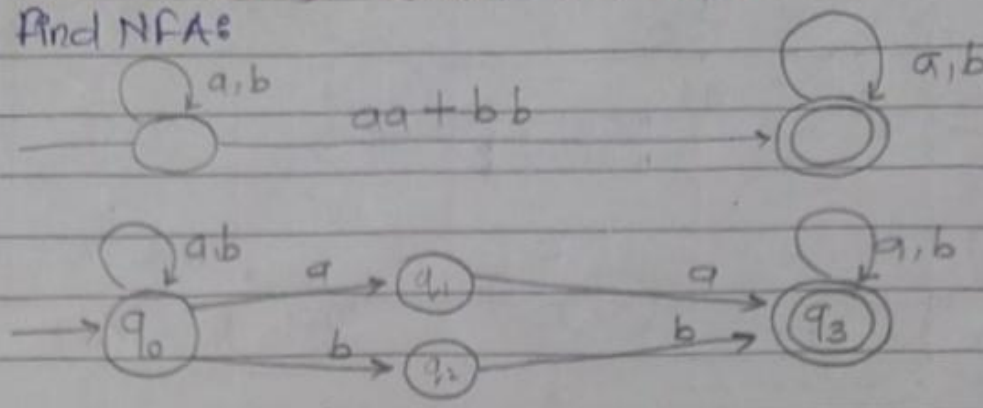
with epsilon

HW. $a(ab)^* a + ab^* ab$

HW $10 + (0+11)0^* 1$

6m S-18 * Construct minimized DFA for the regular expression $(a|b)^*(aa|bb)(a|b)^*$

Sol:n Find NFA:



Transition Table

Q\Σ	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	$\{\phi\}$
q_2	$\{\phi\}$	$\{q_3\}$
((q_3))	$\{q_3\}$	$\{q_3\}$

Transition Table

	Q\Σ	a	b
A	→ $[q_0]$	$[q_0 q_1]$	$[q_0 q_2]$
B	$[q_0 q_1]$	$[q_0 q_1 q_3]$	$[q_0 q_2]$
C	$[q_0 q_2]$	$[q_0 q_1]$	$[q_0 q_2 q_3]$
D	($[q_0 q_1 q_3]$)	$[q_0 q_1 q_3]$	$[q_0 q_2 q_3]$
E	($[q_0 q_2 q_3]$)	$[q_0 q_1 q_3]$	$[q_0 q_2 q_3]$

Q\Σ	a	b
→ A	B	C
B	D	C
C	B	E
(D)	D	E
(E)	D	E

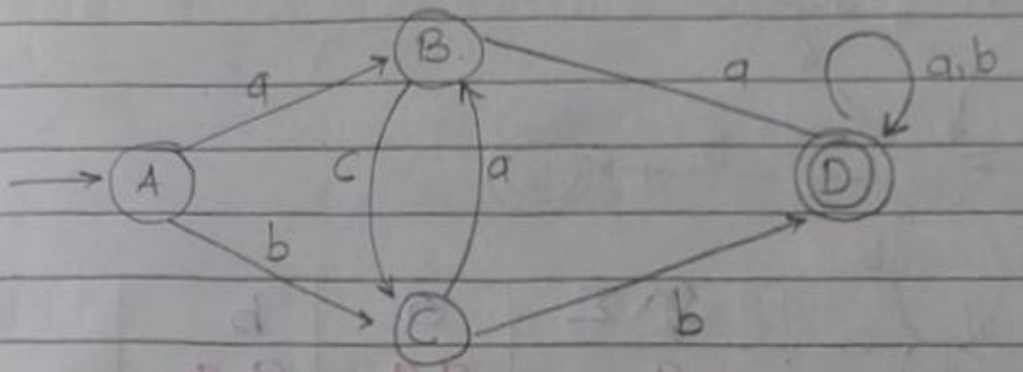
minimization

Page No.
Date : / /201

D=E (also replace occurance of E with D)

q\Σ	a	b
→A	B	C
B	D	C
C	B	D
(D)	D	D

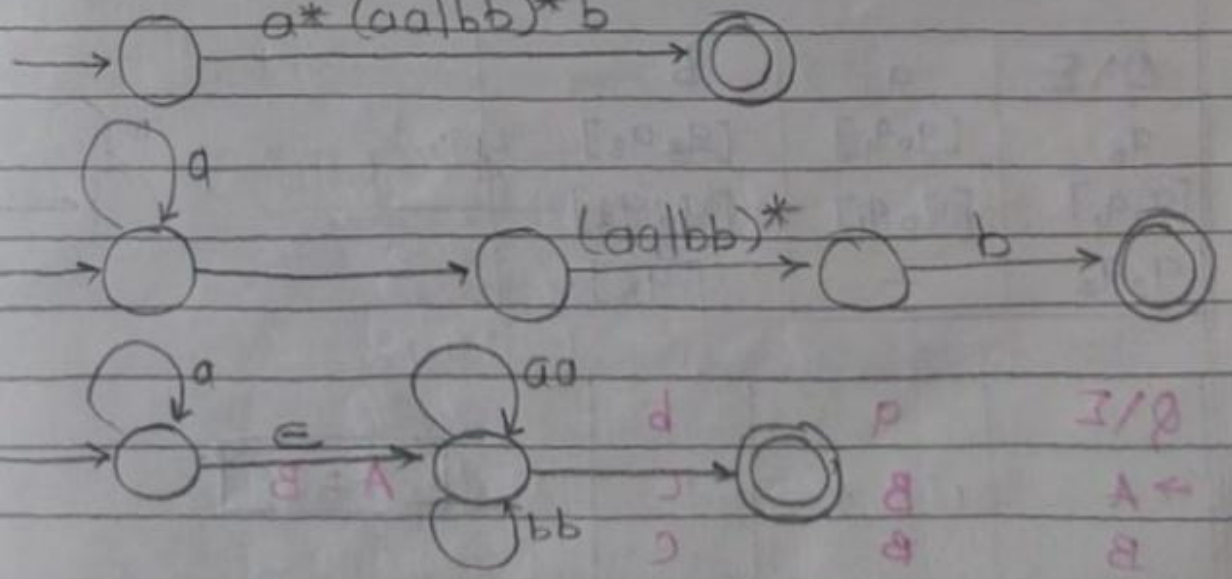
Transition Diagram



HM (a+b)* aba (a+b)*

* a* (aa|bb)* b

dus



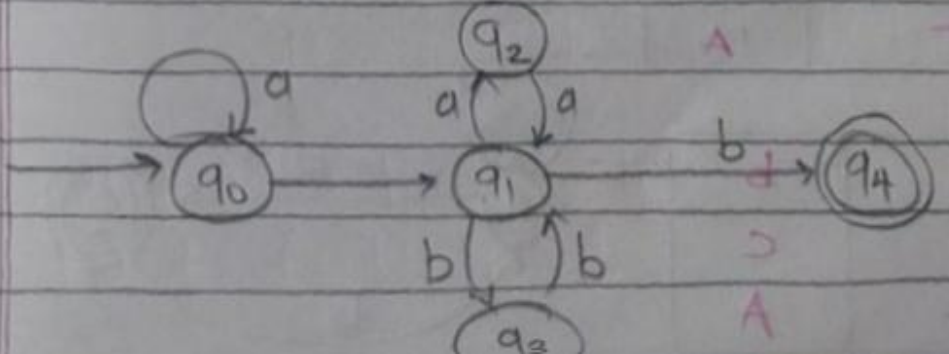
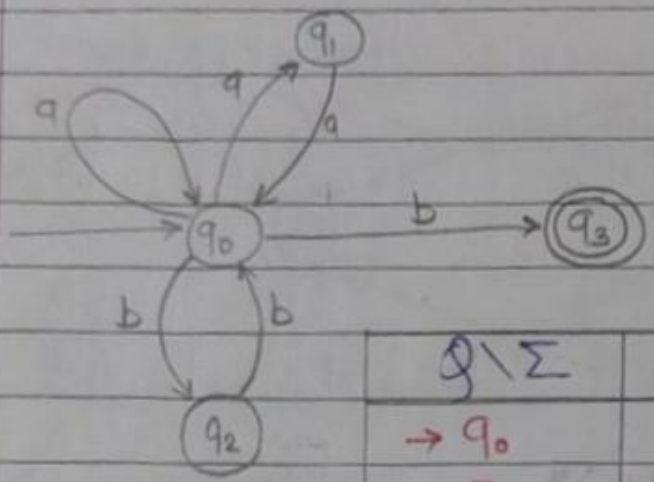


Table Transition of ϵ- NFA.

Q\Σ	a	b	ϵ
→q_0	$\{q_0\}$	–	$\{q_1\}$
q_1	$\{q_2\}$	$\{q_3, q_2\}$	–
q_2	$\{q_1\}$	–	–
q_3	–	$\{q_1\}$	–
(q_4)	–	–	–



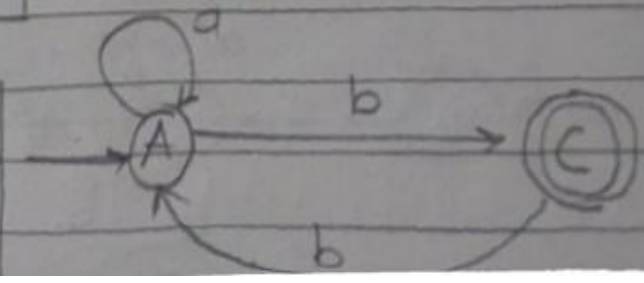
Q\Σ	a	b
→q_0	q_0q_1	q_2q_3
q_1	q_0	ϕ
q_2	ϕ	q_0
(q_3)	ϕ	ϕ

	Q\Σ	a	b
A	q_0	$[q_0q_1]$	$[q_2q_3]$
B	$[q_0q_1]$	$[q_0q_1]$	$[q_2q_3]$
C	q_2q_3	–	$[q_3]$

Q\Σ	a	b
→A	B	C
B	B	C
*C	–	A

A = B

Q\Σ	a	b
→A	A	C
		A



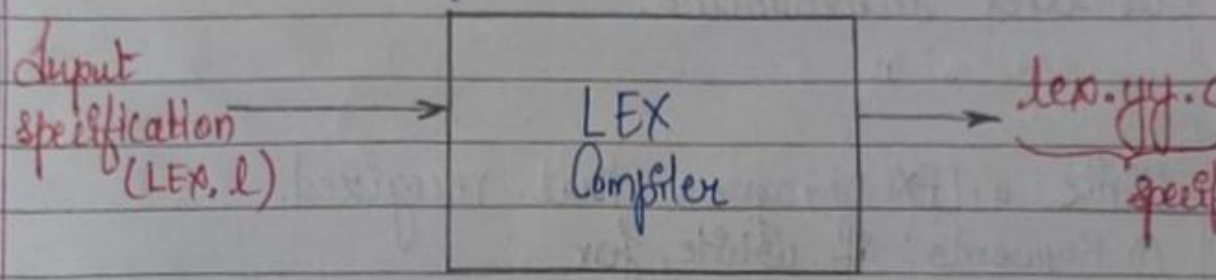
* SCANNER GENERATOR (LEX/FLEX)

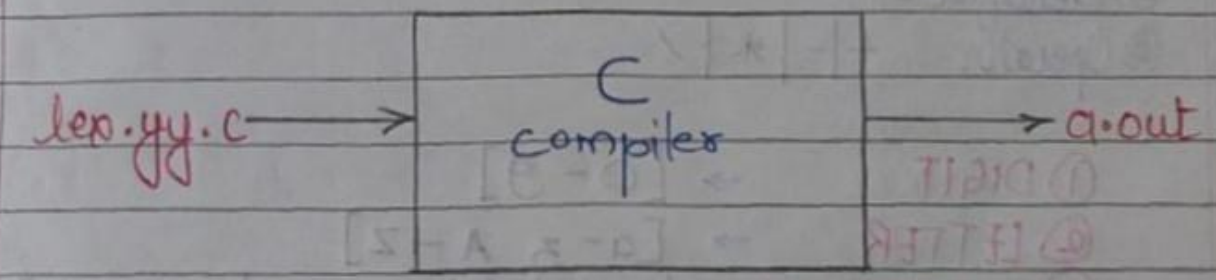
Page No.
Date : / /201

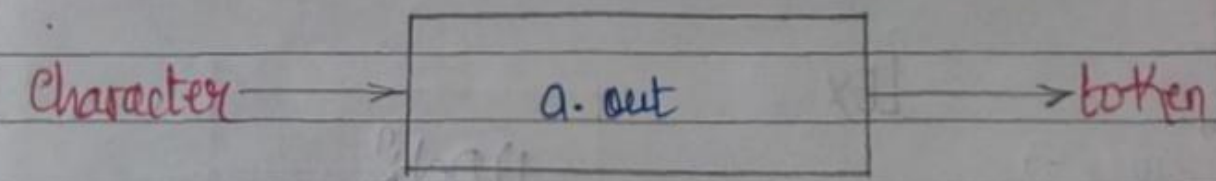
Compilation Question → **LEX:**

LEX is a Compiler writting tool which is used to generate LEXICAL ANALYSIZER or SCANNER from the description of tokens or programming language to be implemented.

The description requried as Regular Expression shown in the figure







INPUT SPECIFICATION

Lex program consist of 3 part

→ Defination
%. %.
→ Rules
%. %.
→ auxiliary Procedure

① Defination has format.

Name _____ Regular Expression

② Rules has format

Regular Expression {C Code}

③ auxiliary Procedure

It is a procedure or function or user subroutine.

TOE

July 9, 19

Write a LEX program that recogized.

① Keywords if, while, for

② Identifier

③ Operator +|-|*|/

① DIGIT → [0-9]

② LETTER → [a-z A-Z]

③ IDENTIFIER → {LETTER} ([LETTER]|[DIGIT])*

LEX

Name RE ←

RE {C Code} ←

sum() ←

1) Def:
%%
2) Rules
%%
3) auxiliary procedure

Page No.
Date : / /201

Program:

```
%{
  #include <stdio.h>
%}

LETTER [a-z A-Z]
DIGIT [0-9]
ID {LETTER} ({LETTER} | {DIGIT})*

%%
if {
  printf("Recognized KEYWORD: %s \n", yytex);
}

while {
  printf("Recognized KEYWORD: %s \n", yytex);
}

for {
  printf("Recognized KEYWORD: %s \n", yytex);
}

{LETTER} ({LETTER} | {DIGIT})*
  {
  printf("Recognized ID: %s \n", yytex);
}

+ {
  printf("Recognized OPERATOR : %s \n", yytex);
}

- {
  printf("Recognized OPERATOR : %s \n", yytex);
}
```

```
printf("Recognized OPERATOR: %s \n", yytex);
}
/{
printf("Recognized OPERATOR: %s \n", yytex);
}
%%
main()
{
yylex();
```

Content of Data file

```
if
while
for
3am
xyz
+
-
*
/
```

output

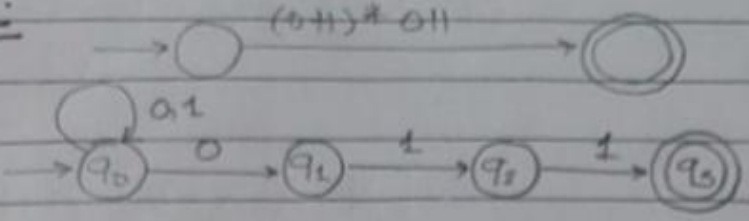
```
Recognized KEYWORD : if
Recognized KEYWORD : while
Recognized KEYWORD : for
Recognized IDENTIFIER: 3am
Recognized IDENTIFIER: xyz
Recognized OPERATOR: +
Recognized OPERATOR: -
Recognized OPERATOR : *
Recognized OPERATOR : /
```

Q Construct DFA for the following regular expression

(0+1)* 011

Page No. Wednesday
Date: 10/09/2019

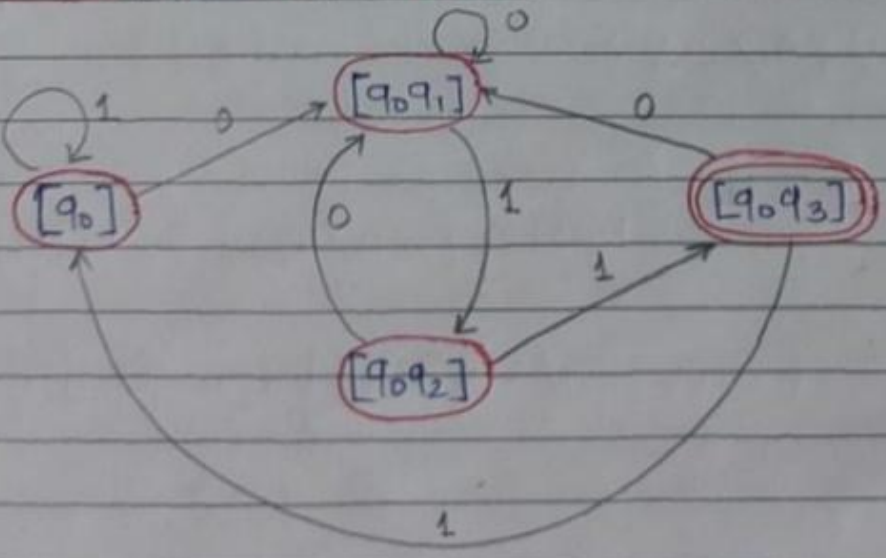
Ans **Step 1:**



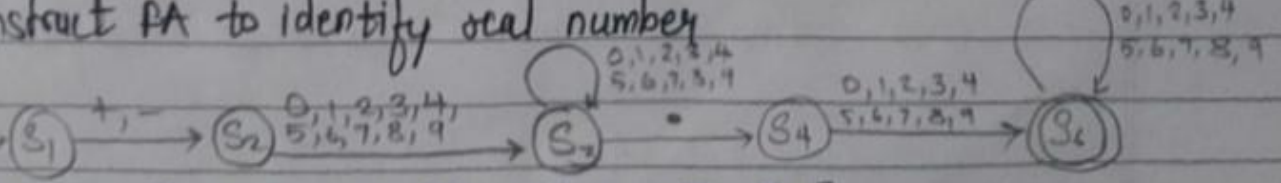
Transition Table

Q\Σ	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
q_2	ϕ	$\{q_3\}$
(q_3)	ϕ	ϕ

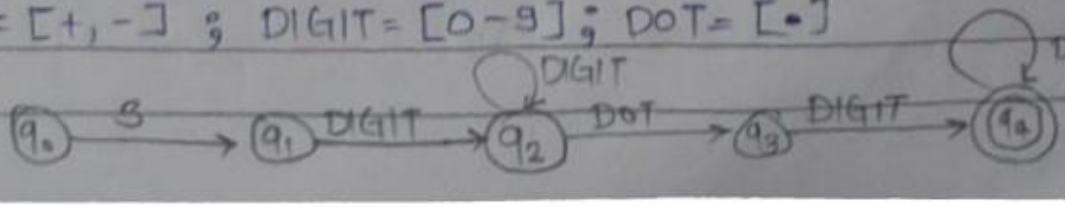
Q\Σ	0	1
→$[q_0]$ A	$[q_0q_1]$ B	$[q_0]$ A
$[q_0q_1]$ B	$[q_0q_1]$ B	$[q_0q_2]$ C
$[q_0q_2]$ C	$[q_0q_1]$ B	$[q_0q_3]$ D
($[q_0q_3]$) D	$[q_0q_1]$ B	$[q_0]$ A



* Construct FA to identify real number



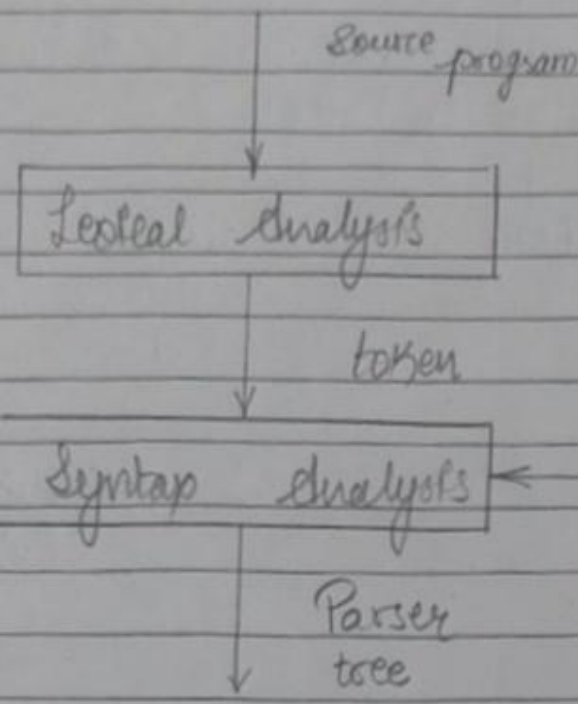
S = [+, -] ; DIGIT = [0-9] ; DOT = [.]



Unit 2. SYNTAX ANALYSIS

Page No.
Date : / /201

* Introduction & Overview.



→ Syntax analysis is second phase of compiler.
→ Syntax analysis verifies TOKEN according to grammar rules (CFG) and process them as valid or invalid. If valid then it generate parse tree or error message is sent.
→ parser (Output is parse tree)

* Passer: A program that takes TOKEN and Context free Grammar as input and validate input token accordingly to the rules of Grammar is called as PARSER.

To design Syntax analysis their are two ISSUES

CFG
PARSER

Page No.
Date : / /201

TOPICS

1. Syntax Specification of programming languages: CFG
2. Dealing with ambiguity
3. Design of TOP-DOWN and BOTTOM-UP passing technique.
   - Design of LL(1) passer
   - Design of LR passer
     - SLR | LR(0)
     - CLR | LR(1)
     - LALR | LR(2)
4. Passer Generator (YACC, BISON)

① SYNTAX SPECIFATION OF PROGRAMMING LANGUAGES

⇒ Context free Grammar

CFG → Programming languages
→ Syntax-specify

CFG is used to recognize context free language. That consist of non-terminal symbol, terminal symbol, production rule and start symbol (S) it is represented by four (4) tuples.

$$G = \langle V, T, P, S \rangle$$

where

V: Set of NON-TERMINAL SYMBOLS OR VARIABLE
T: Set of TERMINAL SYMBOLS
P: Set of PRODUCTION RULE
S: START SYMBOL.

CONTEXT FREE LANGUAGE

Page No.
Date : / /201

Defination:

The language which is generated by CONTEXT FREE GRAMMER or it is accepted by "PUSHDOWN AUTOMATA" then it is called as Context free language.

Context free language

Context free Grammar (CFG) (Type 2)

Pushdown automata = FA + 1 Auxiliary Memory

CONTEXT FREE GRAMMAR (CFG or TYPE 2):→

A Grammar 'G' is said to be an context free grammar if and only if every production having the form $\alpha \to \beta$ where

i) $|\alpha| \leq |\beta|$

ii) $\alpha \in V$ (Single Non-terminal)

iii) $\beta \in (V+T)^*$ [Arbitary string of terminal & non-terminal i.e. any number of terminal and non-terminal can be use at any position

Then such type of grammar is called as Context free Grammar or Type 2 grammar

Ex $S \to AA \mid AaB \mid aaAaaAB \mid aa$

$A \to aB \mid bB \mid bb \mid b \mid a \mid ba$

$B \to Aa \mid Ba \mid BBa \mid abB$

Specify = identify clearly

Page No.
Date : / /201

Every CFG is defined with 4 tuples

$G = \langle V, T, P, S \rangle$

where

V : Set of all non-terminal
T : Set of all terminal
P : Set of all production or Rule
S : Start symbol.

NOTE.

1) The derivation of string by using the productions is called as "SENTENTIAL FORM".

2) The Graphical Derivation of a string is called as Derivation tree or parse tree

3) By default we are applying left most derivation i.e. putting the value in left most non-terminal first or evaluating left most non-terminal first.

Ex. $S \to ABC$

$A \to aAb \mid a$

$B \to cBd \mid c$

$C \to a$

$V = \{S, A, B, C\}$

$T = \{a, b, c, d\}$

$P = \{ S \to ABC,\ A \to aAb,\ A \to a,\ B \to cBd,\ B \to c,\ C \to a \}$

$S = \{S\}$

CFG is use to generate the string

Derivation.

The process of generating string from the given grammar is called as derivation
There are two order of derivation.

1) LEFT MOST DERIVATION (LMD)
2) RIGHT MOST DERIVATION (RMD)

1) Left Most Derivation (LMD)

A derivation is said to be LEFT MOST, if in each step the left most non-terminal in a sentential form is replaced by terminal

2) Right Most Derivation (RMD)

A derivation is said to be RIGHT MOST if in each step the right most non-terminal in a sentential form is replaced by a terminal is called as RMD.

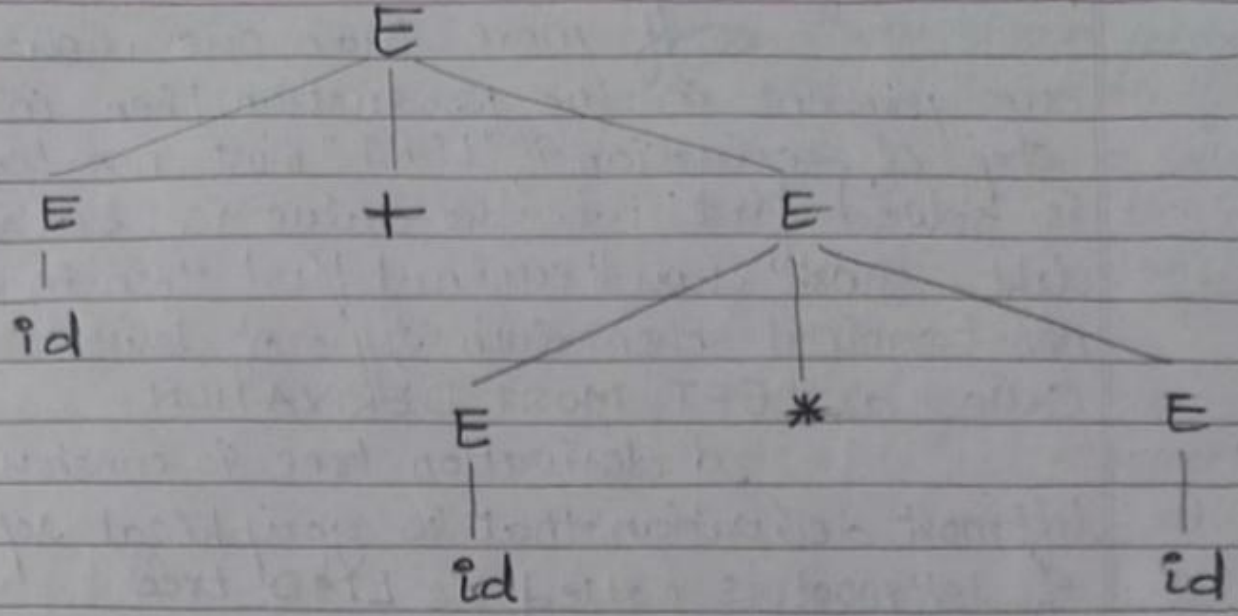
Q.] Find LMD and RMD for the following Grammar
$E \rightarrow E+E \mid E*E \mid id$

Sol:

LMD:

$E \rightarrow E+E$	$\{E \rightarrow E+E\}$
$E \rightarrow id+E$	$\{E \rightarrow id\}$
$E \rightarrow id+E*E$	$\{E \rightarrow E*E\}$
$E \rightarrow id+id*E$	$\{E \rightarrow id\}$
$E \rightarrow id+id*id$	$\{E \rightarrow id\}$

Date : / /201



RMD:

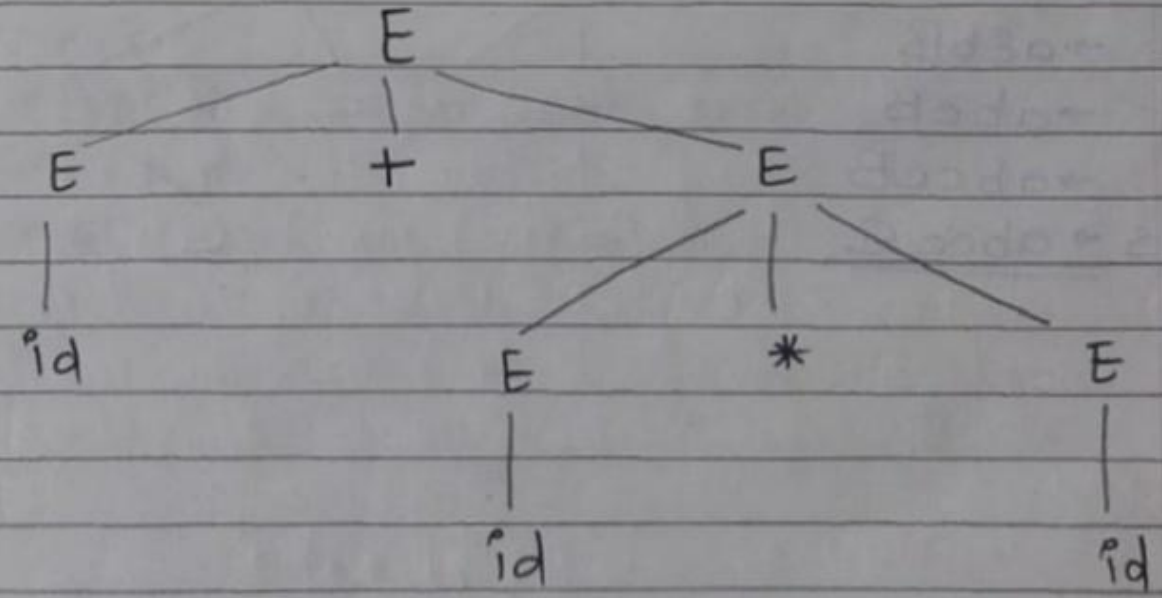
$E \rightarrow E + E$

$E \rightarrow E + E * E$ {$E \rightarrow E * E$}

$E \rightarrow E + E * id$ {$E \rightarrow id$}

$E \rightarrow E + id * id$ {$E \rightarrow id$}

$E \rightarrow id + id * id$ {$E \rightarrow id$}



LMD (Left Most Derivation)

Date: / /201

If more than one non-terminals are present in the production then in each step of derivation of "LEFT" most non-terminal is solved first i.e. the value is substitute in left most non-terminal first then in ~~right~~ most non-terminal then such type of derivation is called as LEFT MOST DERIVATION

A derivation tree is constructed for leftmost derivation that is graphical representation of leftmost is called as LMD tree

Ex:

$S \rightarrow AB$

$A \rightarrow aAb \mid \epsilon$

$B \rightarrow cB \mid \epsilon$

Left Most Derivation

$S \rightarrow AB$

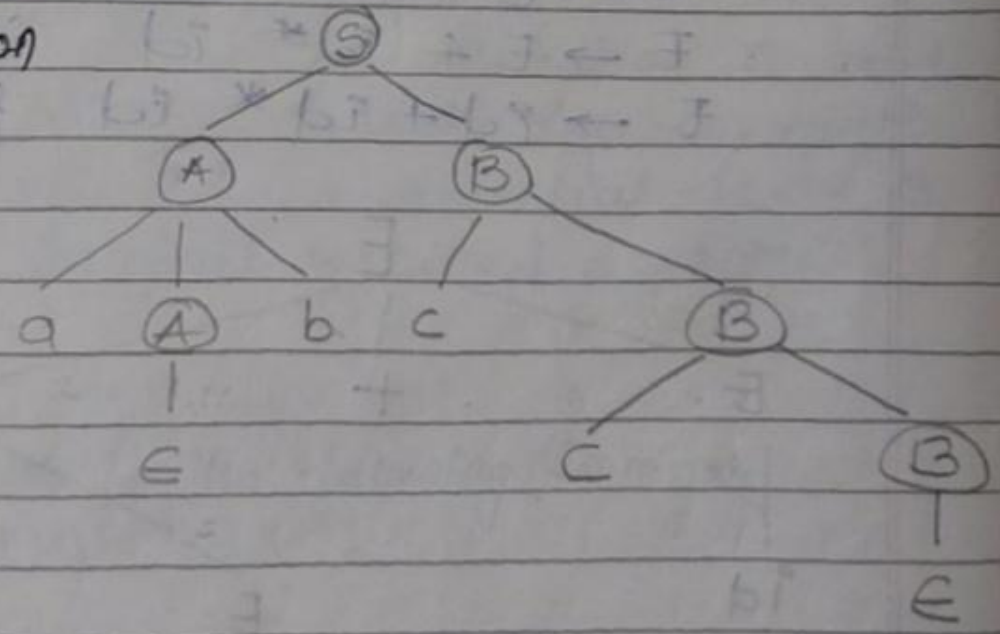
$\rightarrow aAbB$

$\rightarrow a\epsilon bB$

$\rightarrow abcB$

$\rightarrow abccB$

$S \rightarrow abcc\epsilon$.



RMD (Right Most Derivation)

If more than one non terminal are present in the production then in each step of derivation if right most non-terminal is solved first i.e value is substituted in right most non-terminal first then in left most non-terminal such type of derivation is called as Right Most Derivation

A derivation tree is constructed for right most derivation that is graphical representation of RMD is called as Right most derivation tree

Eg:

$S \rightarrow AB$

$A \rightarrow aAb \mid \epsilon$

$B \rightarrow cB \mid \epsilon$

Derive : abcc

RMD derivation.

$S \rightarrow AB$

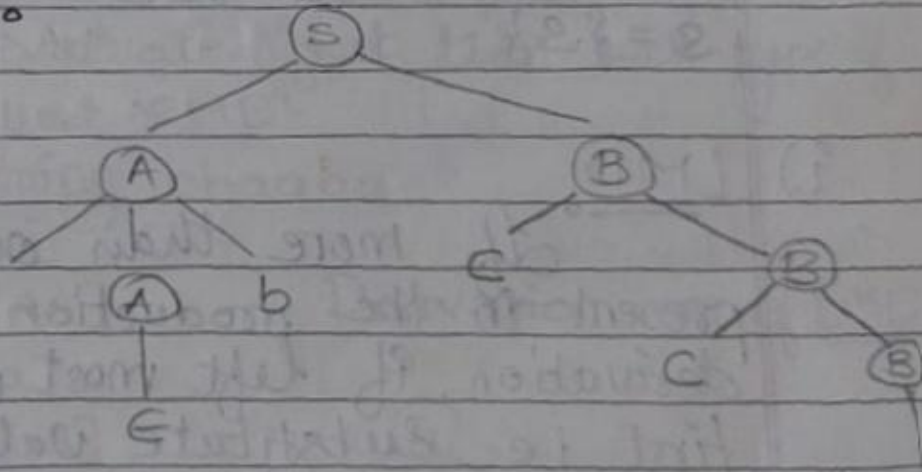
$\rightarrow AcB$

$\rightarrow AccB$

$\rightarrow aAbcc\epsilon$

$\rightarrow aAbcc$

$\rightarrow abcc$



(*) Let 'G' be the grammar whose Date: / /201
production are defined as

$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$

for the string "bbaaba"
Find i) LMD
ii) RMD
iii) Parse tree

Sol: A Grammar "G" is defined with 4 tuple

$G = \langle V, T, P, S \rangle$

$V = \{S, A, B\}$
$T = \{a, b\}$
$P = \begin{cases} S \rightarrow aB \mid bA \\ A \rightarrow a \mid aS \mid bAA \\ B \rightarrow b \mid bS \mid aBB \end{cases}$
$S = \{S\}$

i) LMD

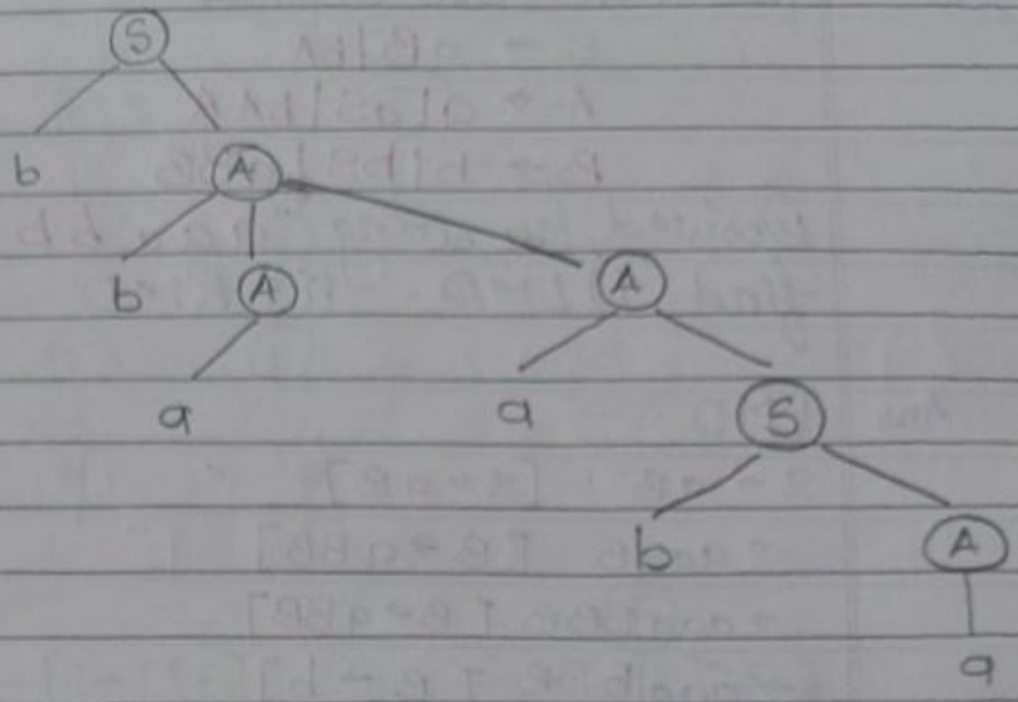
If more than one non terminal is present in the production then in each step of derivation, if left most non-terminal is solved first i.e. substitute value of in left most non-terminal first then in right most non-terminal then such type of derivation is called LMD

Derivation of string "bbaaba"

$S \rightarrow bA$ .....[$S \rightarrow bA$]
$\rightarrow bbAA$ ....[$A \rightarrow bAA$]
$\rightarrow bbaA$ ....[$A \rightarrow a$]
$\rightarrow bbaaS$ ....[$A \rightarrow aS$]
$\rightarrow bbaabA$ ....[$S \rightarrow bA$]
$\rightarrow bbaaba$ .....[$A \rightarrow a$]

Date : / /201

Derivation tree of LMD.



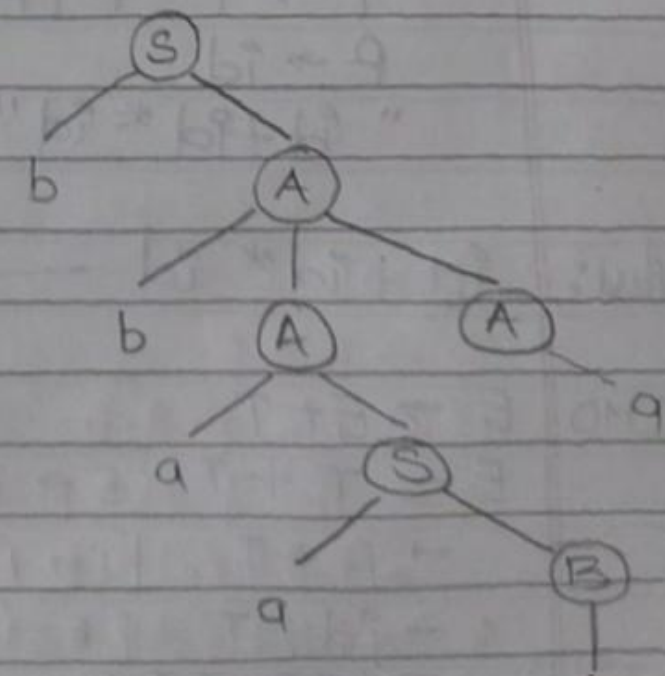
(ii) RMD

If more than production is applied to right most non-terminal first and then production is applied to left most non terminal it means that right most non-terminal is evaluated first then such type of derivation is called "RMD"

Derivation of string bbaaba

S → bA [S → bA]
→ bbAA [A → bAA]
→ bbAa [A → a]
→ bbaSa [A → aS]
→ bbaaBa [S → aB]
→ bbaaba [B → b]

Derivation tree of LMD.



Self by HM

Let 'G' be the grammar whose production are defined

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

Derived the string "aaabbabbba"

find i) LMD ii) RMD iii) Parse tree

Ans LMD

$S \to aB$ [S → aB]

$\to aaBB$ [B → aBB]

$\to aaaBBB$ [B → aBB]

$\to aaabBB$ [B → b]

$\to aaabbB$ [B → b]

$\to aaabbaBB$ [B → aBB]

$\to aaabbabB$ [B → b]

$\to aaabbabbS$ [B → bS]

$\to aaabbabbbA$ [S → A]

$\to aaabbabbba$ [A → a]

Problem: Construct LMD and RMD for the following Grammar

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id$

"id+id*id" string

Sol: id+id*id ⟶ string

LMD

$E \to E+T$

$E \to T+T$ [E → T]

$\to F+T$ [T → F]

$\to id+T$ [F → id]

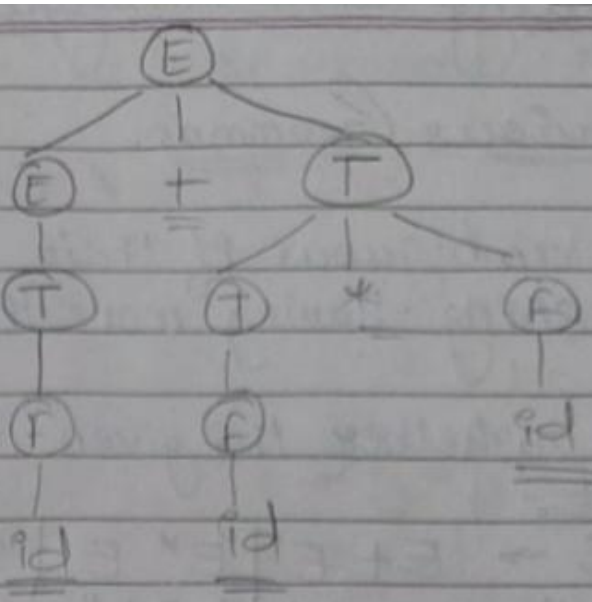
[T → T*F]

$\to id+T*F$

$\to id+F*F$

$\to id+id*F$ [F → id]

$\to id+id*id$ [F → id]



MD $E \rightarrow E+T$

$\rightarrow E+T*F$ —[$T \rightarrow T*F$]

$\rightarrow E+T*id$ [$F \rightarrow id$]

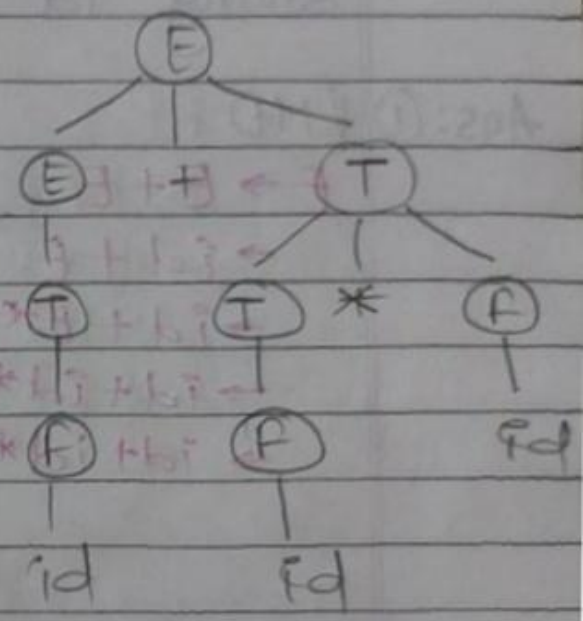
$\rightarrow E+F*id$ [$T \rightarrow F$]

$\rightarrow E+id*id$ [$F \rightarrow id$]

$\rightarrow T+id*id$ [$E \rightarrow T$]

$\rightarrow F+id*id$ [$T \rightarrow F$]

$\rightarrow id+id*id$ [$F \rightarrow id$]



Dealing with Ambiguity.

* **Ambiguious Grammar.**

A Grammar "G" is said to be ambiguous if their exsist at least one string having more than one parse tree.

Q] Check whether the given Grammar is ambiguous or not.

$E \rightarrow E+E \mid E*E \mid id$

string "id + id * id."

Ans: ① LMD

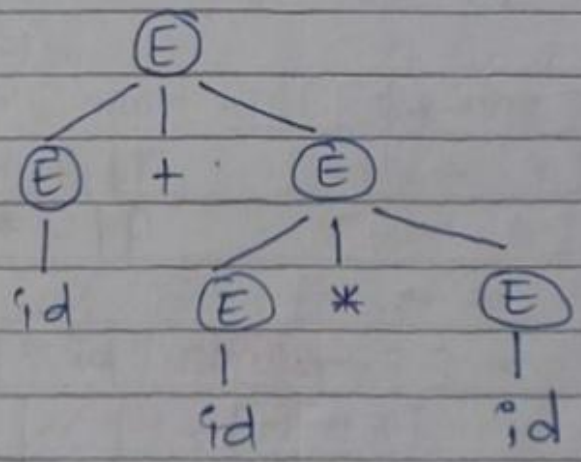
$E \rightarrow E+E$

$\rightarrow id+E$ ____ $E \rightarrow id$

$\rightarrow id+E*E$ ____ $E \rightarrow E*E$

$\rightarrow id+id*E$ ____ $E \rightarrow id$

$\rightarrow id+id*id$ ____ $E \rightarrow id$



② LMD

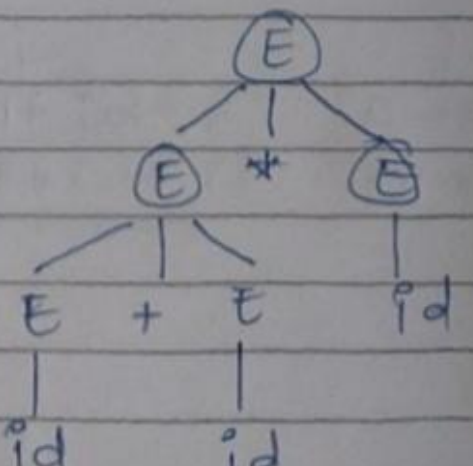
$E \rightarrow E*E$

$E \rightarrow E+E*E$

$\rightarrow id+E*E$

$\rightarrow id+id*E$

$E \rightarrow id+id*id$



There exist more than one parse tree

Hence grammar which is given is

AMBIGUOUS

Substitute the rest of production.

Page No.
Date : / /201

9] Check whether the given Grammar is ambiguous or not.

$S \rightarrow aSbS \mid bSaS \mid \epsilon$

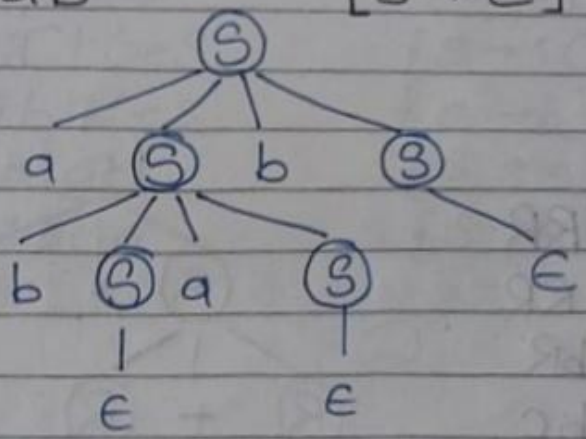
$\omega =$ "abab"

Ans. LMD

$S \rightarrow aSbS$

$\rightarrow abSaSbS \quad [S \rightarrow bSaS]$

$\rightarrow abab \quad [S \rightarrow \epsilon]$



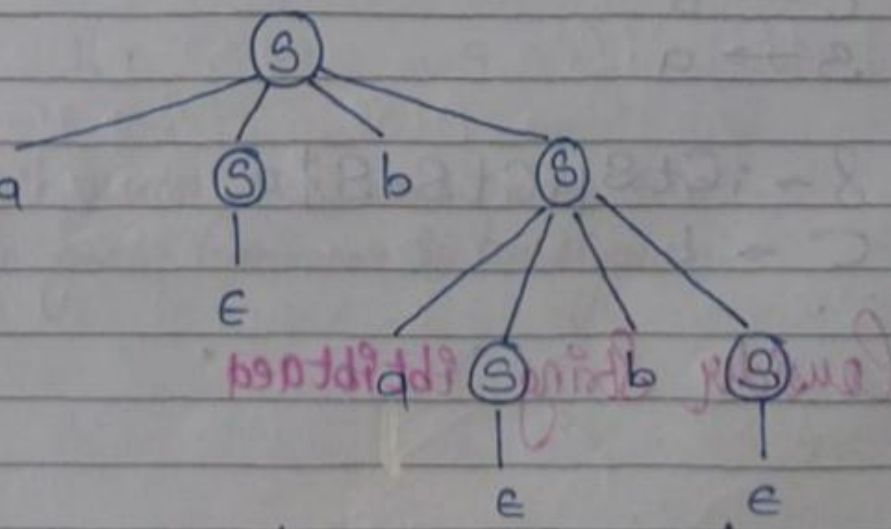
RMD

$S \rightarrow aSbS$

$S \rightarrow abS \longrightarrow [S \rightarrow \epsilon]$

$S \rightarrow abaSbS \longrightarrow [S \rightarrow aSbS]$

$S \rightarrow abab \longrightarrow [S \rightarrow \epsilon]$



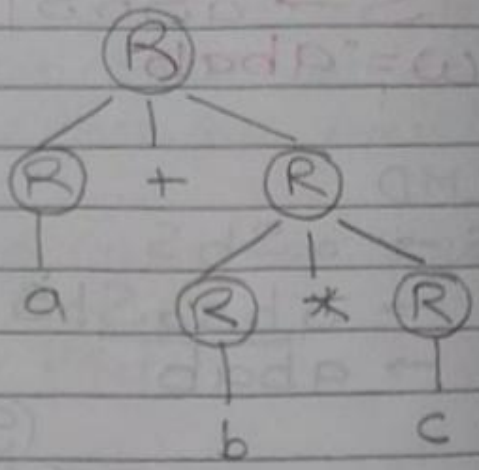
There exist more than one Parse Tree Hence given Grammar is AMBIGUOUS.

H.W. $R \to R+R \mid RR \mid R^* \mid a \mid b \mid c$

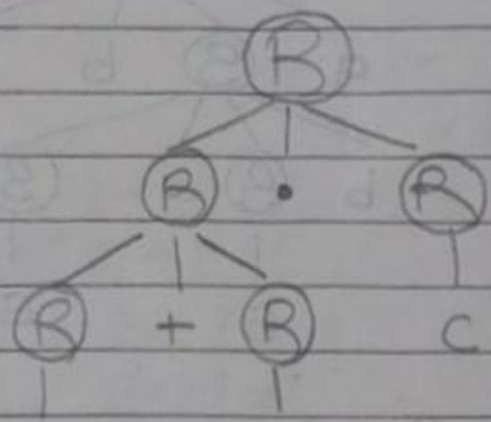
$\omega = a+b*c$

Sol

$R \to R+R$
$\to R+R \cdot R$
$\to a+R \cdot R$
$\to a+b \cdot R$
$\to a+b \cdot c$



$R \to RR$
$\to R+RR$
$\to a+RR$
$\to a+bR$
$\to a+bc$



Q 4M What is meant by ambiguity of grammar? Check the given Grammar is ambiguous or not

$S \to iEtS \mid iCtSeS$
$C \to b$
$S \to a$

Sol $S \to iCtS \mid iCtSeS \mid a$
$C \to b$

Consider String "ibtibtaea"

LMD $S \to iCtS$

$\to ibtS$ $\{C \to b\}$

$\to ibtiCtSeS$ $\{S \to iCtSeS\}$

$\to ibtibtSeS$ $\{C \to b\}$

$\to ibtibtaeS$ $\{S \to a\}$

$\to ibtibtaea$ $\{S \to a\}$

LMD $S \to iCtSeS$

$S \to ibtSeS$ $\{C \to b\}$

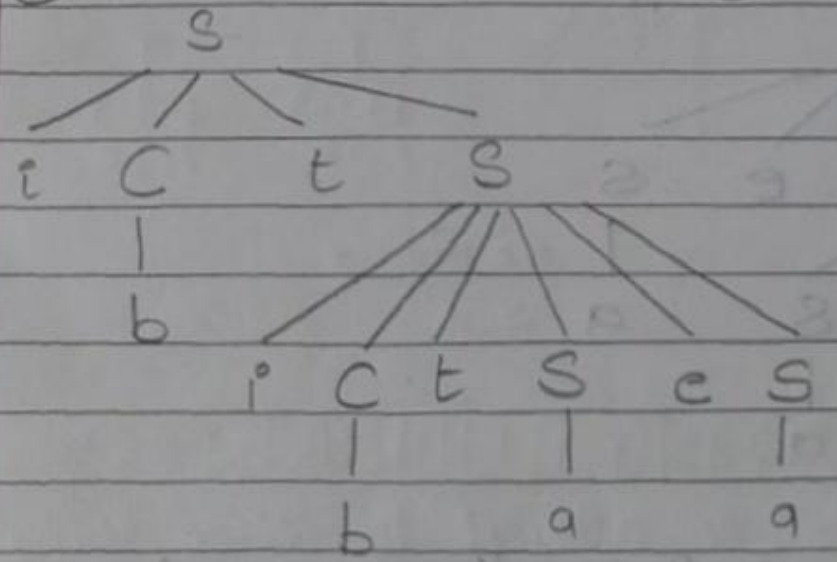
$S \to ibtiCtSeS$ $\{S \to iCtSeS\}$

$S \to ibtibtSeS$ $\{C \to b\}$

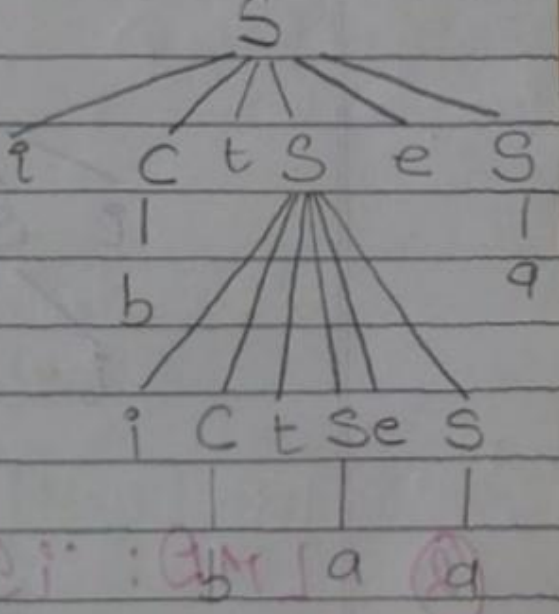
$S \to ibtibtaeS$ $\{S \to a\}$

$S \to ibtibtaea$ $\{S \to a\}$

①



②



There exist more than one parse tree
Hence the given Grammar is ambiguous.

④ Length should not be less than 5 terminals.

Page No.
Date : / /201

W-17 Q] 4M Determine whether given Grammar is ambiguous or not

$$S \rightarrow iSeS \mid iS \mid a$$

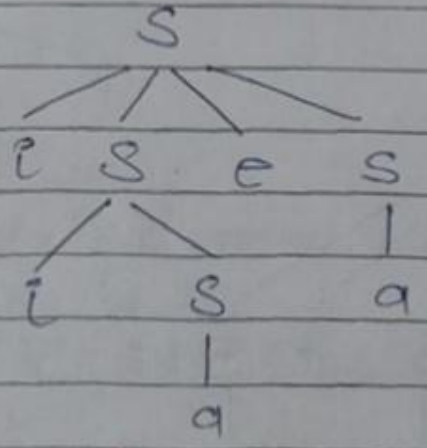
If YES remove ambiguity and re-write the grammar.

Soln $S \rightarrow iSeS \mid iS \mid a$

consider the string → iiaea

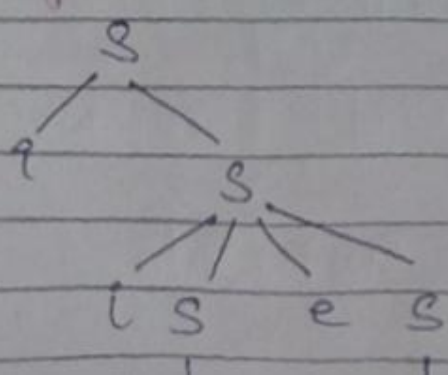
① LMD

$S \rightarrow iSeS$
$\rightarrow iiSeS$ [S → iS]
$\rightarrow iiaeS$ [S → a]
$\rightarrow iiaea$ [S → a]



② LMD: 'iS'

$S \rightarrow iiSeS$ {S → iSeS}
$S \rightarrow iiaeS$ {S → a}
$S \rightarrow iiaea$ {S → a}



There exsit more than one parse tree
Hence the given Grammar is Ambigious.

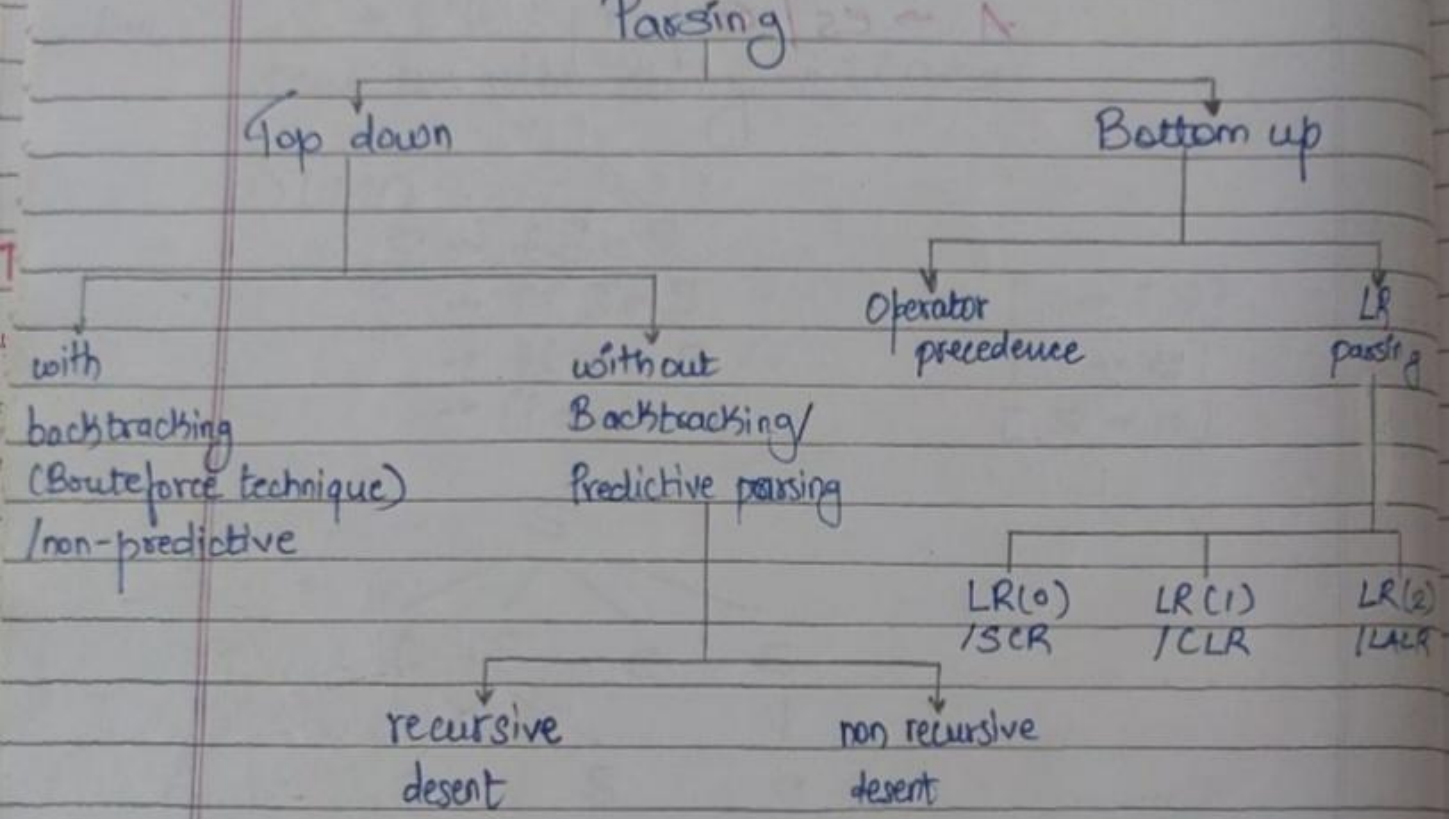
Remove dubiguity.

$$S \rightarrow iSA \mid a$$
$$A \rightarrow es \mid \epsilon$$

3. Design of TOP DOWN & BOTTOM UP Parsing techniques:

Page No.
Date : / /201

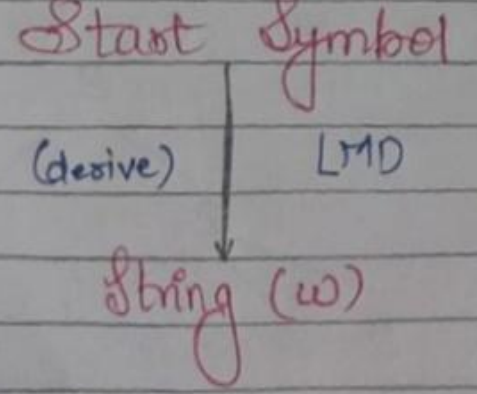
Classified into following types shown in the figure



Wed 17/07/2019

① TOP DOWN PARSING TECHNIQUE:

As an attempt to drive string 'w' from start symbol of grammar by constructing parse tree in LMD shown in the figure

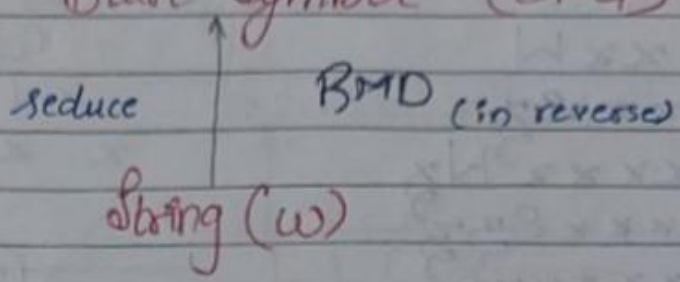


Q TM
Confirm on LL(1).

Page No.
Date : / /201

② BOTTOM-UP PARSING :

Bottom up is an attempt to reduce a string 'w' to the start symbol of Grammar by constructing parse tree in RMD reverse order



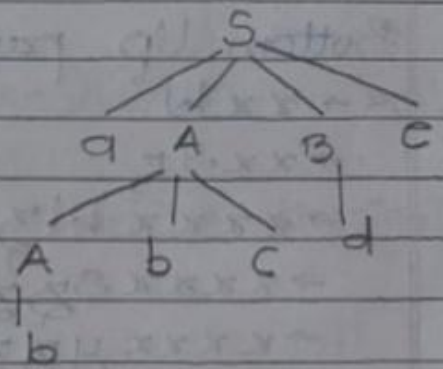
Problem
$S \rightarrow aABe$
$A \rightarrow Abc / b$
$B \rightarrow d$
String (w) = abbcde.

Ans

1. TOP DOWN PARSING

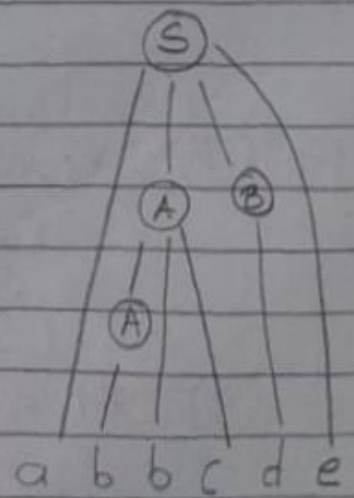
LMD

$S \rightarrow aABe$
$\rightarrow aAbcBe$ {S→Abc}
$\rightarrow abbcBe$ {A→b}
$\rightarrow abbcde$ {B→d}



2. BOTTOM UP PARSING

$S \rightarrow aABe$
$\rightarrow aAde$ {B→d}
$\rightarrow aAbcde$ {A→Abc}
$\rightarrow abbcde$ {A→d}

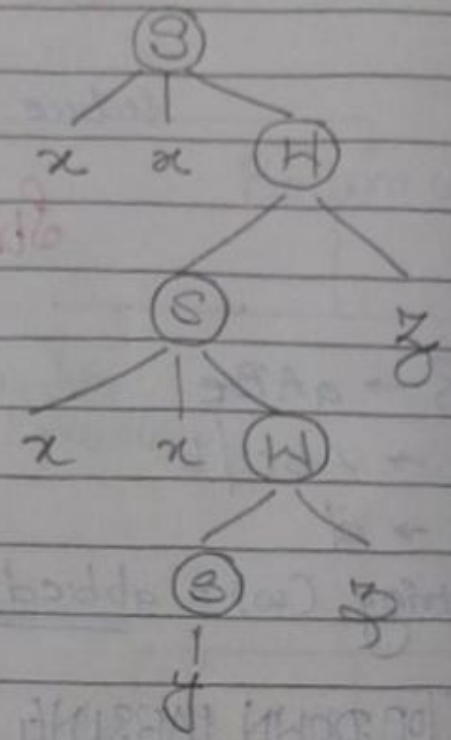


Q. $S \rightarrow xxW|y$
$W \rightarrow Sz$

string: xxxxyzz

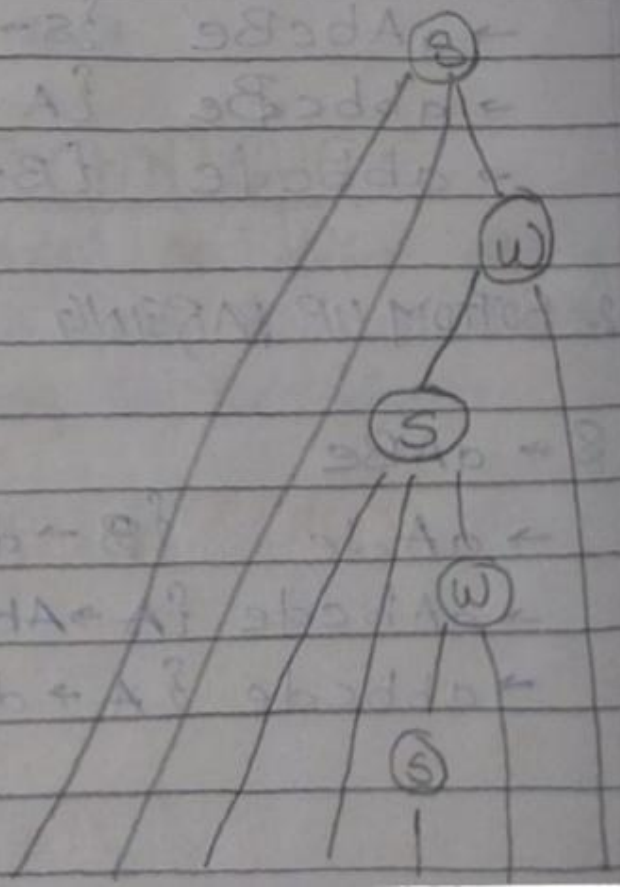
Ans Top down passing

$S \rightarrow xxW$
$S \rightarrow xxSz$
$S \rightarrow xxxxWz$
$S \rightarrow xxxxSzz$
$S \rightarrow xxxxyzz$



Bottom Up passing

$S \rightarrow xxW$
$\rightarrow xxSz$
$\rightarrow xxxxWz$
$\rightarrow xxxxSzz$
$\rightarrow xxxxyzz$



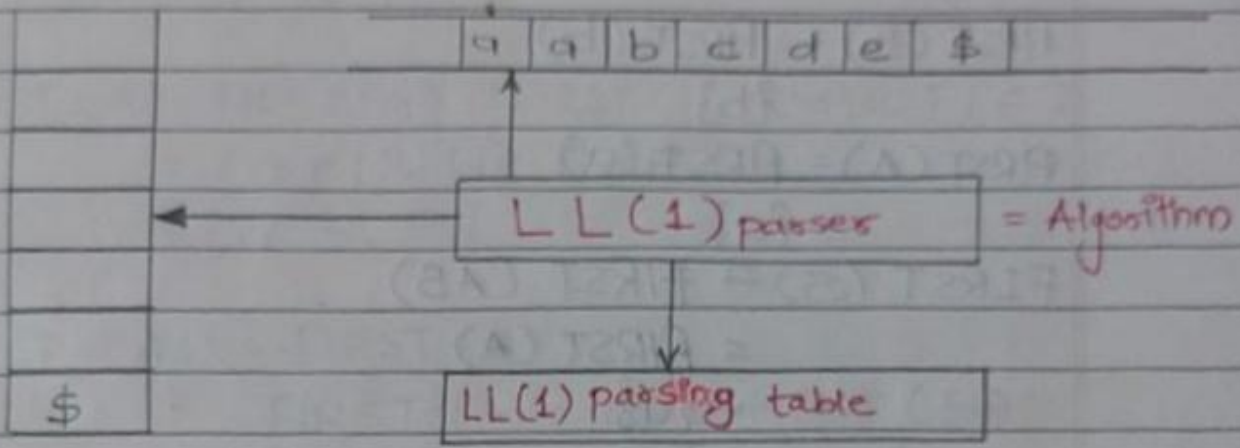
TOP DOWN:

Page No.
Date: / /201

→ length of look-ahead symbol.

LL(1) passer
- ↳ Left most derivation (LMD)
- ↳ left to right scan

Structure or Component:



1) FIRST( ).
2) FOLLOW( ).

Symbol
- Non-terminal (upper case) (A, B, ... Z, A', B', ..Z')
- terminal (T) → other than Non-terminal.

FIRST SET

1) FIRST (terminal) = {terminal}

2) If $A \to a\alpha$
where $a \in T$ & $\alpha \in (V \cup T)^*$
FIRST (A) = {a}

3) If $A \to B\alpha$ (where $B \neq \epsilon$)
than FIRST (A) = FIRST (B)

4 If $A \to B\alpha$ (where $B = \epsilon$) than
FIRST (A) = FIRST (B) − ϵ ∪ FIRST (α)

PROBLEM Calculate FIRST set for the following Grammar

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$

Ans:

Calculation of FIRST SET

FIRST (B) = FIRST (b)
= {b}
FIRST (A) = FIRST (a)
= {a}
FIRST (S) = FIRST (AB)
= FIRST (A)
= {a}

PROBLEM

$S \rightarrow AB$
$A \rightarrow a|b$
$B \rightarrow c|d$

Ans:

Calculation of FIRST SET

Fiost (B) = Fiost (c) ∪ FioBt (d)
= {c} ∪ {d}
= {c, d}

FIRST (A) = Fiost (a) ∪ FIRST (b)
= {a} ∪ {b}
= {a, b}

Date : / /201

PROBLEM $S \rightarrow AB$
$A \rightarrow a|b|\epsilon$
$B \rightarrow c|d$

Ans FIRST (B) = FIRST (c) ∪ FIRST (d)
= {c} ∪ {d}
= {c, d}

FIRST (A) = FIRST (a) ∪ FIRST (b) ∪ FIRST (ϵ)
= {a} ∪ {b} ∪ {ϵ}
= {a, b, ϵ}

FIRST (AB) = FIRST (AB)
= FIRST (A) − ϵ ∪ FIRST (B)
= {a, b, ϵ} − ϵ ∪ {c, d}
= {a, b, c, d}

PROBLEM $S \rightarrow ABC$
$A \rightarrow a|b|\epsilon$
$B \rightarrow c|d|\epsilon$
$C \rightarrow e|\epsilon$

Ans Calculation of FIRST :
FIRST (C) = FIRST (e) ∪ FIRST (ϵ)
= {e} ∪ {ϵ}
= {e, ϵ}
FIRST (B) = FIRST (c) ∪ FIRST (d) ∪ FIRST (ϵ)
= {c} ∪ {d} ∪ {ϵ}
= {c, d, ϵ}
FIRST (A) = FIRST (a) ∪ FIRST (b) ∪ FIRST (ϵ)
= {a} ∪ {b} ∪ {ϵ}
= {a, b, ϵ}

Date : / /201

FIRST (S) = FIRST (ABC)
= FIRST (A) − ∈ FIRST (BC)
= (a, b, ∉) − ∈ FIRST (BC)
= {a, b} ∪ FIRST (B) ∪ FIRST (C)
= {a, b} ∪ FIRST (B) − ∈ FIRST (C)
= {a, b} ∪ {c, d, ∈} − ∈ ∪ FIRST (C)
= {a, b, c, d} ∪ FIRST (C)
= {a, b, c, d} ∪ {e, ∈} − ∈
= {a, b, c, d, e}

PROBLE

Sus:

* S → ABC
A → a | b | ∈
B → c | d
C → e | ∈

Sus Calculation of FIRST SET.

FIRST (∈) = first (e) ∪ first (∈)
= {e} ∪ {∈}
= {e, ∈}

FIRST (B) = first (c) ∪ first (d)
= {c} ∪ {d}
= {c, d}

FIRST (A) = FIRST (a) ∪ FIRST (b) ∪ FIRST (∈)
= {a} ∪ {b} ∪ {∈}
= {a, b, ∈}

FIRST (S) = FIRST (ABC)
= FIRST (A) ∪ FIRST (BC)
= {a, b, ∈) − ∉ FIRST (BC)

$= \{a,b\} \cup FIRST(B, C\}$

$= \{a,b\} \cup FIRST(B)$

$= \{a,b\} \cup \{c,d\}$

$= \{a,b,c,d\}$

PROBLEM $f \to XYZa$

$X \to x \mid \epsilon$

$Y \to y \mid \epsilon$

$Z \to z \mid \epsilon$

Ans: Calculate of FIRST SET

$FIRST(Z) = FIRST(z) \cup FIRST(\epsilon)$

$= \{z\} \cup \{\epsilon\}$

$= \{z, \epsilon\}$

$FIRST(Y) = FIRST(y) \cup FIRST(\epsilon)$

$= \{y\} \cup \{\epsilon\}$

$= \{y, \epsilon\}$

$FIRST(X) = FIRST(x) \cup FIRST(\epsilon)$

$= \{x\} \cup \{\epsilon\}$

$= \{x, \epsilon\}$

$FIRST(F) = FIRST(X) \cup FIRST(Y) \cup FIRST(Z) \cup a$

$= \{x, \epsilon\} - \epsilon \cup FIRST(Y) \cup FIRST(z) \cup a$

$= \{x\} \cup \{y, \epsilon\} - \epsilon \cup FIRST(Z) \cup a$

$= \{x, y\} \cup FIRST(Z) \cup a$

$= \{x, y\} \cup \{z, \epsilon\} - \epsilon \cup a$

$= \{x, y, z, a\}$

$= \{a, x, y, z\}$

PROBLEM
$S \rightarrow aABC$ → {a}
$A \rightarrow aAB \mid \epsilon$ → {a, ∈}
$B \rightarrow bB \mid c$ → {b, c}
$C \rightarrow d$ → {d}

PROBLEM
$S \rightarrow Aabb$ → {a, b}
$A \rightarrow a \mid b$ → {a, b}
$B \rightarrow c \mid d$ → {c, d}

PROBLEM
$S \rightarrow aAB \mid bA \mid \epsilon$
$A \rightarrow aAB \mid \epsilon$
$B \rightarrow bB \mid c$

Ans:

FIRST (B) = First (b, B) ∪ FIRST (c)
= First (b) ∪ FIRST (c)
= {b} ∪ {c}
= {b, c}

FIRST (A) = FIRST (aAB) ∪ FIRST (∈)
= FIRST (a) ∪ FIRST (∈)
= {a} ∪ {∈}
= {a, ∈}

FIRST (S) = FIRST (aAB) ∪ FIRST (bA) ∪ FIRST (∈)
= FIRST (a) ∪ FIRST (b) ∪ FIRST (∈)
= {a} ∪ {b} ∪ {∈}
= {a, b, ∈}

Page No.
Date: / /201

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

Ans: $First(F) = First((E)) \cup First(id)$
$= First(() \cup \{id\}$
$= (, id)$

$FIRST(T) = FIRST(T*F) \cup FIRST(F)$
$= FIRST(T*F) \cup FIRST(F)$
$= FIRST(T) \cup FIRST((, id)$
$= FIRST(T) \cup (, id)$
$= FIRST(F) \cup (, id)$
$= (, id) \cup (, id)$
$= (, id)$

$FIRST(E) = FIRST(E+T) \cup FIRST(T)$
$= FIRST(E) \cup FIRST(T)$
$= FIRST(T) \cup (, id)$
$= FIRST(F) \cup (, id)$
$= (, id) \cup (, id)$
$= (, id$

PROBLEM $E \to 10*T \mid 5+T$
$T \to PS$
$S \to QP \mid E$
$Q \to + \mid *$
$P \to a \mid b \mid c$

Date : / /201

Ans. FIRST (P) = {a, b, c}

FIRST (Q) = {+, *}

FIRST (S) = FIRST (QP) ∪ FIRST (∈)

= FIRST (Q) ∪ FIRST (E)

= (+, *) ∪ {5, 10}

= {5, 10, +, *}

FIRST (T) = FIRST (PS)

= FIRST (P)

= {a, b, c}

FIRST (E) = FIRST (10* T) ∪ FIRST (5+T)

= FIRST (10) ∪ FIRST (5)

= {10} ∪ {5}

= {5, 10}

PROBLEM $S \rightarrow \{P\}$

$P \rightarrow P; P \mid \in$

$P \rightarrow a \mid b$

Sol:" FIRST (P) = FIRST (P, P) ∪

FIRST (a) ∪ FIRST (b) ∪ FIRST (∈)

= FIRST (P; P) ∪ {a} ∪ {b} ∪ {∈}

= FIRST (P) − ∈ ∪ FIRST (; P) ∪ {a, b, ∈}

= {a, b, ∈} − ∈ ∪ {;} ∪ {a, b, ∈}

= {a, b} ∪ {;} ∪ {a, b, ∈}

= {a, b, ;, ∈}

FOLLOW SET

Page No.
Date : / /201

To find all Next terminal Symbol.

RULES :

1. FOLLOW (start) = {$}
2. If $A \to \alpha B$
   than FOLLOW (B) = FOLLOW (A)
3. If $A \to \alpha B\beta$ $(\beta \neq \epsilon)$
   than
   FOLLOW (B) = FIRST (β)
4. If $A \to \alpha B\beta$ $(\beta = \epsilon)$
   than
   FOLLOW (B) = FIRST (β) $- \epsilon \cup$ FOLLOW (A)

NOTE FOLLOW SET NEVER CONTAIN ϵ (EPSILON)

First & follow only for Non-terminal
First to calculate from BOTTOM of production
Follow to calculate from TOP of production

PROBLEM Calculate 1st and FOLLOW of the following Gramm

$S \to AB$
$A \to a$
$B \to b$

Ans ① Calculate of FIRST SET

First (B) = {b}
First (A) = {a}
First (S) = First (A)
= {a}

Page No.
Date : / /

Calculate of FOLLOW

FOLLOW (B) = {$}
FOLLOW (A) = FIRST (B)
= {b}
FOLLOW (B) = FOLLOW (S)
= {$}

② S → AB
A → a|b
B → c|d

Sol: FIRST (B) = {c, d}
FIRST (A) = {a, b}
FIRST (S) = {a, b}

FOLLOW (S) = {$}
FOLLOW (A) = FIRST (B)
= {c, d}
FOLLOW (B) = FOLLOW (S)
= {$}

3 S → AB
A → a|b|ϵ
B → c|d

Ans FIRST (B) = {c, d}
FIRST (A) = {a, b, ϵ}
FIRST (S) = {a, b, c, d}

FOLLOW (S) = {$}
FOLLOW (A) = FIRST (B)
= {c, d}
FOLLOW (B) = FOLLOW (S)
= {$}

★ S → ABC
A → a|b|ϵ
B → c|d|ϵ
C → e|ϵ

Ans FIRST (C) = {e, ϵ}
FIRST (B) = {c, d, ϵ}
FIRST (A) = {a, b, ϵ}
FIRST (S) = {a, b, c, d, e, }

FOLLOW (S) = {$}.
FOLLOW (A) = FIRST (BC)
= FIRST (B) - ϵ ∪ FIRST (C)
= {c, d, ~~ϵ~~} - ϵ FIRST (C)
= {c, d} ∪ {e, ϵ} - ϵ ∪ {S}
= {c, d}, {e} ∪ {$}
= {$, c, d, e}
FOLLOW (B) = FIRST (C)
= {e - ~~ϵ~~} - ϵ ∪ FOLLOW (S)
= {e} ∪ {$}
= {e, $}
FOLLOW (C) = FOLLOW (S)
= {$}

★ F → XYZa
X → x | ϵ
Y → y | ϵ
Z → z | ϵ

Ans FIRST (Z) = {z, ϵ}

FIRST (Y) = {y, ϵ}

FIRST (X) = {x, ϵ}

FIRST (F) = {a, x, y, z}

So

FOLLOW SET

FOLLOW (F) = {$}

FOLLOW (X) = FIRST (YZ)

= FIRST (Y) − ϵ ∪ FIRST (Z) − ϵ ∪ FIRST (a)

= FIRST (Y − ϵ) ~~or~~ FIRST (Z − ϵ) ∪ {a}

= {y} ∪ {z} ∪ {a}

= {y, z, a}

FOLLOW (Y) = FIRST (Z)

= {z, a}

FOLLOW (Z) = FIRST (a)

= a

★ S → aABC

A → a | b | ϵ

B → c | d

C → e | ϵ

Page No.
Date : / /201

★ FIRST (C) = {e, ∈}
FIRST (B) = {c, d}
FIRST (A) = {a, b, ∈}
FIRST (S) = {a}

FOLLOW SET
FOLLOW (S) = {$}
FOLLOW (A) = FIRST (BC)
= FIRST (B)
= {c, d}
FOLLOW (B) = FIRST (C)
= {e - ∈} - ∈ ∪ FOLLOW (S)
= {e} ∪ {$}
= {e, $}
FOLLOW (C) = FOLLOW (S)
= {$}

★
S → aABC	{a}
A → a | ∈	{a, ∈}
B → b | c	{b, c}
C → d	{d}

FIRST

Ans FOLLOW SET
FOLLOW (S) = {$}
FOLLOW (A) = FIRST (BC)
= FIRST (B)
= {b, c}
FOLLOW (B) = FIRST (C)
= {d}
FOLLOW (C) = FOLLOW (S)
= {$}

Date : / /201

★ $S \to AabB$ {a,b}
$A \to a|b$ {a,b} FIRST
$B \to c|d$ {c,d}

Ans FOLLOW SET

FOLLOW(S) = {$}

FOLLOW(A) = FIRST(abB)
= {a}

FOLLOW(B) = FOLLOW(S)
= {$}

★

★ $S \to aAB|bA|\epsilon$ (a,b,ϵ)
$A \to aAb|\epsilon$ (a,ϵ)
$B \to bB|\epsilon$ (b,ϵ)

Ans

Ans FOLLOW SET

FOLLOW(S) = {$}

FOLLOW(A) = FIRST(B) ∪ FOLLOW(S) ∪ FIRST(b)
= {b,c} ∪ {$} ∪ {b}
= {b, c, $}

FOLLOW(B) = FOLLOW(S) ∪ FOLLOW(B)
= {$}

★ $E \to E+T|T$ ((, id)
$T \to T*F|F$ ((, id) FIRST
$F \to (E)|id$ (, id)

Ans FOLLOW SET

FOLLOW(E) = {$} ∪ FIRST(+T) ∪ FIRST())
= {+,), $}

FOLLOW(T) = FOLLOW(E) ∪ FOLLOW(E) ∪ FIRST(*, f)

= {+,), $} ∪ {+,), $} ∪ {*}
= {+, *,), $}

FOLLOW(f) = FOLLOW(T) ∪ FOLLOW(T)
= {+, *,), $}

* E → 10* T | 5+T
T → PS
S → QP | E
Q → + | *
P → a|b|c

{5, 10}
{a, b, c}
{+, *, 5, 10} FIRST
{+, *}
{a, b, c}

Ans FOLLOW(E) = {$} ∪ FOLLOW(S)
= {$} ∪ FOLLOW(T)
= {$} ∪ ~~FOLLOW(E)~~
= {$}

FOLLOW(T) = FOLLOW(E) ∪ FOLLOW(E)
= FOLLOW(E)
= {$}

FOLLOW(S) = FOLLOW(T)
= {$}

FOLLOW(Q) = FIRST(P)
= {a, b, c}

FOLLOW(P) = FIRST(S) ∪ FOLLOW(S)
= {+, *, 5, 10} ∪ FOLLOW(S)
= {+, *, 5, 10} ∪ {$}
= {$, +, *, 5, 10}

★ S → {P}
P → P; P | ∈
P → a | b

★ FIRST (P) = {a, b, c}
FIRST (S) = FIRST ({P})
= {{}

FOLLOW SET
FOLLOW (S) = {$}
FOLLOW (P) = FIRST (}) ∪ FIRST {; P} ∪ FOLLOW(P)
= {} ∪ {;}
= (}, ;)

* LEFT RECURSION (LR):

A production of form

$$A \rightarrow A\alpha \mid \beta$$

is called a LR

To eliminate LR

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \epsilon$$

GENERAL RULE

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots\ldots \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots$$

To eliminate LR

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots\ldots$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots\ldots \mid \epsilon$$

Date : / /201

Q1. Eliminate Left Recursive for the following Grammar

$S \rightarrow aA \mid Bc$

$A \rightarrow Aa \mid Ac \mid b$

$B \rightarrow b$

→ Elimination of LEFT RECUSSION

$S \rightarrow aA \mid Bc$

$A \rightarrow bA'$

$A' \rightarrow aA' \mid cA' \mid \epsilon$

$B \rightarrow b$

Q2. $E \rightarrow E+T \mid T$

$T \rightarrow T^* F \mid F$

$F \rightarrow (E) \mid id$

→ Elimination of left Recursion

$E \rightarrow TE'$

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

$T \rightarrow FT'$

$T \rightarrow * FT' \mid \epsilon$

$f \rightarrow (E) \mid id$

Q3 $S \rightarrow a \mid b \mid c \mid aS \mid Sb \mid Sc \mid \epsilon$

Ans Elimination of left recussion

$S \rightarrow aS' \mid bS' \mid cS' \mid aSS' \mid S'$

$S' \rightarrow bS' \mid cS' \mid \epsilon$

$S \rightarrow aS' \mid bS' \mid cS' \mid aSS' \mid s'$

$S' \rightarrow bS' \mid cS' \mid \epsilon$

LEFT FACTORING (LF)

Date : / /201

* A production of form

$A \to \alpha\beta_1 \mid \alpha\beta_2$

To eliminate LF

$A \to \alpha A'$

$A' \to \beta_1 \mid \beta_2$

General Rule

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_2 \mid \ldots\ldots \mid \gamma_1 \mid \gamma_2 \mid \gamma_3$

To Eliminate LF

$A \to \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \gamma_3$

$A' \to \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots\ldots$

1. Eliminate left factor for the following Grammar

$A \to cD \mid cd \mid cb \mid \varepsilon \mid R$

Eliminate of left factoring

$A \to CA' \mid \varepsilon \mid R$

$A' \to D \mid d \mid b.$

Date : / /201

Q2 $S \rightarrow ABC$
$A \rightarrow abB \mid c \mid abc \mid b$
$B \rightarrow bB \mid c$
$C \rightarrow abc \mid aA \mid c$

Sol Elimination of LF

$S \rightarrow ABC$
$A \rightarrow abA' \mid c \mid b$
$A' \rightarrow B \mid C$
$B \rightarrow bB \mid c$
$C \rightarrow aC' \mid c$
$C' \rightarrow bc \mid A$

$S \rightarrow ABC$
$A \rightarrow c \mid b \mid aA'$
$A' \rightarrow B \mid C$
$B \rightarrow$

* CONSTRUSTION OF LL(1) PARSING TABLE

Page No.
Date : / /201

To construct LL(1) table follow the following step.

1) Check whether the given Grammar contain left recussion or Left factoring than eliminated first

NOTE Before Eliminating left factoring, left recursion should be eliminated FIRST.

2) Calculated first and follow set

3) Construct passing table

PARSING TABLE:

If the table contain multiple entries in a single cell then it is not LL(1) otherwise LL(1).

Q1. Check whether LL(1) or not

S → aAB | bA | ∈

A → aAb | ∈

B → bB | c

STEP 1.

there is no LR and LF

Date : / /201

STEP 2. Calculation of FIRST and FOLLOW SET

I) FIRST SET.

FIRST (B) = FIRST (b) ∪ FIRST (c)
= {b, c}

FIRST (A) = FIRST (a) ∪ FIRST (ϵ)
= {a, ϵ}

FIRST (S) = FIRST (a) ∪ FIRST (b) ∪ FIRST (ϵ)
= {a, b, ϵ}

II) FOLLOW SET.

FOLLOW (S) = {$}

FOLLOW (A) = FIRST (B) ∪ FOLLOW (S) ∪ FIRST (b)
= {b, c} ∪ {$} ∪ {b}
= {$, b, c}

FOLLOW (B) = FIRST (S) ∪ FIRST (B)
= {$}

STEP 3. Construction LL(1) parsing table.

V \ T	a	b	c	$
S	S → aAB	S → bA		S → ϵ
A	A → aAb	A → ϵ	A → ϵ	A → ϵ
B		B → bB	B → c	

Since table does not contain multiple entries Hence it is LL(1) Grammar.

9-10

* S → AaAb | BaBb
A → ∈
B → ∈

Ans Step 1:
There are no left recursion or left factoring

Step 2:
Calculation of FIRST and FOLLOW SET

FIRST SET
FIRST (B) = FIRST {∈}
= {∈}
FIRST (A) = FIRST {∈}
= {∈}
FIRST (S) = FIRST (AaAb) ∪ FIRST (BaBb)
= FIRST (A) − ∈ ∪ FIRST (aAb) ∪ FIRST (B) − ∈ ∪ FIRST (aBb)
= FIRST (∈) − ∈ ∪ FIRST (a) ∪ FIRST (∈) − ∈ ∪ FIRST (a)
= {a} ∪ {a}
= {a}

FOLLOW SET
FOLLOW(S) = {$}
FOLLOW (A) = FIRST (aAb) ∪ FIRST {b}
= {a} ∪ {b}
= {a, b}

FOLLOW (B) = FIRST (aBa) ∪ FIRST (b)
= {a} ∪ {b}
= {a, b}

NOT CORRECT

Page No.
Date: / /201

V/T	a	b	$
S	S → AaAb	S → BbBa	
A	A → ϵ	A → ϵ	
B	B → ϵ	B → ϵ	

Hence table does not contain multiple entries.

It is LL(1) Grammar

★ D → L:T
L → L, id | id
T → integer.

Ans: **STEP 1:**

Elimination of Left Recursion

D → L:T
L → idL'
L' → ,idL' | ϵ
T → integer

STEP 2:

Calculation of FIRST & FOLLOW SET

FIRST SET	FOLLOW SET
FIRST (T) = {integer}	FOLLOW SET
FIRST (L') = {, , ϵ}	FOLLOW (D) = {$}
FIRST (L) = {id}	FOLLOW (L) = {:}
FIRST (D) = {id}	FOLLOW (L') = {:}
	FOLLOW (T) = {$}

STEP 3:

LL(1) passing table.

V/T	id	integer	,	;	$
D	D→L;T				
L	id L'				
L'			L'→,idL'	L'→ϵ	
T		T→integer			

Since the Grammar does not contain multiple entries.
Hence the Grammar is LL(1)

4-15 $A \to aCDg \mid aBg \mid \epsilon$
$C \to \beta \mid \epsilon \mid Ct \mid BD \mid \gamma AB$
$D \to d \mid \epsilon$
$B \to e \mid \epsilon$

Ans. STEP1 Eliminate LR & LF
$A \to aCDg \mid aBg \mid \epsilon$
$C \to \beta C' \mid \epsilon C' \mid BDC' \mid \gamma ABC'$
$C' \to tC' \mid \epsilon$
$D \to d \mid \epsilon$
$B \to e \mid \epsilon$

Eliminate of LF

$A \to aA' | \epsilon$

$A' \to CDg | Bg$

$C \to pC' | c' | BDC' | rABC'$

$C' \to tC' | \epsilon$

$D \to d | \epsilon$

$B \to e | \epsilon$

STEP2. Calculate of FIRST & FOLLOW SET

* FIRST SET

FIRST (B) = {e, ∈}

FIRST (D) = {d, ∈}

FIRST (C') = {t, ∈}

FIRST (C) = FIRST (pC') ∪ FIRST (C') ∪ FIRST (BD
∪ FIRST (rABC')

= {p} ∪ {t, ∈} ∪ FIRST (B) − ∈ ∪ FIRST (D
∪ {r}

= {p, r, t, ∈} ∪ {e, ∈} − ∈ ∪ FIRST (D) − ∈ ∪
FIRST (C')

= {p, r, t, ∈} ∪ {e} ∪ {d, ∈} − ∈ ∪ FIRST (C')

= {p, r, t, ∈} ∪ {e} ∪ {d} ∪ {t, ∈}

= {d, e, p, r, t, ∈}

FIRST (A') = FIRST (CDg) ∪ FIRST (Bg)

= FIRST (C) − ∈ ∪ FIRST (Dg) ∪
FIRST (B) − ∈ ∪ FIRST (g)

= {d, e, p, r, t, ∈} − ∈ FIRST (D) − ∈ ∪
FIRST (g) ∪ {e, ∈} − ∈ ∪ {g}

= {d, e, p, r, t} ∪ {g → ∈} − ∈ ∪ {g} ∪ {e} ∪
{g}

= {d, e, g, p, q, r, t}

FIRST (A) = {q, ∈}

Page No.
Date : / /

⊛ FOLLOW SET

FOLLOW (A) = {$} ∪ FIRST (BC')
= {$} ∪ FIRST (B) - ∈ ∪ FIRST(C)
= {$} ∪ {e, ∈} - ∈ ∪ {t, ∈} - ∈ ∪ FOLLOW(
= {$, e, t} ∪ FOLLOW(C)
= {$, e, t} ∪ {d, q}

FOLLOW (A') = FOLLOW (A)
= {t, e, q, t, $}

FOLLOW (C) = FIRST (Dq)
= FIRST {D} - ∈ ∪ FIRST (q)
= {d, ∈} - ∈ ∪ {q}
= {d, q}

FOLLOW (C') = FOLLOW (C) ∪ FOLLOW (C) ∪
FOLLOW (C') ∪ FOLLOW (C) ∪
FOLLOW (C')
= FOLLOW (C)
= {d, q}

FOLLOW (D) = FIRST (q) ∪ FIRST (C')
= {q} ∪ {t, ∈} - ∈ ∪ FOLLOW (C)
= {q, t} ∪ {d, q}
= {d, q, t}

FOLLOW (B) = FIRST (q) ∪ FIRST (DC') ∪ FIRST (
= {q} ∪ FIRST (D) - ∈ ∪ FIRST (c) ∪
= {d - ∈} - ∈ ∪ FOLLOW (C)
= {q} ∪ {d - ∈} - ∈ ∪ {t, ∈} - ∈ ∪ FOLLOW (

Date : / /201

V \ T	a	d	e	g	p	q	r	t	$
A	A→aA'	A→ϵ	A→ϵ			A→ϵ		A→ϵ	A→ϵ
A'		A'→CDq	A'→Bg	A'→Bg	A'→CDq	A'→Cq	A'→CDq	A'→Cq	
C		C→ϵ C→BDC'	C→BDC'		C→pC'	C→ϵ	C→ABC'	C→C'	
C'		C'→ϵ				C'→ϵ		C'→tC'	
D		D→d D→ϵ				D→ϵ		D→ϵ	
B		B→ϵ	B→e	B→ϵ		B→ϵ		B→ϵ	

This table contain multiple entries.
Hence this Grammar is not LL(1).

S→iLqET|a
L→LS|ϵ
E→b
T→dLe|ϵ

STEP 1: Elimination of LR

S→iLqET|a
L→ϵL'
L'→SL'|ϵ
E→b
T→dLe|ϵ

S→iLqET|a
L→L'
L'→SL'|ϵ
E→b
T→dLe|ϵ

2 STEP 2: Calculate of FIRST & FOLLOW SET

FIRST SET

FIRST (T) = {d, ∈}

FIRST (E) = {b}

FIRST (L') = {a, i, ∈}

FIRST (L) = {i, ∈}

FIRST (S) = {i, a}

FOLLOW SET

FOLLOW (S) = {$} ∪ FIRST (L')

= {$} ∪ {a, i, ∈} - ∈ ∪ FOLLOW (L')

= {$} ∪ {a, i} ∪ FOLLOW (L')

= {$} ∪ {a, i} ∪ {e, u}

= {a, e, i, u, $}

FOLLOW (L) = FIRST (UET) ∪ FIRST (e)

= {e, u}

FOLLOW (L') = FIRST (i) ∪ FOLLOW (i)

= {e, u}

FOLLOW (E) = FIRST (T)

= {d, ∈} - ∈ ∪ FOLLOW (S)

= {d} ∪ FOLLOW (S)

= {a, d, e, i, o, $}

FOLLOW (T) = FOLLOW (S)

= {a, e, i, u, $}

STEP 3:

LL(1) parsing table

V\T	a	b	d	e	i	u	$
S	S→a				S→iLuET		
L	L→L'			L→ϵ	L→L'	L→ϵ	
L'	L'→SL'			L'→ϵ	L'→SL'	L'→ϵ	
E		E→b					
T	T→ϵ		T→dLe	T→ϵ	T→ϵ	T→ϵ	T→ϵ

Since the table contain single entries for each column Hence the Grammar is LL(1).

★ Construct LL(1) passing table for the Grammar

E → E+T | T
T → T* F | F
F → (E) | id

Show the moves made by the Parser for the string

ω = id+id* id

Ans. STEP 1: Elimination of LR and LF

LR Elimination

E → TE'
E' → +TE' | ϵ
T → FT'
T' → * FT' | ϵ
F → (E) | id

2 STEP 2: Find FIRST & FOLLOW SET

FIRST SET

FIRST (F) = { c, id }
FIRST (T') = { *, ϵ }
FIRST (T) = { c, id }
FIRST (E') = { +, ϵ }
FIRST (E) = { c, id }

FOLLOW SET

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

3 STEP 3. LL(1) parsing table

V\T	+	*	(	)	id	$
E			E→TE'		E→TE'	
E'	E'→+TE'			E→ϵ		E→ϵ
T			T→FT'		T→FT'	
T'	T'→ϵ	T'→*FT		T'→ϵ		T→ϵ
F			F→(E)		f→id	

④ Validation / Verification / Action

Page No.
Date : / /201

STACK	INPUT	MOVES
$E	id + id * id $	E → TE'
$E'T	id + id * id $	T → FT'
$E'T'F	id + id * id $	F → id
$E'T'id	id + id * id $	popping id
$E'T'	+ id * id $	T' → ϵ
$E'	+ id * id $	E' → +TE'
$E'T+	+ id * id $	popping +
$E'T	id * id $	T → FT'
$E'T'F	id * id $	F → id
$E'T'id	id * id $	popping id
$E'T'	* id $	T' → *FT'
$E'T'F*	* id $	popping *
$E'T'F	id $	F → id
$E'T'id	id $	popping id
$E'T'	$	T' → ϵ
$E'	$	E' → ϵ
$	$	

* id + id + id
* id * id * id

8m

W-17 Q. Design LL(1) passer for the given Grammar

$S \rightarrow UVW$
$U \rightarrow (S) \mid aSb \mid d$
$V \rightarrow aV \mid \epsilon$
$W \rightarrow cW \mid \epsilon$

Also, give passing action for i/p string "(d(a"

Ans **STEP 1:** Eliminate left recursion and Left factoring

$S \rightarrow UVW$
$U \rightarrow (S) \mid aSb \mid d$
$V \rightarrow aV \mid \epsilon$
$W \rightarrow cW \mid \epsilon$

No left recursion & left factoring.

STEP 2: Find the FIRST & FOLLOW SET

FIRST SET
FIRST (W) = {c, ε}
FIRST (V) = {a, ε}
FIRST (U) = {(, a, d}
FIRST (S) = FIRST (U)
= {(, a, d}

follow S =), b, $
follow U = a, c,), b, $
follow V = c, b,), $
follow W = b,), $

FOLLOW SET
FOLLOW (S) = {$} ∪ {b} ⇒ {b,), $}
FOLLOW (U) = {a, b, c,), $}
FOLLOW (V) = {b, c, $,)}
FOLLOW (U) = FOLLOW (S)

STEP 3: LL(1) passing table

V/T	a	b	c	d	(	)	$
S	S→UVW			S→UVW	S→UVW		
U	U→aSb			U→d	U→(S)		
V	V→aV	V→ϵ	V→ϵ			V→ϵ	V→ϵ
W		W→ϵ	W→cW			W→ϵ	W→ϵ

STEP 4: action table

STACK	INPUT	MOVES
$S	(dc)ac$	S→UVW
$WVU	(dc)ac$	U→(S)
$WV)S(	(dc)ac$	popping (
$WV)S	dc)ac$	S→UVW
$WV)WVU	dc)ac$	U→d
$WV)WVd	dc)ac$	popping d
$WV)WV	c)ac$	V→ϵ
$WV)W	c)ac$	W→cW
$WV)Wc	c)ac$	popping c
$WV)W	)ac$	W→ϵ
$WV)	)ac$	popping)
$WV	ac$	V→aV
$WVa	ac$	popping a
$WV	c$	V→ϵ
$W	c$	W→cW
$Wc	c$	popping c
$W	$	W→ϵ
$	$	Accepted

6M

Q 317 What is significance of FIRST & FOLLOW in top down parser "A ϵ-free $\rightarrow$ LL(1) parsing table without follow set" Justify the statement.

Ans Justification: Consider a Grammar which is ϵ-free

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a$

FIRST SET

FIRST (B) = FIRST (b)

= {b}

FIRST (A) = {a}

FIRST (S) = FIRST (A)

= {a}

LL(1) parsing table

V \ T	a	b	$
S	S→AB		
A	A→a		
B		B→a	

Hence the Grammar is LL(1) as it does not contain multiple entries.

Hence this statement of ϵ-free grammar $\rightarrow$ LL(1) parsing table can be constructed without follow set.

Q Check whether the given grammar is LL(1) or not without using LL(1) parsing table

$S \rightarrow aAB \mid bA \mid \epsilon$
$A \rightarrow aAb \mid \epsilon$
$B \rightarrow bB \mid c$

Ans **STEP 1.**
There are no left recursion and No left factoring

Step 2.
Calculation of FIRST & FOLLOW SET

1) FIRST SET
FIRST (B) → {b, c}
FIRST (A) → {a, ∈}
FIRST (S) → {a, b, ∈}

2) FOLLOW SET
FOLLOW (S) = {$}
FOLLOW (A) = {b, c, $}
FOLLOW (B) = FOLLOW (S) ∪ FOLLOW (B)
= FOLLOW (S)
= {$}

for each and every Non-terminal symbol (V)
If FIRST (X) = ∈ &
FIRST (X) ∩ FOLLOW (X) = ϕ
then LL(1)
else
NOT LL(1).

BOTTOM UP PARSER

Date: / /20

1) Simple Left to Right (SLR)/LR(0)

To design LR(0) passer follow the following steps

STEP 1. Augumented Grammar

STEP 2. Calculation of FIRST & FOLLOW SET

STEP 3. Transition Table

STEP 4. LR(0) Diagram

1) Augumented GRAMMER

eg ①

$S \rightarrow AB$

$A \rightarrow a|b$

$B \rightarrow c$

Aus: $S' \rightarrow .S$

$S \rightarrow .AB$

$A \rightarrow .a$

$A \rightarrow .b$

$B \rightarrow .c$

Top down have drawbacks like LR & LR

- is use to know the scan status.

Page No.
Date : / /201

eg.

$E \rightarrow E+E \mid E*E \mid id$

$E' \rightarrow .E$

$E \rightarrow .E+E$

$E \rightarrow .E*E$

$E \rightarrow .id$

③ Transition Diagram:

Symbol → Non-terminal (V) → generate

→ Terminal (T) → Terminate.

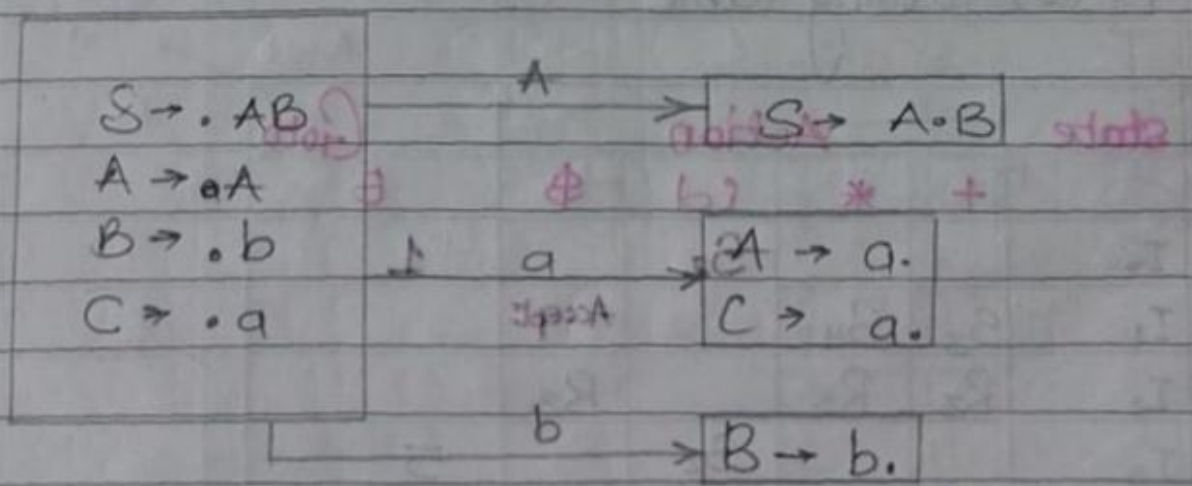
eg.

$S \rightarrow .AB$

$A \rightarrow .a$

$B \rightarrow .b$

$C \rightarrow .a$



④ LR(0) parsing table

If table contain shift (S)/ reduced (R) or Reduced (R)/Reduced (R) conflict then it is not LR(0). Otherwise LR(0).

Question: Check whether the given Grammar is LR(0) or not

$E \rightarrow E+E \mid E*E \mid id$

Ans: Step 1: Augumented Grammar

$E' \rightarrow .E$ ——— (0)

$E \rightarrow .E+E$ ——— (1)

$E \rightarrow .E*E$ ——— (2)

$E \rightarrow .id$ ——— (3)

Step 2: Calculation of FIRST & FOLLOW SET

(I) FIRST SET

FIRST (E) = {id}

(II) FOLLOW SET

FOLLOW (E) = {+, *, $}

Step 4 LR(0) Parsing table

terminal in Action

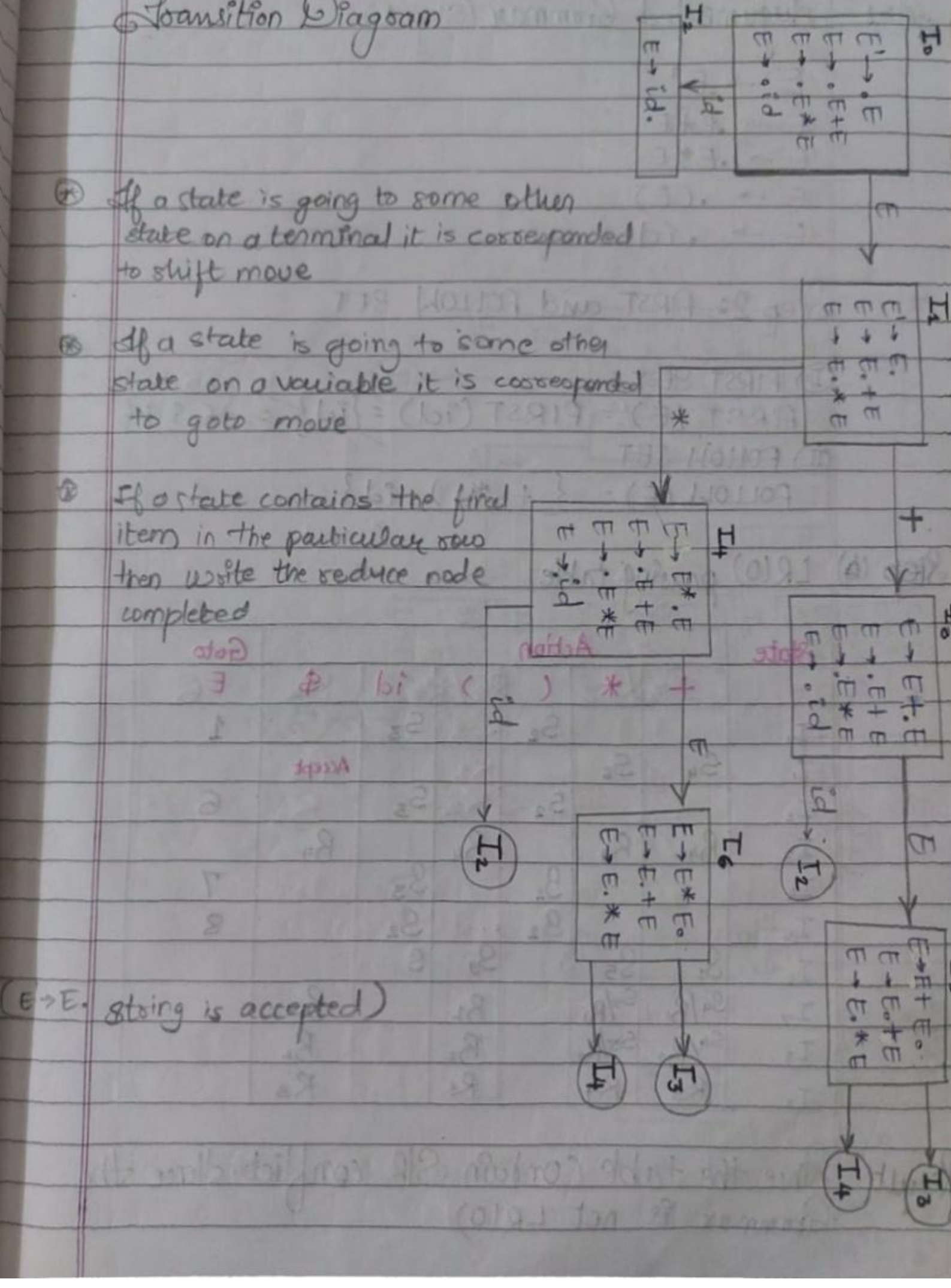
Non-terminal in Goto

state	Action				Goto
	+	*	id	$	E
I_0			S_2		1
I_1	S_3	S_4		Accept	
I_2	R_3	R_3		R_3	
I_3			S_2		5
I_4			S_2		6
I_5	S_3/R_1	S_4/R_1		R_1	
I_6	S_3/R_2	S_4/R_2		R_2	

Result: Since table contain S/R conflict. Hence the

STEP 3:

Transition Diagram



① If a state is going to some other state on a terminal it is corresponded to shift move

② If a state is going to some other state on a variable it is corresponded to goto move

③ If a state contains the final item in the particular rule then write the reduce node completed

(E → E. string is accepted)

Question $E \rightarrow E+E \mid E*E \mid (E) \mid id$

Page No.
Date: / /

Augumented Grammar (Step 1)

$E \rightarrow .E'$
$E \rightarrow .E+E$
$E \rightarrow .E*E$
$E \rightarrow .(E)$
$E \rightarrow .id$

Step 2: FIRST and FOLLOW SET

(I) FIRST SET

FIRST (E) = FIRST (id) = ~~{id}~~ = {(, id}

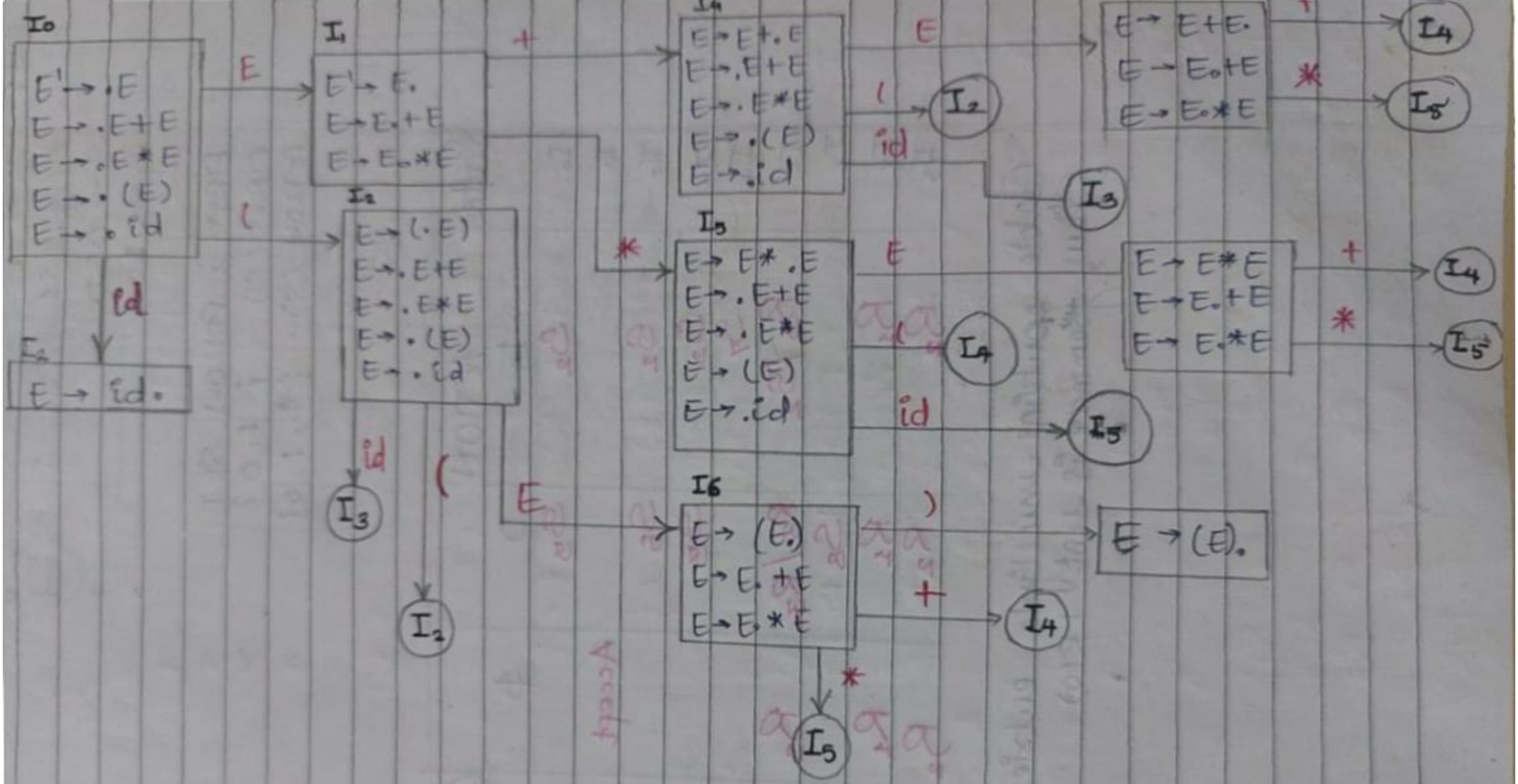
(II) FOLLOW SET

FOLLOW (E) = { +, *,), id }

Step (3) LR(0) parsing table

State	Action						Goto
	+	*	(	)	id	$	E
I_0			S_2		S_3		1
I_1	S_4	S_6				Accept	
I_2			S_2		S_3		6
I_3	R_4	R_4		R_4		R_4	
I_4			S_2		S_3		7
I_5			S_2		S_2		8
I_6	S_1	S_5		S_9			
I_7	S_4/R_1	S_5/R_1		R_1		R_1	
I_8	S_4/R_2	S_5/R_2		R_2		R_2	
I_9	R_3	R_4		R_3		R_3	

Since the table contain S/R conflict. Hence the Grammar is not LR(0)


I0
E' → .E
E → .E+E
E → .E*E
E → .(E)
E → .id
E
I1
E' → E.
E → E.+E
E → E.*E
+
I4
E → E+.E
E → .E+E
E → .E*E
E → .(E)
E → .id
(
I2
id
I3
I5
E → id.
id
I3
(
I2
I6
E → (.E)
E → .E+E
E → .E*E
E → .(E)
E → .id
E → (E.)
E → E.+E
E → E.*E
)
E → (E).
E → E+E.
E → E.+E
E → E.*E
*
I5
E → E*E.
E → E.+E
E → E.*E

3 | $S \rightarrow 1S1 | 0S0 | 10$

Page No. 63
Date : / /

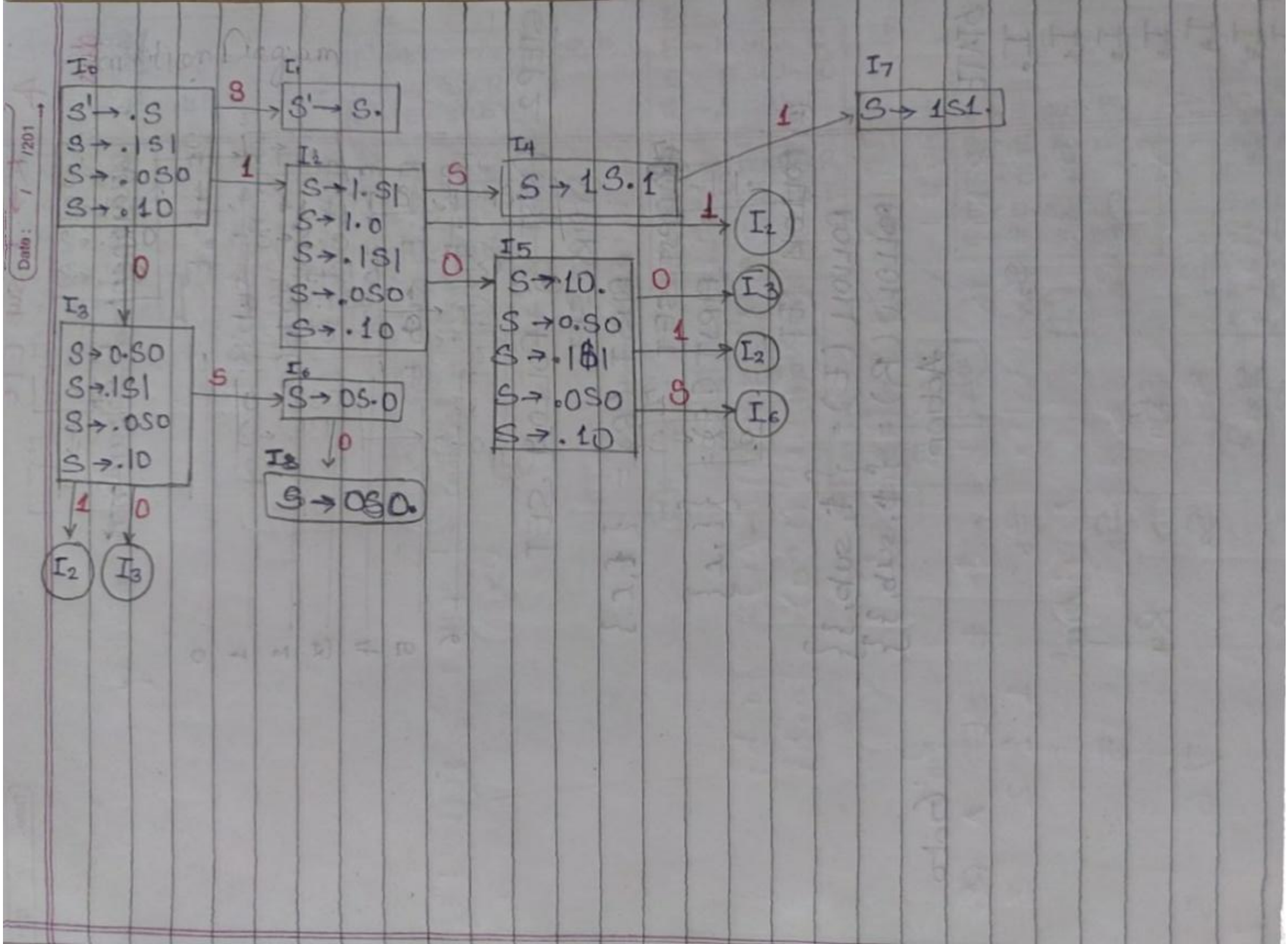
dus:

$S \rightarrow .S'$ ——— 0
$S \rightarrow .1S1$ ——— 1
$S \rightarrow .0S0$ ——— 2
$S \rightarrow .10$ ——— 3

FIRST & FOLLOW SET
FIRST (S) = { 1, 0 }
FOLLOW (S) = { $, 1, 0 }

States	ACTION			GOTO
	1	0	$	S
I_0	S_2	S_3		1
I_1			Accept	
I_2	S_2	S_5		4
I_3	S_2	S_3		6
I_4	S_7			
I_5	R_3 / S_2	R_3 / R_3	R_3	6
I_6		S_8		
I_7	R_1	R_1	R_1	
I_8	R_2	R_2	R_2	

∴ Table contains multiple entries
Thus grammar is not LR(0)


I0
S'→ .S
S→ .1S1
S→ .0S0
S→ .10
S
I1
S'→ S.
1
I2
S→ 1.S1
S→ 1.0
S→ .1S1
S→ .0S0
S→ .10
S
S→ 1S.1
1
I7
S→ 1S1.
1
I2
0
I3
S→ 0.S0
S→ .1S1
S→ .0S0
S→ .10
S
I6
S→ 0S.0
0
I8
S→ 0S0.
I4
S→ 1S.1
I5
S→ 10.
S→ 0.S0
S→ .1S1
S→ .0S0
S→ .10
0
I3
1
I2
S
I6
1
0
I2
I3

E → E sub R | E sub E | (E) | c

R → E sub E | E

6 4

1 dus

STEP 1 Augumented Goammar

E' → .E ——— 0
E → .E sub R ——— 1
E → .E sub E ——— 2
E → .{E} ——— 3
E → .c ——— 4
R → .E sub E ——— 5
R → .E ——— 6

STEP 2 FIRST & FOLLOW SET

① FIRST SET

FIRST (R) = {{, c}

② FIRST SET

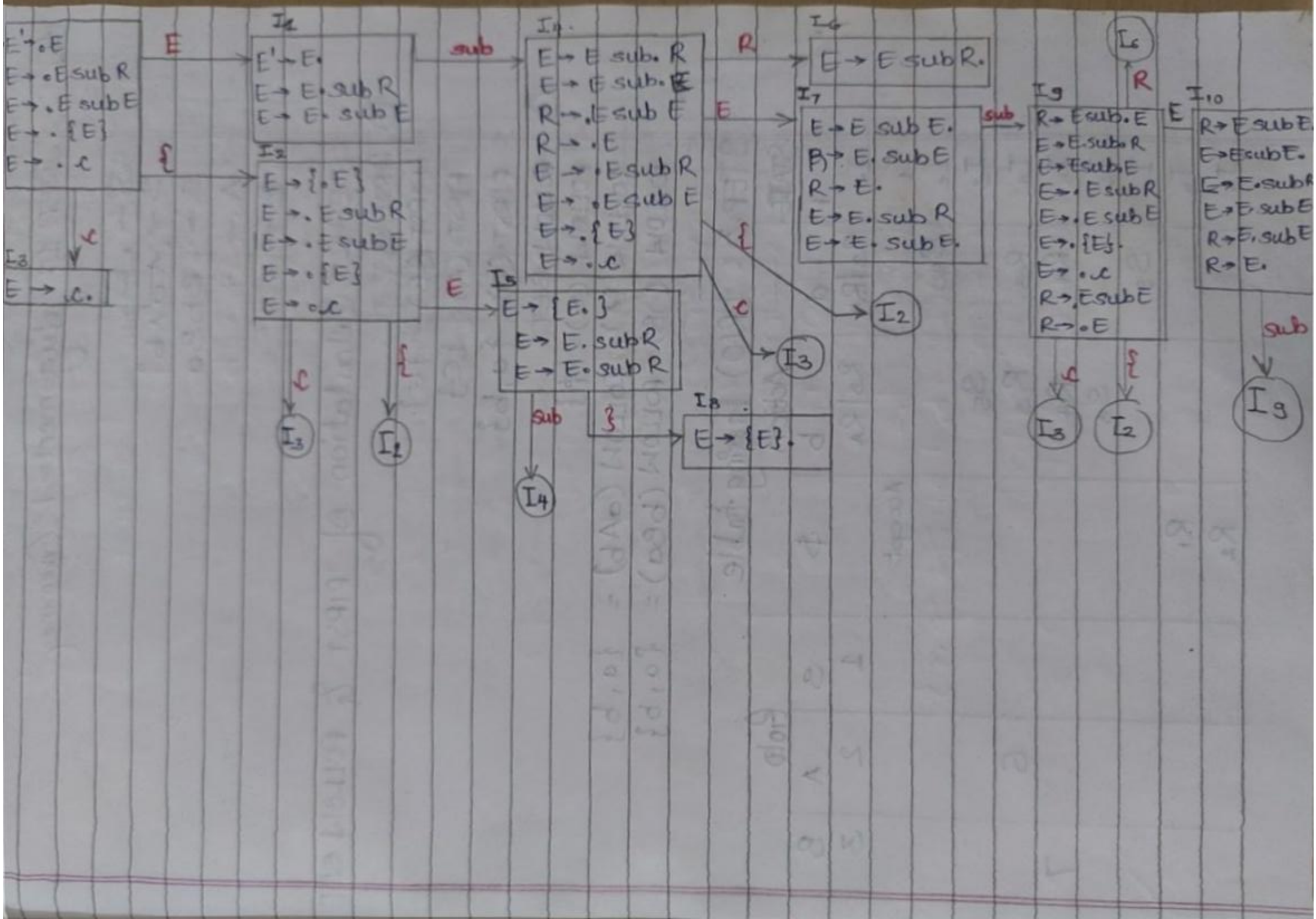
→ FIRST (E) = {{, c}

② FOLLOW SET

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

FOLLOW (R) = {$, sub, }}

	Action					Goto	
STATE	sub	{	}	c	$	E	R
I_0		S_2		S_3		1	
I_1	S_4				accept		
I_2		S_2		S_3		5	
I_3	R_3		R_4		R_4		
I_4		S_2		S_3		7	6
I_5	S_4		S_8				
I_6	R_1		R_1		R_1		6
I_7	$S_4/R_2/R_2$		R_2/R_6		R_2/R_6		
I_8	R_3		R_3		R_3		


I0
E'→.E
E→.E sub R
E→.E sub E
E→.{E}
E→.c
E
{
c
I3
E→c.
I1
E'→E.
E→E.sub R
E→E.sub E
I2
E→{.E}
E→.E sub R
E→.E sub E
E→.{E}
E→.c
sub
I4
E→E sub.R
E→E sub.E
R→.E sub E
R→.E
E→.E sub R
E→.E sub E
E→.{E}
E→.c
R
E
{
c
I5
E→{E.}
E→E.sub R
E→E.sub R
sub
}
I6
E→E sub R.
I7
E→E sub E.
R→E sub E.
R→E.
E→E.sub R
E→E.sub E
I8
E→{E}.
I9
R→E sub.E
E→E.sub R
E→E sub E
E→.E sub R
E→.E sub E
E→.{E}
E→.c
R→E sub E
R→.E
I10
R→E sub E.
E→E sub E.
E→E.sub R
E→E sub E
R→E.sub E
R→E.

5 $S \to AaAb \mid BbBa$
$A \to \epsilon$
$B \to \epsilon$

Page No.
Date : / /

Sol. STEP 1: Augumented Grammar

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

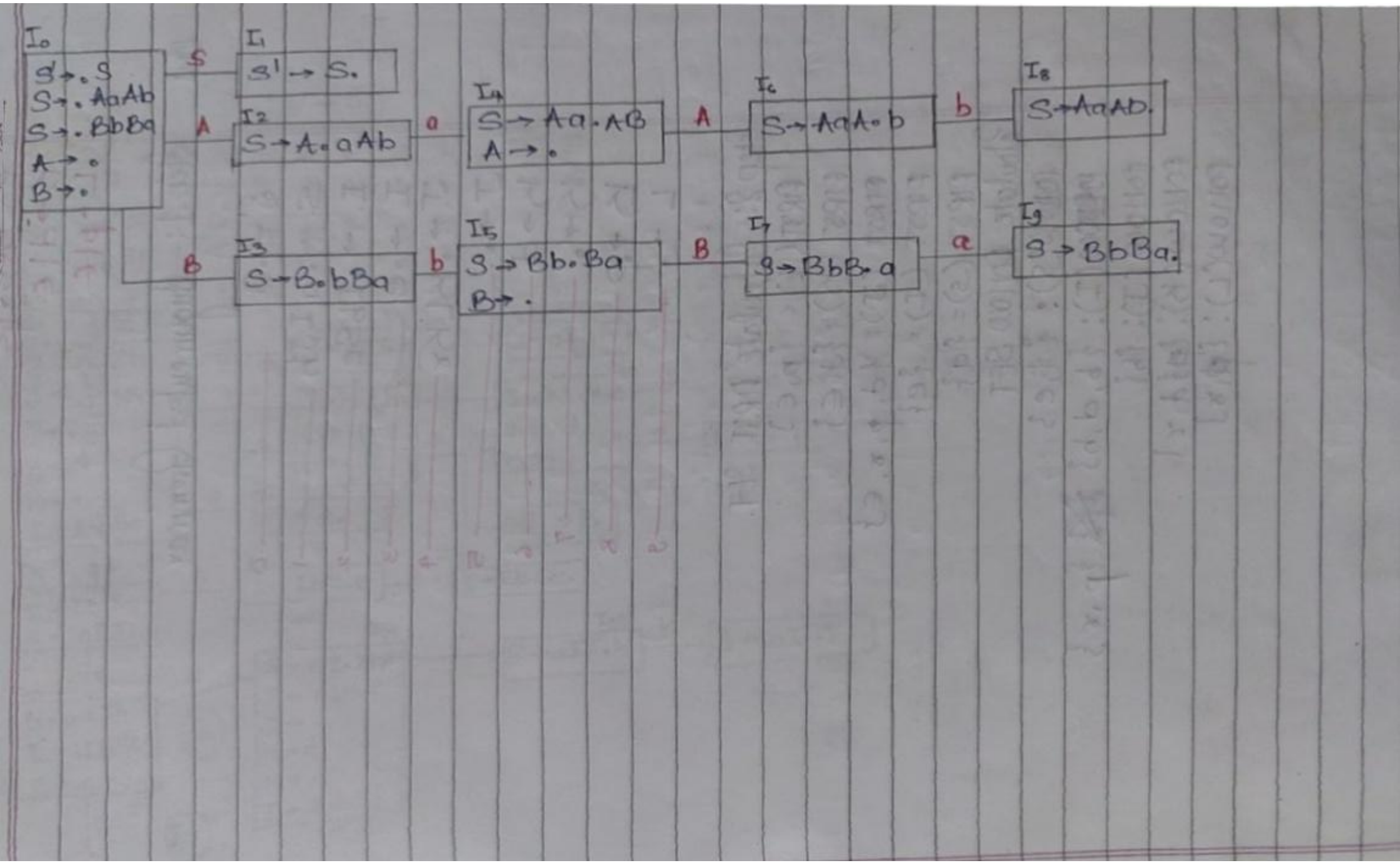
STEP 2: Calculation of FIRST & FOLLOW SET

① FIRST SET
FIRST (B) = {ϵ}
FIRST (A) = {ϵ}
FIRST (S) = {a, b}

② FOLLOW SET
FOLLOW (S) = {$}
FOLLOW (A) = FOLLOW (aAb) = {a, b}
FOLLOW (B) = FOLLOW (bBa) = {a, b}

STEP 3: LR(0) Parsing table

STATE	Action			Goto		
	a	b	$	S	A	B
I_0	$R_3 \mid R_4$	$R_3 \mid R_4$		1	2	3
I_1			Accept			
I_2	S_4					
I_3		S_5				
I_4	R_3	R_3			6	
I_5	R_4	R_4				7
I_6		S_8				
I_7	S_9					


I0
S' → .S
S → .AaAb
S → .BbBa
A → .
B → .
S
I1
S' → S.
A
I2
S → A.aAb
a
I4
S → Aa.AB
A → .
A
I6
S → AaA.b
b
I8
S → AaAb.
B
I3
S → B.bBa
b
I5
S → Bb.Ba
B → .
B
I7
S → BbB.a
a
I9
S → BbBa.

HM

I → IbSe | ε
J → KLKγ | ε
K → d | ε
L → p | ε

Step 1: Augumented Grammar

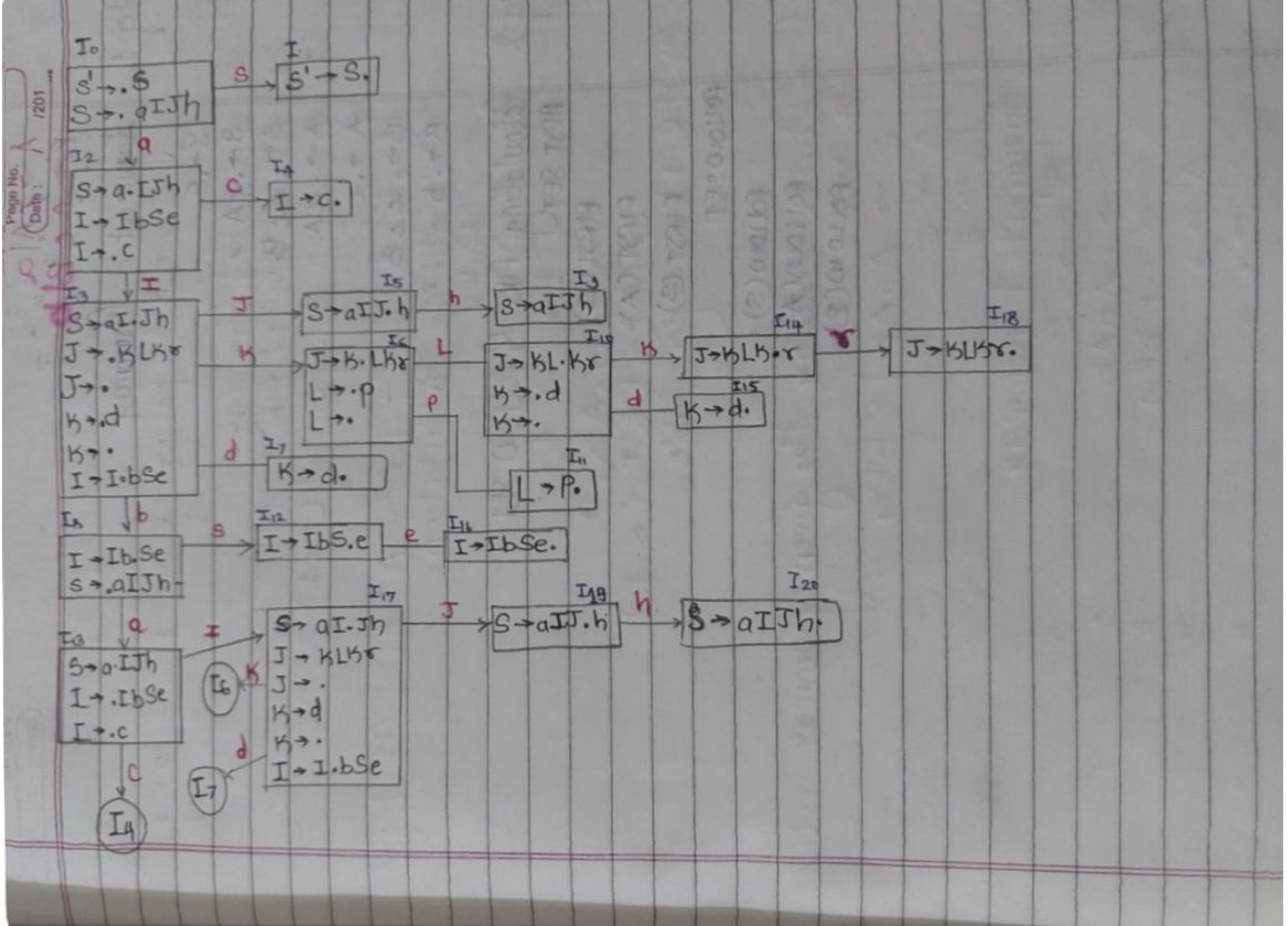
S' → .S ——— 0
S → .aIJh ——— 1
I → .IbSe ——— 2
I → .e ——— 3
J → .KLKγ ——— 4
J → . ——— 5
K → d ——— 6
K → . ——— 7
L → .p ——— 8
L → . ——— 9

Step 2: Calculate FIRST SET.

FIRST(L) = {p, ε}
FIRST(K) = {d, ε}
FIRST(J) = {d, p, γ, ε}
FIRST(I) = {e}
FIRST(S) = {a}

Calculate FOLLOW SET

FOLLOW(S): {$, e}
FOLLOW(I): {b, d, h} ~~{p, γ}~~
FOLLOW(J): {h}
FOLLOW(K): {p, d, γ}
FOLLOW(L): {d, γ}


I0
S'→.S
S→.aIJh
S
I
S'→S.
a
I2
S→a.IJh
I→IbSe
I→.c
c
I4
I→c.
I3
S→aI.Jh
J→.KLKγ
J→.
K→.d
K→.
I→I.bSe
I
J
K
d
I5
S→aIJ.h
I6
J→K.LKγ
L→.p
L→.
I7
K→d.
h
I9
S→aIJh
L
p
I10
J→KL.Kγ
K→.d
K→.
I11
L→p.
K
d
I14
J→KLK.γ
I15
K→d.
γ
I18
J→KLKγ.
I8
b
I→Ib.Se
S→.aIJh
S
I12
I→IbS.e
e
I16
I→IbSe.
a
I3
S→a.IJh
I→.IbSe
I→.c
I
K
d
c
I4
I6
I7
I17
S→aI.Jh
J→KLKγ
J→.
K→d
K→.
I→I.bSe
J
I19
S→aIJ.h
h
I20
S→aIJh.

JM

$S \to cA \mid ccB$

$A \to cA \mid a$

$B \to ccB \mid b$

Step 1. Augumented Grammar

$S' \to .S$

$S \to .cA$

$S \to .ccB$

$A \to .cA$

$A \to .a$

$B \to .ccB$

$B \to .b$

Step 2 Calculating FIRST & FOLLOW SET

FIRST SET

FIRST(B) =

FIRST(A) =

FIRST(S) =

FOLLOW SET

FOLLOW(S) =

FOLLOW(A) =

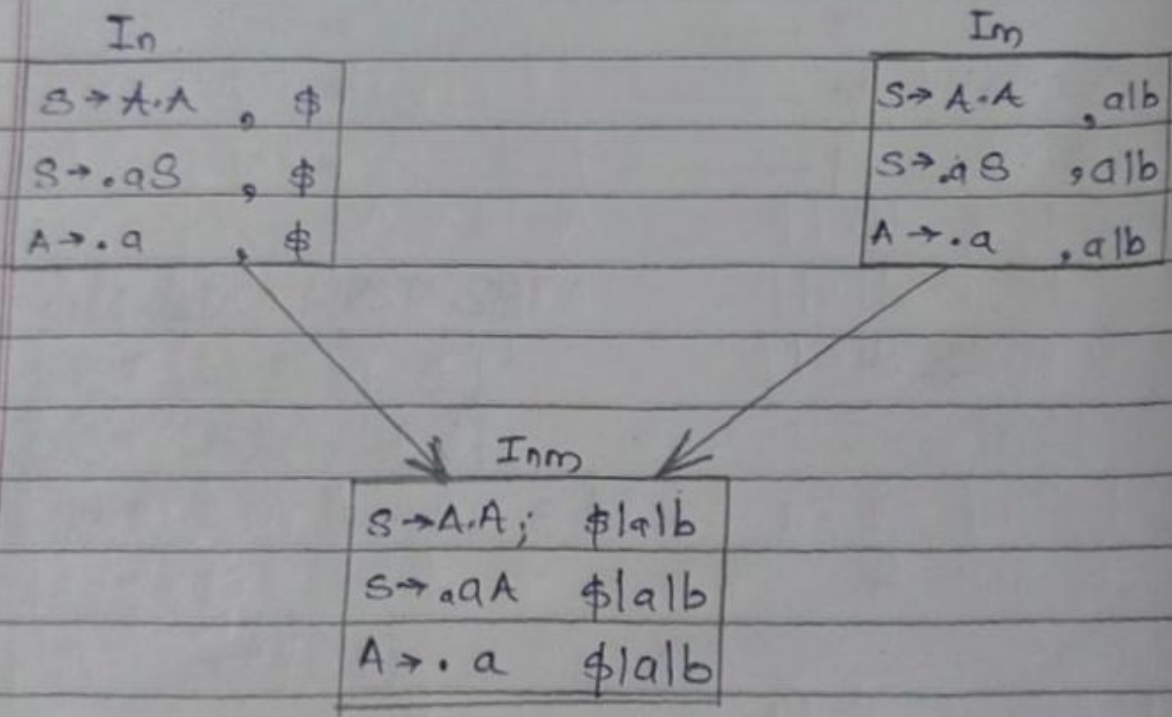
FOLLOW(B) =

Look Ahead LEFT TO RIGHT (LALR) Parser

To design LALR parser follow the following steps

Step 1: Design LR(1) parser

Step 2: from constructed transition diagram identify those states which have similar production rule but different LOOK AHEAD shown in the figure



Step 3. Transition Diagram of LALR

Step 4. LALR parsing table

Page No.
Date : / /201

* Design LALR parser

1. $S \to AA$
   $A \to aA \mid b$

Ans. Step 1) Design LR(1) parser

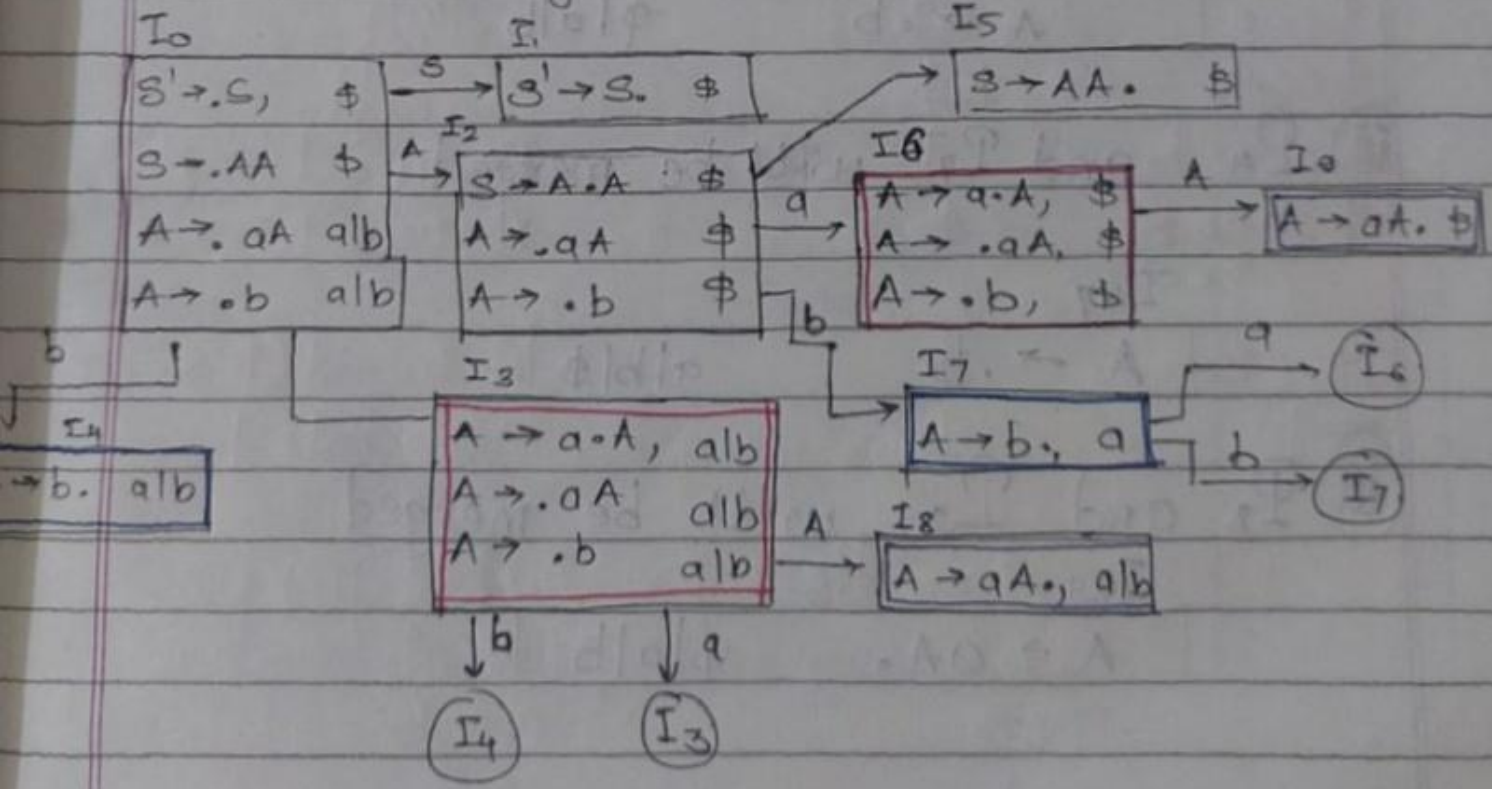
() dugumented Grammar.

$S' \to .S$ ——— 0
$S \to .AA$ ——— 1
$A \to .aA$ ——— 2
$A \to .b$ ——— 3

Calculate FIRST SET

FIRST (A) = {a, b}
FIRST (S) = {a, b}

Transition diagram



state	Action			Goto	
	a	b	$	S	A
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_4	S_7			5
I_3	S_3	S_4			8
I_4	R_3	R_3			
I_5			R_2		9
I_6	S_6	S_7			
I_7					
I_8	R_2	R_2			
I_9			R_2		

(2) (i) I_3 and I_6 will be merged

I_{36}

$A \to a.A$	$\$\|a\|b$
$A \to .aA$	$\$\|a\|b$
$A \to .b$	$\$\|a\|b$

(ii) I_4 and I_7 will be merged

I_{47}

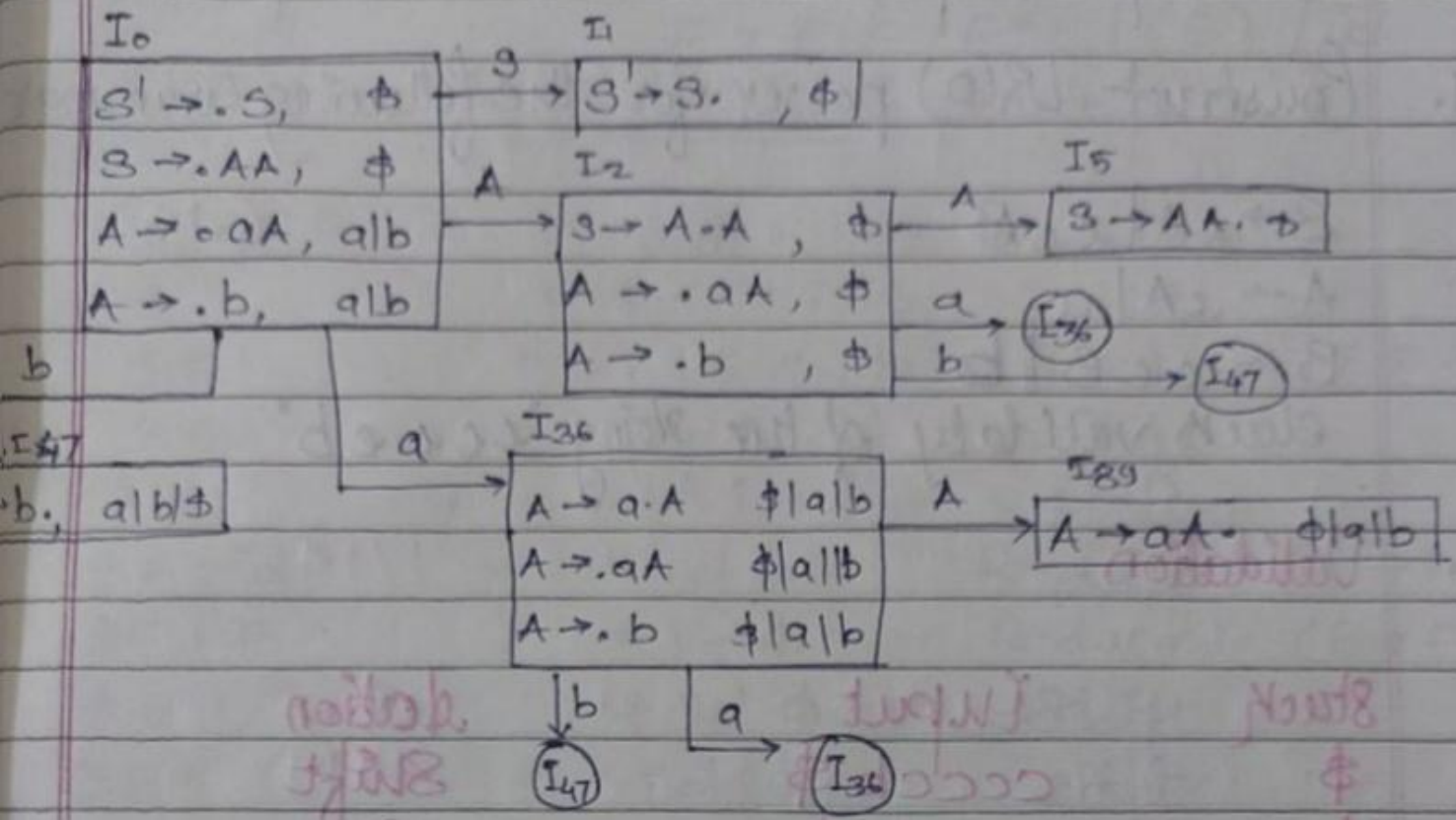
$A \to .b$	$a\|b\|\$$

(iii) I_8 and I_9 will be merged

I_{89}

$A \to aA.$	$\$\|a\|b$

Transition Diagram of LALR



LALR parsing table

State	action			goto	
	a	b	$	S	A
I_0	S_{36}	S_{47}		1	2
I_1			Accept		
I_2	S_{36}				5
I_{36}	S_{36}				89
I_{47}	R_3	R_3	R_3		
I_5			R_1		
I_{89}	R_2	R_2	R_2		

Shift Reduce Technique / Stack Implementation Technique.

1. Construct LR(0) passer for the following Grammar

$S \to cA \mid ccB$
$A \to cA \mid a$
$B \to ccB \mid b$
check validity of the string "ccccb"

2. Validation

Stack	Input	Action
$	ccccb $	shift
$c	cccb $	shift
$cc	ccb$	shift
$ccc	cb$	shift
$cccc	b$	shift
$ccccb	$	Reduced (B→b)
$ccccB	$	Reduced (B→ccB)
$ccB	$	Reduced ($→ccB)
S$	$	Validated

Handle	Viable prefix
b	ccccb
ccB	cccB
ccB	ccB

Top-Down CFG derive ↓ string | Bottom-up CFG ↑ reduce string.

IBM 711

Page No.

Date : / /201

2. Design LR(0) passer for
$E \to E+E \mid E*E \mid (E) \mid id$
Validate "id +id * id" check?

Stack	Input	Action
$	id+ id* id$	shift
$id	+id* id$	Reduced ($E \to id$)
$E	+id* id$	Shift
$E+	id* id$	shift
$E+id	* id $	Reduced ($E \to id$)
$E+E	* id $	Reduced ($E \to E+E$)
$E	* id $	Shift
$E*	id $	shift
$E*id	$	Reduced $E \to id$
$E*E	$	Reduced ($E \to E*E$)
$E	$	Validate

Handle	Viable prefix
id	id
id	E+id
E+E	E+E
id	E*id
E*id	id
E*E	E*E

Handle: Reduce entries only (right Hand side)
Viable prefix: content of stack

Unit III Syntax Directed Translation Schemes (SDT)

TOPICS

1. Syntax Directed Transition (SDT)
2. Syntax Directed Defination (SDD) - 6M
3. Implementation of SDT
4. Intermediate code Representation
5. Intermediate code generation using SDTS for
   i) Control structure - 7M
   ii) Declaration
   iii) Procedure Call
   iv) Array Reference (14M)

1. Syntax Directed Translation: deals with translation of language by using Gram

Syntax Directed translation (SDT) deals with the translation of program language by using Grammar (CFG) (CFG)

To perform various operation on a language

i) Change language into computer executable form

ii) Perform various action like.

(I) INFIX to POSTFIX notation

(II) Immediate code representation (TAC)

(III) Type Checking

attb
fix: Human understable
fix: computer understable

Page No.
Date: / /201

(i) (SDD) Syntax Directed Defination

SDD = CFG + {Sementics Rules}

CFG: Specify the syntatic structure of programming language

Sementics Rules: to evaluate the value of Grammar Symbol

Consider Syntax directed Defination of mathematical expression

$E \rightarrow E+T$	$\{E.val = E_1.val + T.val\}$
T	$\{E.val = T.val\}$
$T \rightarrow T * F$	$\{T.val = T_1.val * f.val\}$
f	$\{T.val = f.val\}$
$f \rightarrow id$	$\{f.val = id.Lval\}$ {Lexical value}

2+3*4

CFG + Sementics Values

for example 2+3*4

E.val = (2+12 = 14)

E1.val = 2 | + | T.val = 12

T.val = 2 | T.val = 3 | * | f.val = 4

f.val = 2 | f.val = 3 | id (1.val

id (L.val = 2) | id.Lval = 3 | (4)

(2) | (3)

if more than one variable with same letter rename them with adding suffix

(.) Dot mean attributes

Topic related to SDD

TYPES OF ATTRIBUTES

Page No.
Date: / /

There are two types of attributes

1. Synthesized Attributes.
2. Inherited attributes.

1. Synthesized attributes.

If the value of attributes at parse tree nodes determine its children node then attributes are referenced as synthesized attributes

* Synthesized attributes Example

$$E \longrightarrow E + T \mid T$$
$$T \longrightarrow T * F \mid F$$
$$F \longrightarrow id$$

SDD = CFG + {Semantic Rules}

$E \to E + T$	$\{ E.val = E_1.val + T.val \}$
$E \to T$	$\{ E.val = T.val \}$
$T \to T * F$	$\{ T.val = T_1.val * F.val \}$
$T \to F$	$\{ T.val = F.val \}$
$F \to id$	$\{ F.val = id.Lval \}$

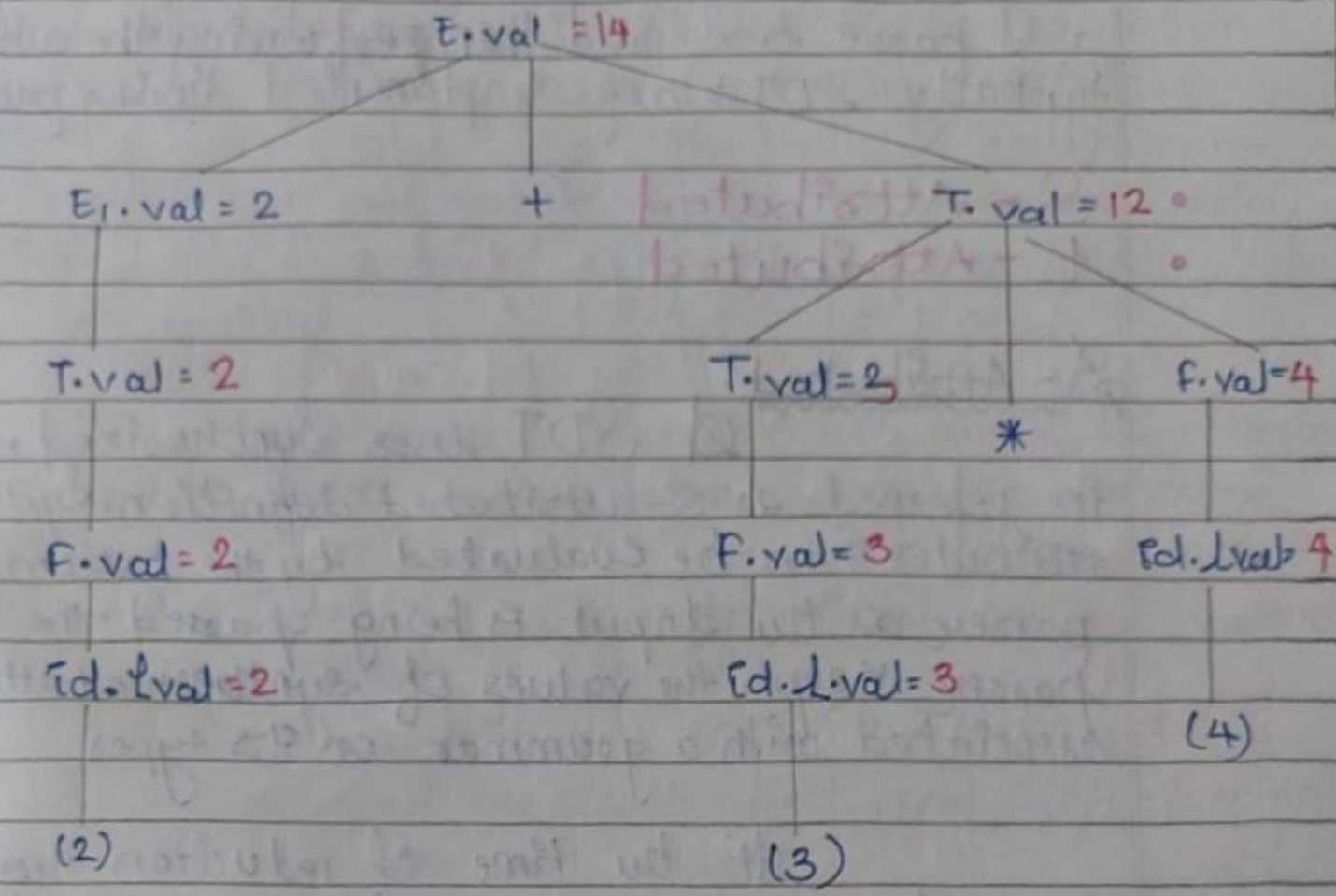
A → BCD

↓

E can only take values from my child production

→ Parent is dependent on child for values

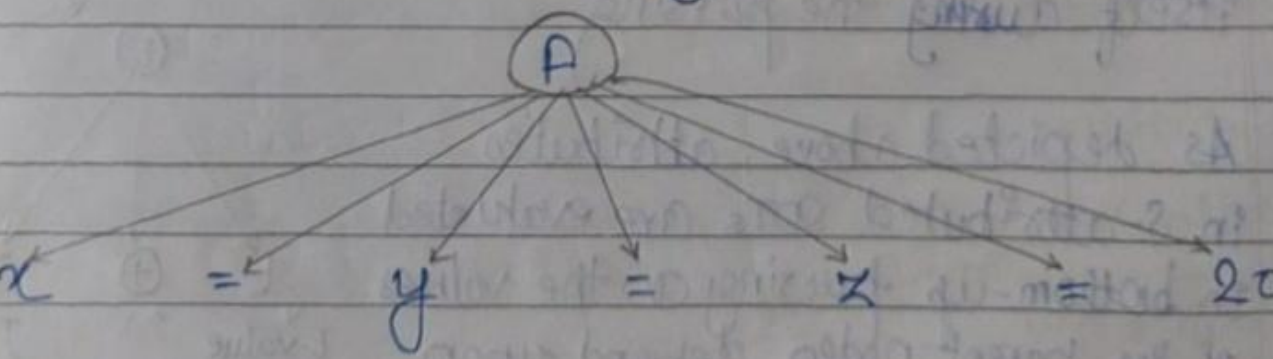
To evaluate : 2 + 3 × 4



2. INHERITED ATTRIBUTES

If the value of attribu
at parse tree node determines its parent
or sibling are defined as INHERITED Attribut

$P \rightarrow x = y = z = 20$ (R.HS)



(Higher to lower) values can be given from the siblings

Date: / /201

L-Attributed

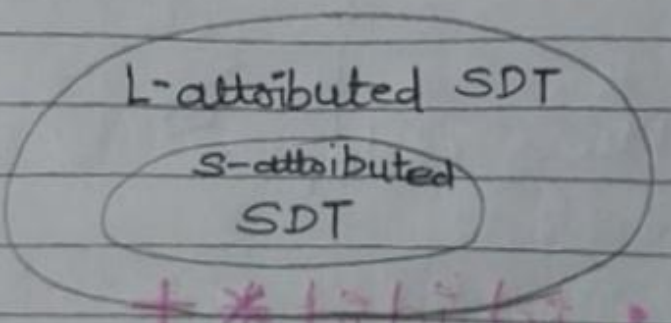
This form of SDT uses both synthesized and inherited attributes with restriction of not taking values from right siblings.

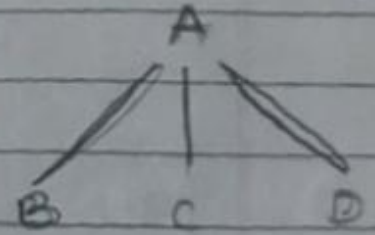
In L-attributed SDTs, a non-terminal can get values from its parent, child, and sibling nodes. As in the following production

S → ABC

S can take values from A, B, and C (Synthesized). A can take values from S only. B can take values from S and A. C can get values from S, A, and B. No non-terminal can get values from the sibling to its right.

Attributes in L-Attributed SDTs are evaluated by depth-first and left-to-right parsing manner





if i wish to get C's value in have to go up to A & then two value B which are already attain

We may conclude that if a definition is S-attributed, then it is also L-attributed as L-attributed definition encloses S-attributed definitions

S-attributed SDT

→ Synthesized attribute usage

→ parent shall take children value

Bottom up passing.

Semantic action position can vy be at the end of production

L-attributed SDT

- Both Synthesized & inherited attributes with restriction (left side only)

- semantic action (position can be anywhere)

A → B {C} [D]

→ can only consider values on left i.e. B, A

- Depth first left to right

3. IMPLEMENTATION OF SDTS:

Any SDT can be implemented by first building a parse tree and then ~~performing~~ the actions. Typically, SDTs are implemented during parsing.

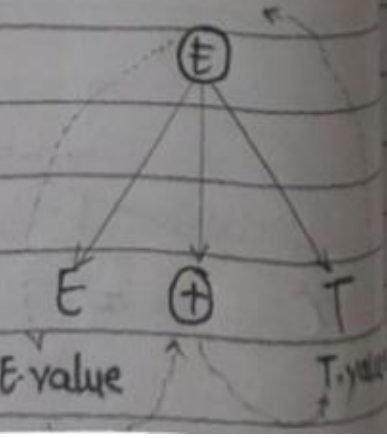
- S-Attributed
- L-Attributed

S-Attributed:

An SDT uses Synthesized attributes are referred as S-attributed definition. Synthesized attributes can be evaluated by the bottom-up parser as the input is being parsed. The parser keeps the values of synthesized attributes associated with a grammar on its stacks.

At the time of reduction the values of new synthesized are computed from the attributes appearing on the stack for the grammar symbols on the right side of reducing production.

The translation can also be specified by S attributed definitions then the semantics rules can be evaluated by LR parser E.value = E.value + T.value itself during the parsing



As depicted above, attributes in S-attributed SDTs are evaluated in bottom-up parsing as the values of the parent nodes depend upon

Problem: Consider the SDTS

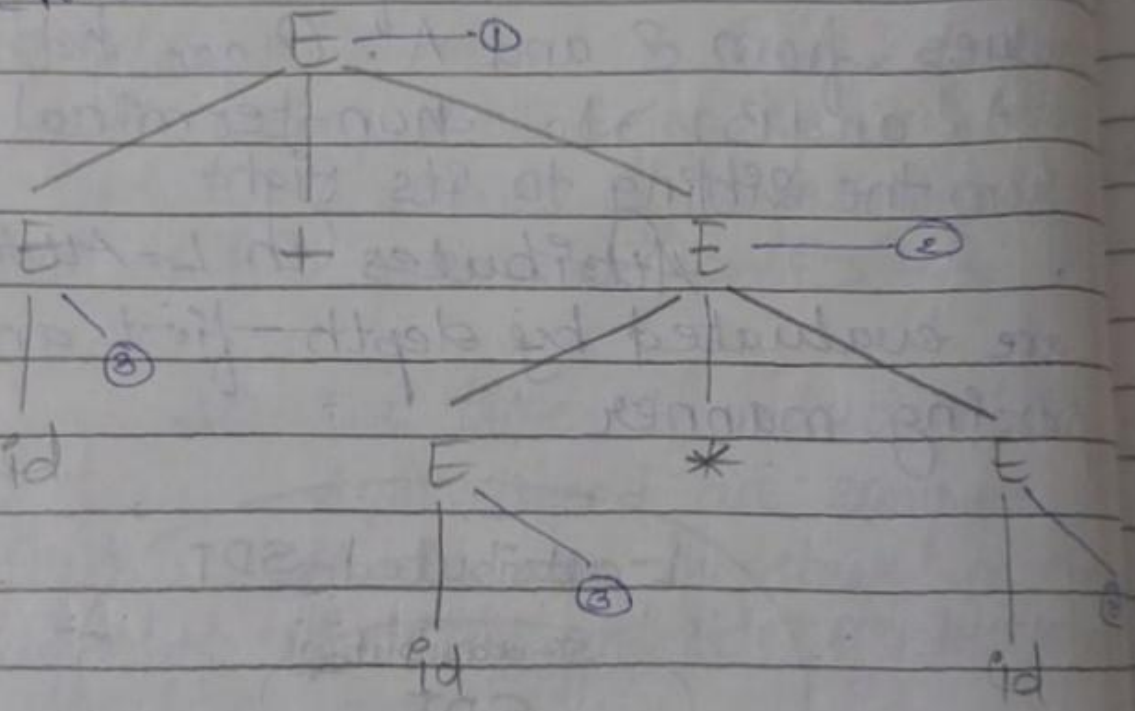
$E \rightarrow E+E$ { print '+' }
$E \rightarrow E*E$ { print '*' }
$E \rightarrow id$ { print id.name }

convert suffix "id* id + id" into postfix

Sol:

$E \rightarrow E+E$ { print '+' } ——— ①
$E \rightarrow E*E$ { print '*' } ——— ②
$E \rightarrow id$ { print id.name } ——— ③

Parse Tree



postfix Expression: id id id * +

Date: / /201

Q. Consider the SDTS

$E \rightarrow E + T$ { print '+' }

$E \rightarrow T$

$T \rightarrow T * f$ { print '*' }

$T \rightarrow f$

$f \rightarrow num$ { print numeral }

Convert "2+3*4" into postfix

$E \rightarrow E + T$ { print '+' } ——— ①

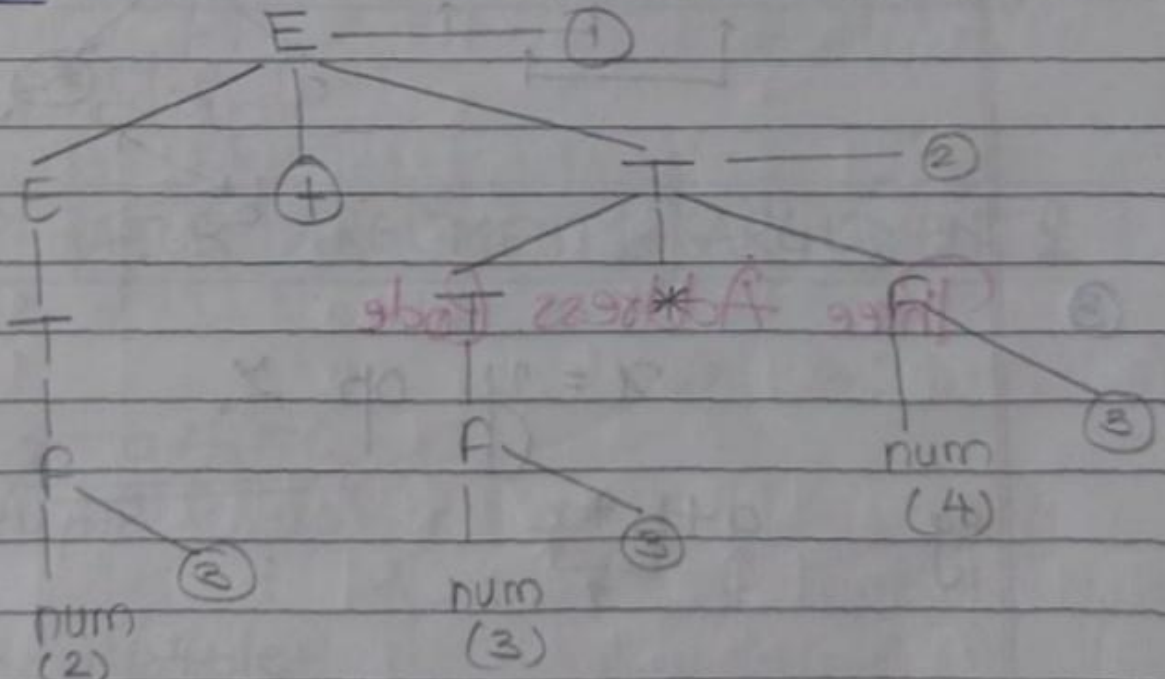
$E \rightarrow T$ { }

$T \rightarrow T * f$ { print '*' } ——— ②

$T \rightarrow f$ { }

$f \rightarrow num$ { print numeral } ——— ③

PARSE TREE



postfix Expression : 234*+

Intermediate Code Representation

There are following representation used

① **Postfix Notation:**

In this type of notation operator is placed after the operands is known as "POSTFIX NOTATION"

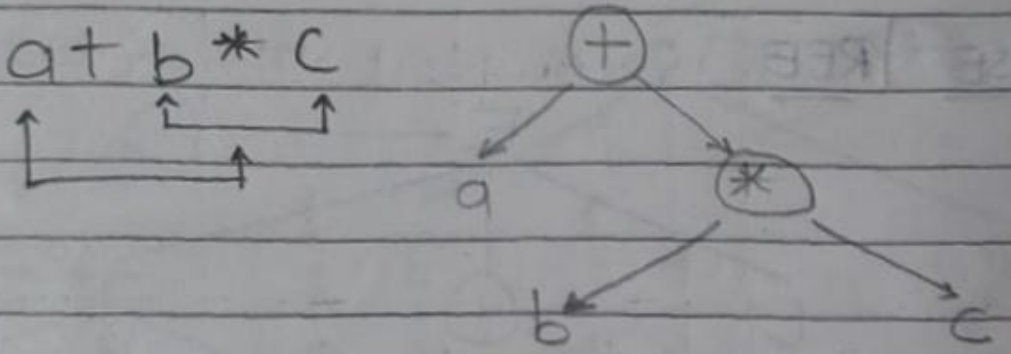
eg.

a+b*c

postfix abc*+.

② **Syntax Tree**

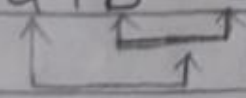
for example "a+b*c" will be



Three Address Code

$x = y \text{ op } z$

eg a+b*c



$t_1 = b * c$

$t_2 = a + t_1$

Representation of Three Address Code OR Data Structure to represent Three Address Code.

There are three methods used to represent Three address code (TAC)

1. Quadruple.
2. Triple.
3. Indirect Triple.

1) Quadruple

OPERATOR	ARGUMENT1	ARGUMENT2	RESULT

2) Triple

OPERATOR	ARGUMENT1	ARGUMENT2

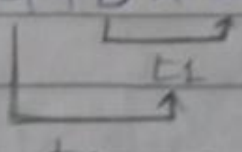
3) Indirect Triple

LIST	OPERATOR	ARGUMENT1	ARGUMENT2

Represent $x = a+b*c$

1. Quadruple
2. Triple
3. Indirect Triple

$x = a+b*c$



TAC →

$t_1 = b * c$

$t_2 = t_1 * a$

$x = t_2$

Intermediate Code Generation using SDT

Important terms related to SDT

1) Make list: Create new node

2) Backpatch (P, I): pointer P contain address of travelling from P to I

3) Merge (P1, P2): Concatenates P1 and P2

4) Quad: Statement number

5) Gen(): Generate output (in TAC).

SDTS for Assignment or MIX MODE Assignment statement.

Grammar:

$A \rightarrow id := E$

$E \rightarrow E+E \mid E*E \mid -E \mid (E) \mid id$

SDTS:

$A \rightarrow id := E$

{

A.code := E.code || gen(id.place := E.place)

}

Page No.
Date : / /201

$E \rightarrow E_1 + E_2$

{

T := New Temp();

E.place := T

E.code := E_1.code || E_2.code || gen (E.place := E_1.place '+' E_2.place)

}

$E \rightarrow E_1 * E_2$

{

T := New Temp();

E.place := T

E.code := E_1.code || E_2.code || gen (E.place := E_1.place '*' E_2.place)

}

$E \rightarrow - E_1$

{

T := New Temp();

E.place := T

E.code := E_1.code || gen (E.place := '-' E_1.place)

}

$E \rightarrow (E_1)$

{

E.place := E_1.place

E.code := E_1.place

}

$E \rightarrow id$

{

E.place := id.place

E.code := null

Problem: Write Three address code and generate parse tree for the given statement
$x := a + b * c$ (by SDTS)

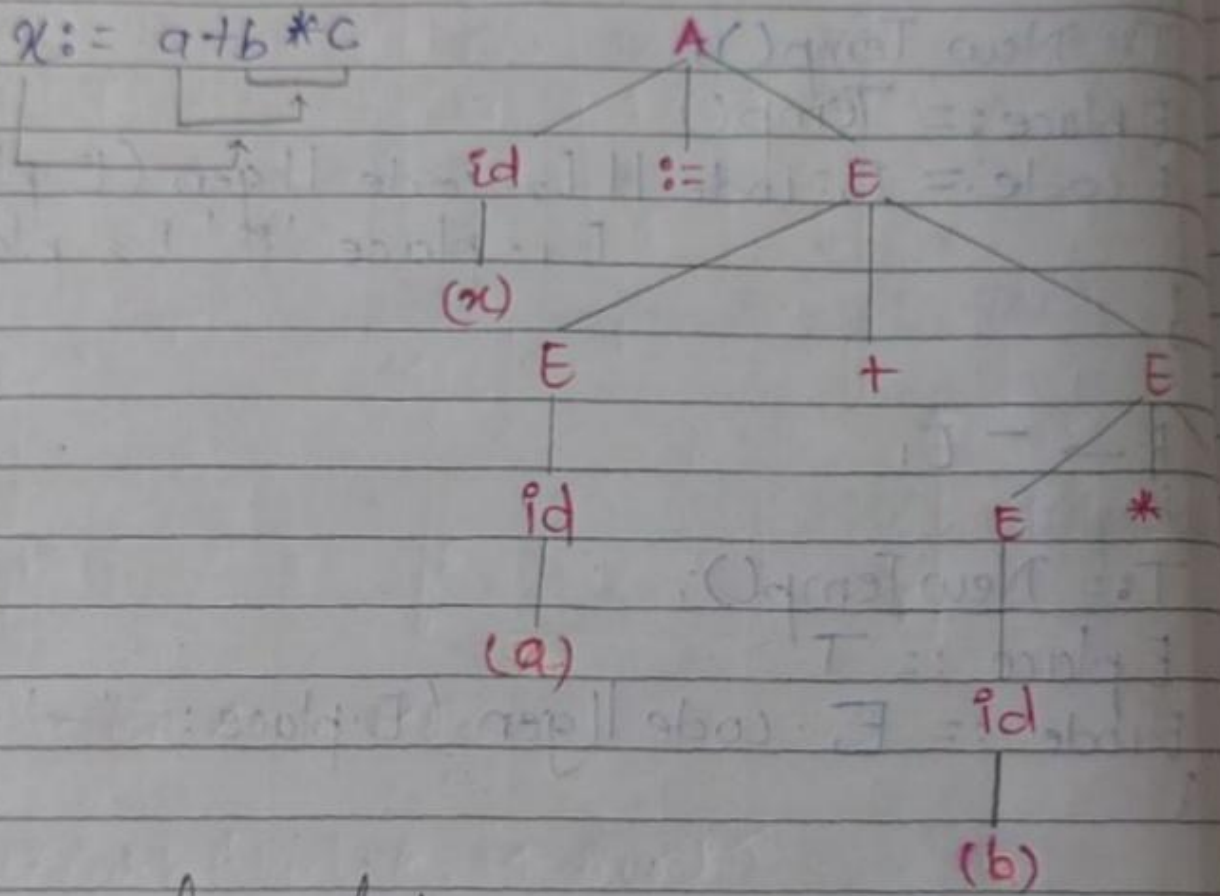
four step solution

1. Generate Grammar

$A \rightarrow id := E$

$E \rightarrow E+E \mid E*E \mid -E \mid (E) \mid id$

2) Parse Tree



* SDTS for assignment of MIX MODE Assignment statement

Grammer:

$A \rightarrow id := E$

$E \rightarrow E+E \mid E*E \mid -E \mid (E) \mid id$

SDTS:

$A \rightarrow id := E$

{ A.code := E.code || gen(id.place := E.place)

Page No.
Date : / /201

$E \rightarrow E_1 + E_2$
{
T := New Temp();
E.place := T
E.code := E_1.code || E_2.code || gen(E.place := E_1.place '+' E_2.place)
}

$E \rightarrow E_1 * E_2$
{
T := New Temp();
E.place := T
E.code := E_1.code || E_2.code || gen(E.place := E_1.place '*' E_2.place)
}

$E \rightarrow -E_1$
{
T := New Temp();
E.place := T
E.code := E.code || gen(E.place := '-' E_1.place)
}

$E \rightarrow (E_1)$
{
E.place := E.place
E.code := E_1
}

$E \rightarrow id$
{
E.place := id.place
E.code := null
}

4. $T_1 := b * c$
$T_2 := a + T_1$
$x := T_2$

2. SDTS for Boolean Expression / Short circuit code for logical expression

AND

$E \rightarrow E_1$ and ME_2 (Repeat)

{

Backpatch (E_1.true, M.quad)

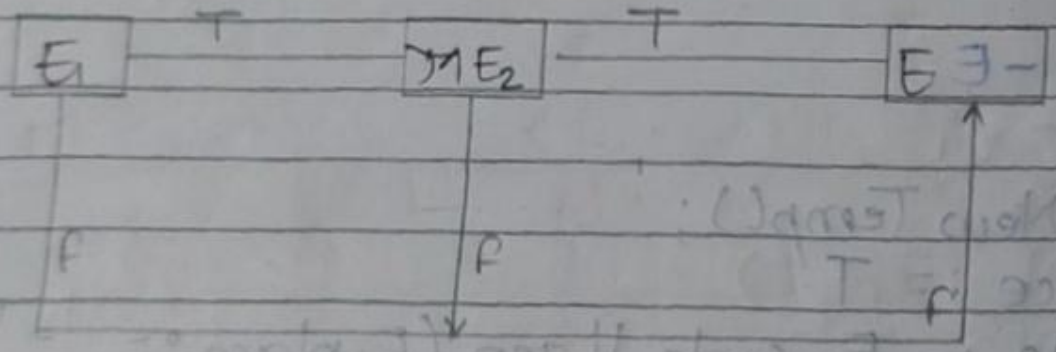
E.true := E_2.true

E.false := Merge (E_1.false, E_2.false)

1) AND 'TT'

E_1	E_2	E
0	0	0
0	1	0
1	0	0
1	1	1

1) AND
2) OR
3) NOT



AND $E \rightarrow E_1$ AND ME_2

{

Backpatch (E_1.true, M.quad)

E.true := E_2.true

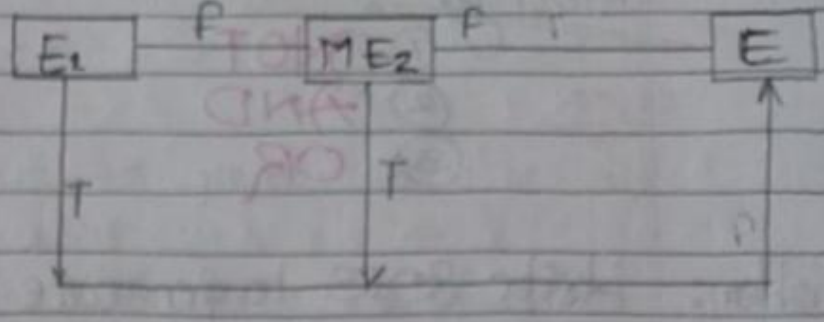
E.false := Merge (E_1.false, E_2.false)

}

Date : / /201

② OR

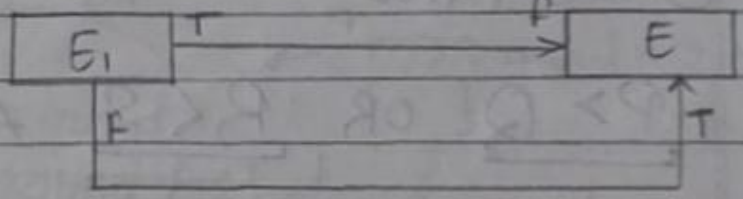
E_1	E_2	E
0	0	0
0	1	1
1	0	1
1	1	1



E → E1 OR ME2
{
Backpatch (E1.false, M.quad)
E.false := E2.false
E.true := Merge (E1.true, E2.true)
}

③ NOT

E_1	E
0	1
1	0



E → not E1
{
E.true := E1.false
E.false := E1.true
}

M → ∈
{
M.quad := Nextquad
}

E → id relop id
{
E.true := Next quad
E.false := Next quad +1
gencode (if id relop id goto_)
gencode (goto_)
}

→ Priority.

① NOT
② AND
③ OR

Problem: Write SDTS to generate 3 addree code (TAC) expression

$P > Q$ OR $R < S$ AND $T < U$

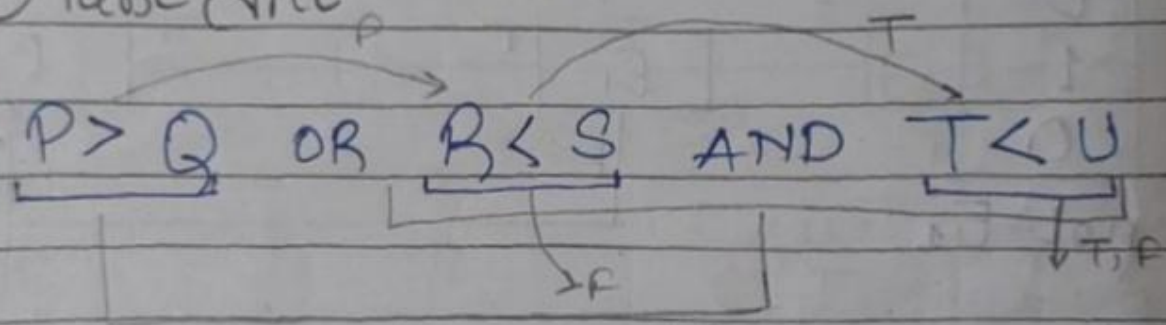
Sol: ① Generate Grammar

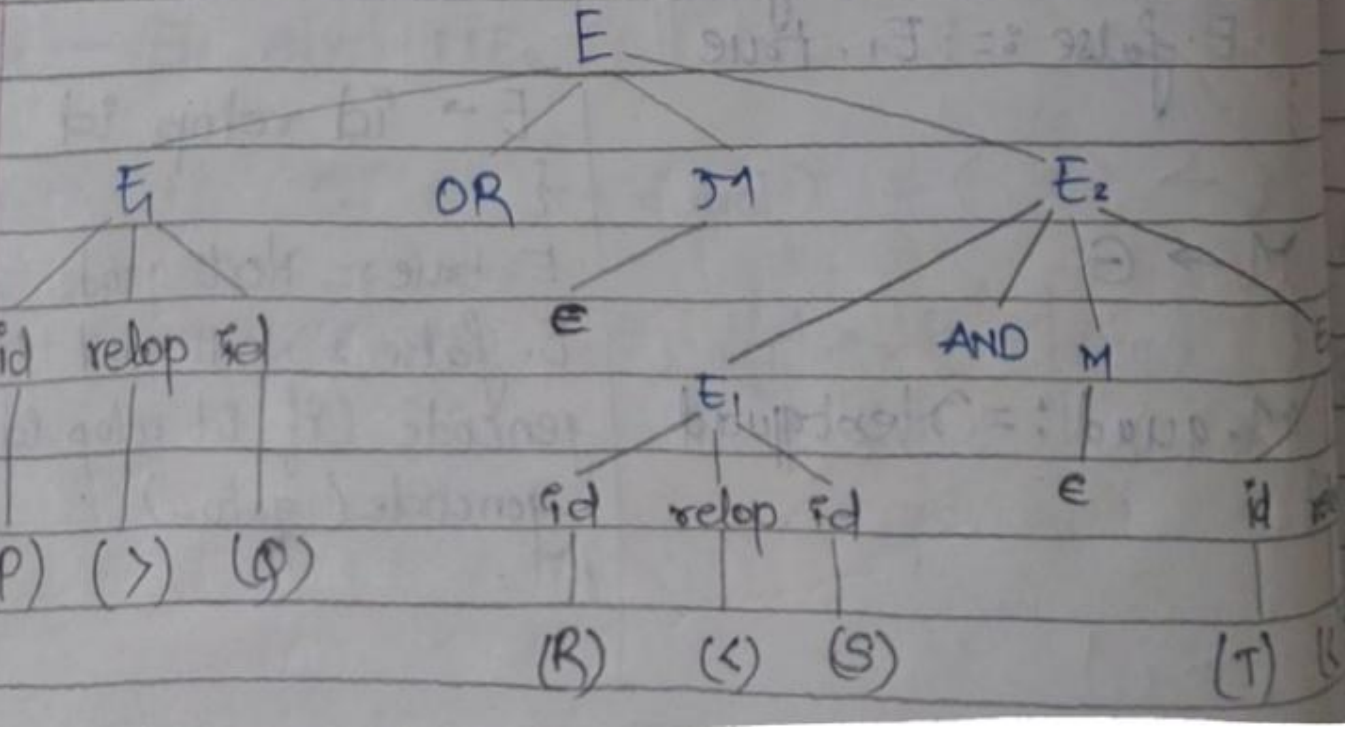
$E \rightarrow E_1$ AND $M E_2$

$E \rightarrow E_1$ OR $M E_2$

$E \rightarrow$ id relop id

② Parse Tree





Date: / /201

3 SDTS.

$E \rightarrow E_1$ AND $M E_2$

{

Backpatch (E_1.true, M.quad)

E.true := E_2.true

E.false := Merge (E_1.false, E_2.false)

AND

$E \rightarrow E_1$ OR $M E_2$

{

Backpatch (E_1.false, M.quad)

E.false := E_2.false

E.true := Merge (E_2.true, E_2.true)

}

OR

$M \rightarrow \epsilon$

{

M.quad := Next quad

}

$E \rightarrow$ id relop id

{

E.true := Next quad

E.false := Next quad + 1

gencode (if Id relop Id goto)

gencode (goto _)

4 T.A.C.

100 if P > Q goto 106

101 goto 102

102 if R < S goto 104

103 goto 106

104 if T < U goto 106

105 goto 106

106 Next

Problem: Write SDTS to generate TAC for
(P<Q AND R<S) OR NOT (T<U AND

Soln: ① Generate Grammar

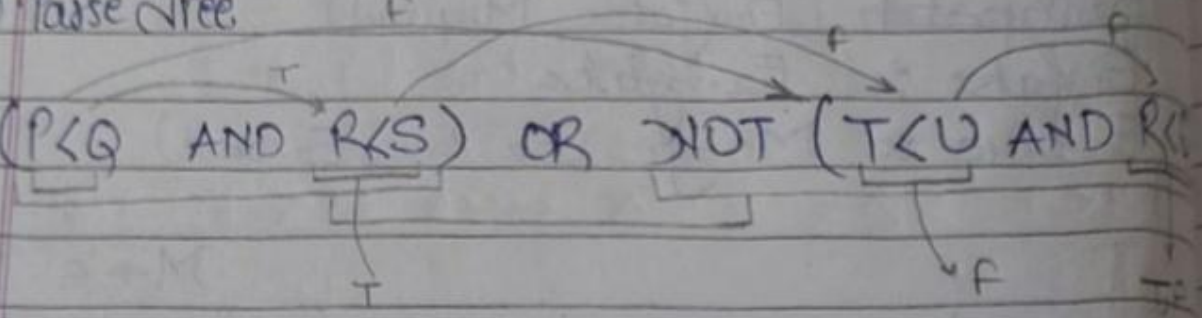
$E \rightarrow E_1$ AND M E_2

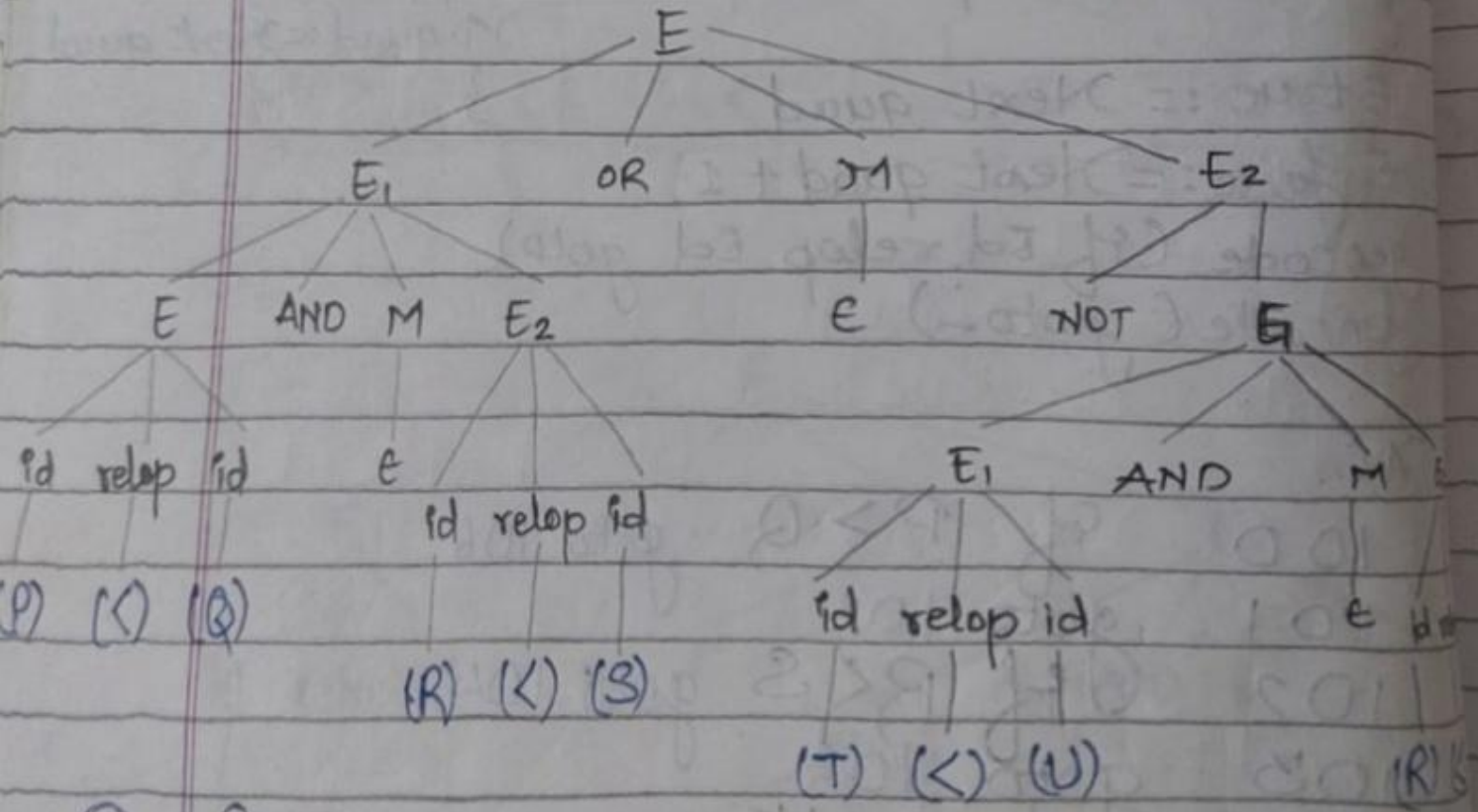
$E \rightarrow E_1$ OR M E_2

$E \rightarrow$ NOT E_1

$E \rightarrow$ id relop id

② Parse Tree





③ SDTS

$E \rightarrow E_1$ AND M E_2

{

Backpatch (E_1.true, M.quad)

E.true := E_2.true

OR

$E \rightarrow E_1$ or M E_2

{

Backpatch(E_1.false, M.quad)

E.false := E_2.false

E.true := Merge (E_1.true, E_2.true)

}

NOT

$E \rightarrow$ NOT E_1

{

E.true := E_1.false

E.false := E_1.true

}

$M \rightarrow \epsilon$

{

M.quad := Next quad

}

$E \rightarrow$ id relop id

{

E.true := Next quad

E.false := Next quad + 1

gencode (if id relop id goto _)

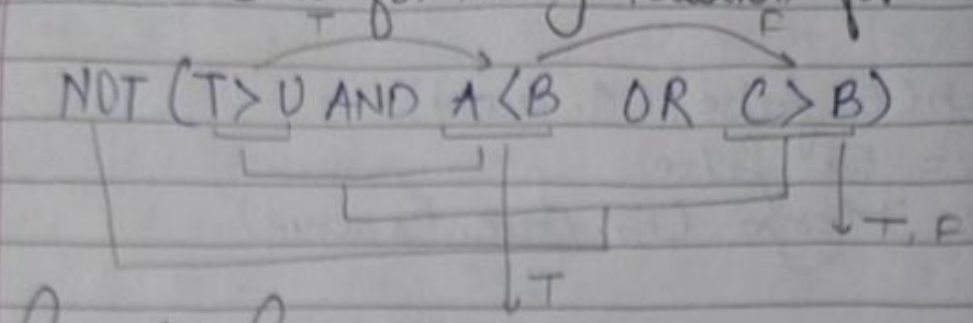
gencode (goto _)

4 TAC

100 if P<Q goto 102
101 goto 104
102 if R< S goto 108
103 goto 104
104 if T<V goto 106
105 goto 108
106 if R<Q goto 108
107 goto 108
108 Next

Problem: Write the SDTS for TAC generation for

NOT (T > U AND A < B OR C > B)



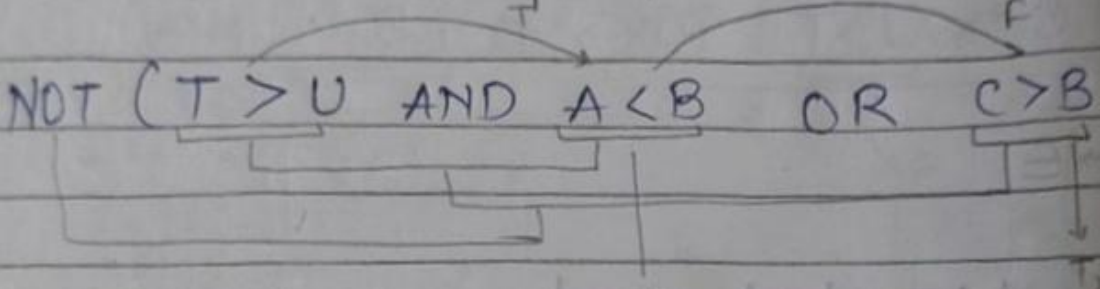
Soln: Generate Grammar

$E \rightarrow E_1$ AND ME_2

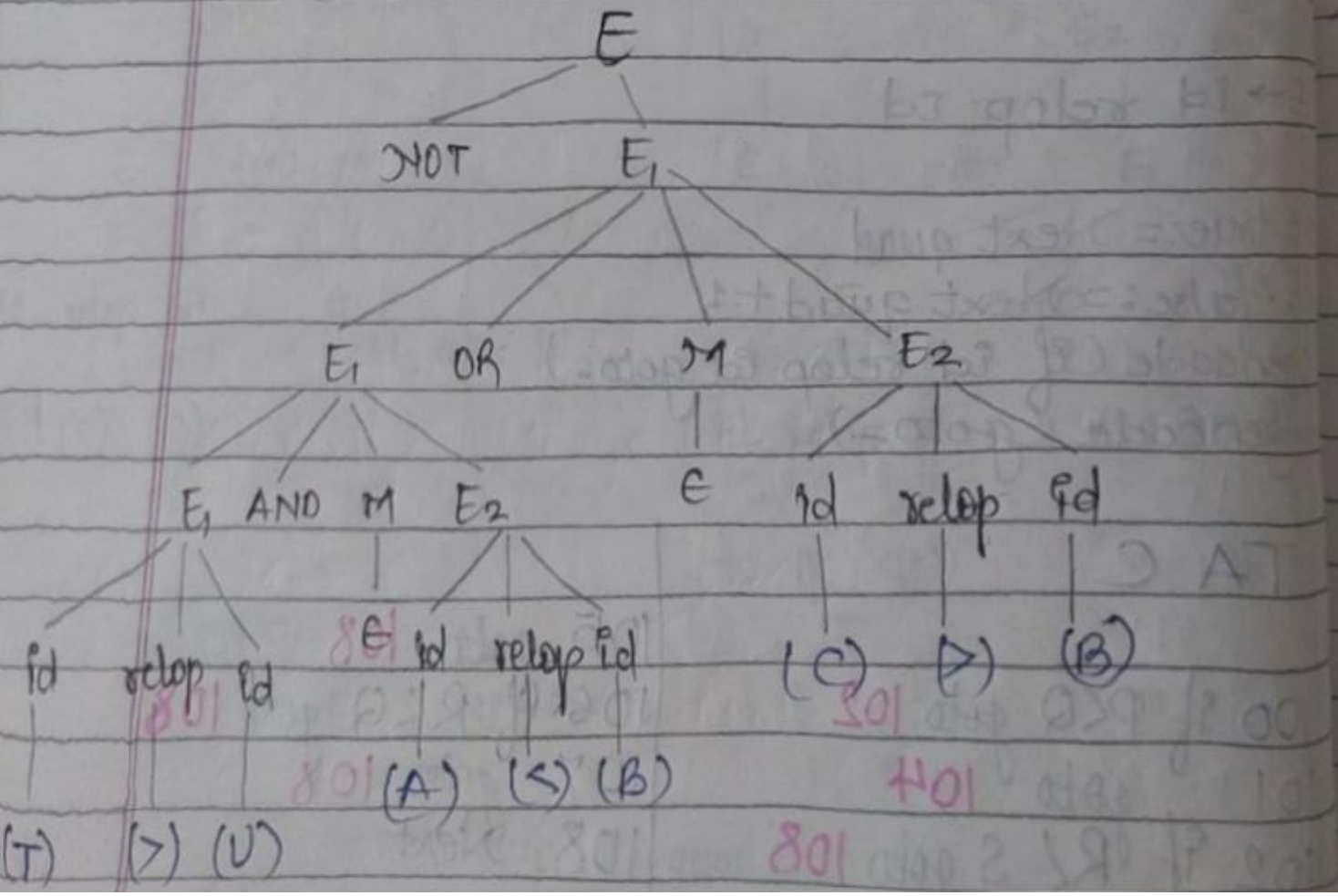
$E \rightarrow E_1$ OR ME_2

$E \rightarrow$ NOT E_1

$E \rightarrow$ id relop id



Parse tree



(8) SDTS

$E \rightarrow E_1$ AND ME_2

{

Backpatch (E_1.true, M.quad)

AND E.true := E_2.true

E.false := Merge (E_1.false, E_2.false)

}

$E \rightarrow E_1$ OR ME_2

{

Backpatch (E_1.false, M.quad)

OR E.false := E_2.false

E.true := Merge (E_1.true, E_2.true)

}

$E \rightarrow$ NOT E_1

NOT {

E.true := E_1.false

E.false := E_1.true

}

$M \rightarrow \epsilon$

{

M.quad := Next quad

}

$E \rightarrow$ id relop id

{

E.true := Next quad

E.false := Next quad + 1

gencode (if id relop id goto _)

gencode (goto _)

}

4. TAC

100 if T > U goto 102

101 goto 104

102 if A < B goto _

103 goto 104

104 if C > D goto _

105 goto

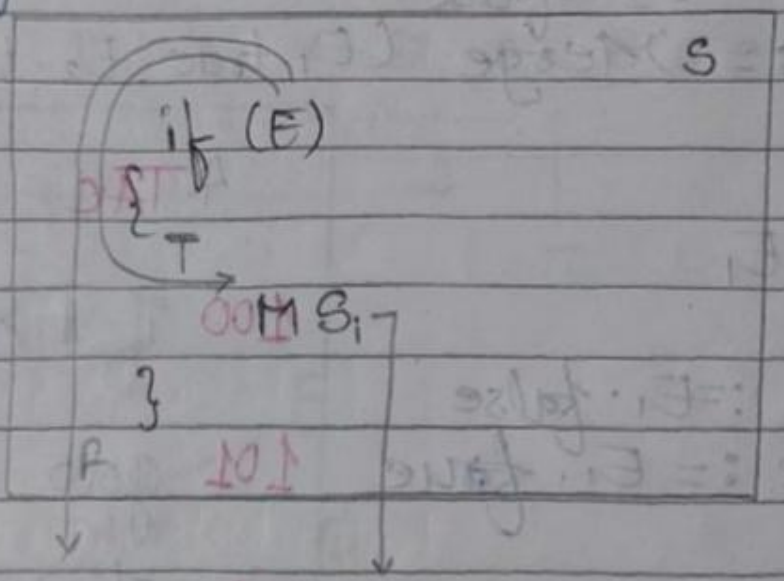
SDTS for Control Structure

Imp topic

1) if then
2) if then else
3) while do → entry control
4) do while → exit control
5) Repeat until
6) for

1. if then

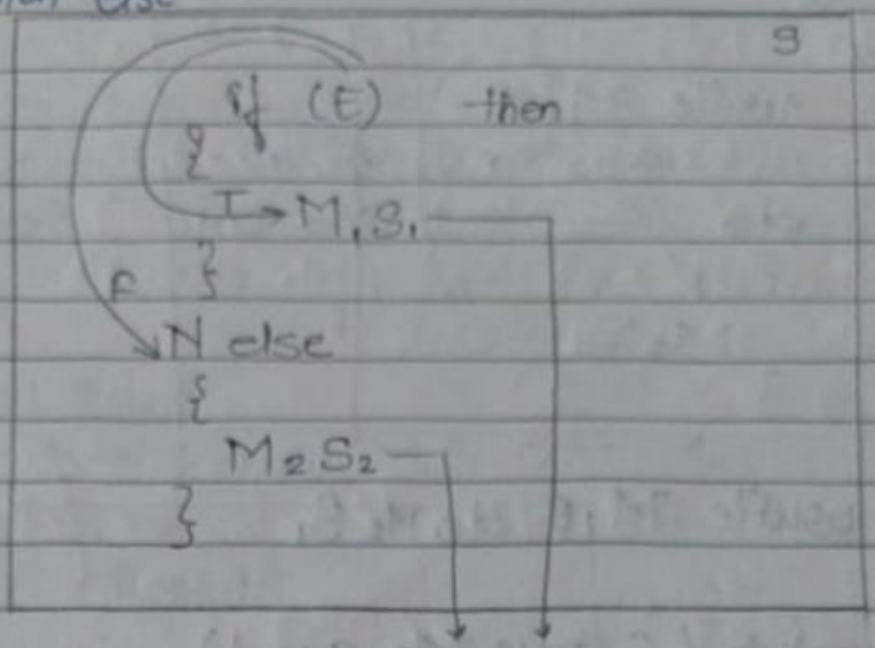
logic



S → if E then M S1
{
Backpatch (E.true, M.quad)
S.Next := Merge (E.false, S1.Next)
}

M → ∈
{
M.quad := Next quad
}

2. if then else

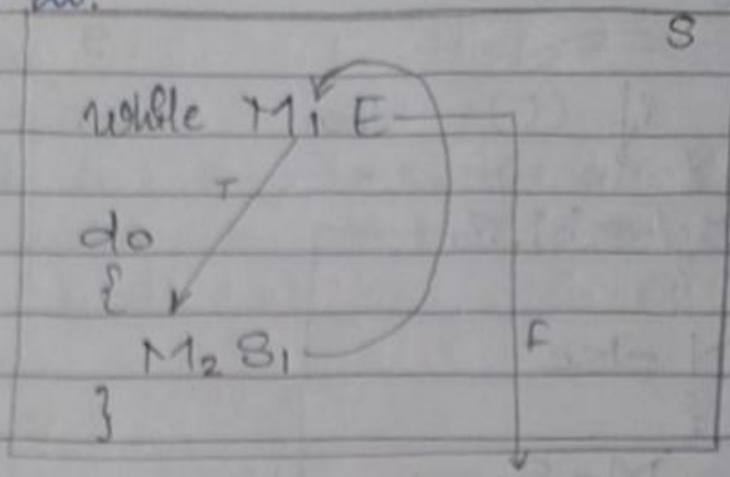


S → if E then M_1 S_1 N else M_2 S_2
{
Backpatch (E.true, M_1.quad)
Backpatch (E.false, M_2.quad)
S.Next = Merge (S_1.Next, S_2.Next, N.Next)
}

M → ϵ
{
M.quad := Next quad
}

N → ϵ
{
N.quad := Nextquad
gen (goto _)
}

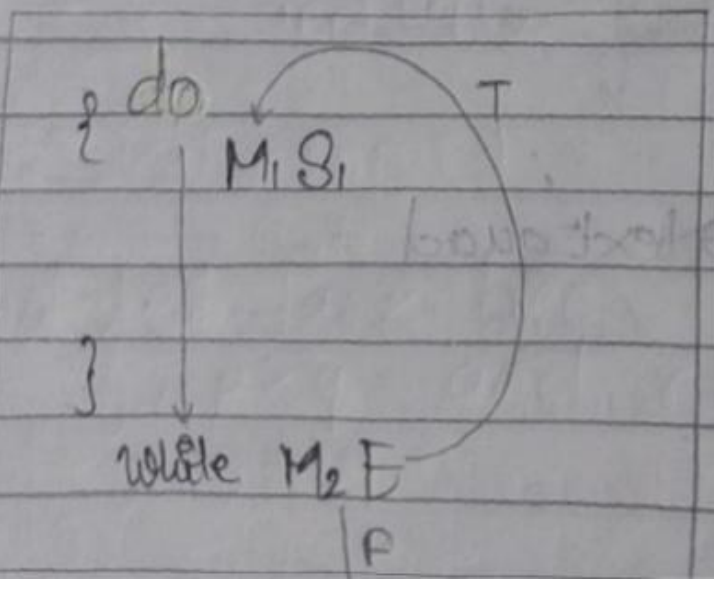
③ while do.



S → while M1 E do M2 S1
{
Backpatch (E.true, M2.quad)
Backpatch (S1.Next, M1.quad)
S.Next := E.false
gen (goto)
}

M → E.
{
M.quad := Next quad
}

④ Do while



Page No.
Date : / /201

$S \rightarrow$ do $M_1 S_1$ while $M_2 E$

{

Backpatch (E.true, M_1.quad)

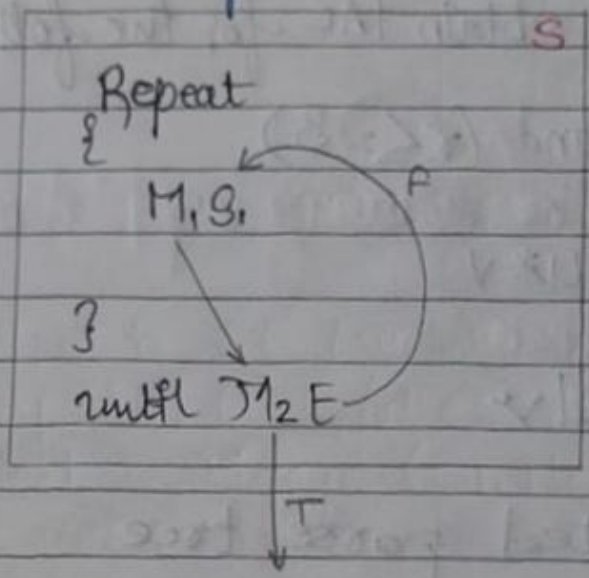
Backpatch (S_1.Next, M_2.quad)

S.Next := E.false

}

⑤ Repeat Until

$S \rightarrow$ Repeat $M_1 S_1$ Until $M_2 E$



$S \rightarrow$ Repeat $M_1 S_1$ until $M_2 E$

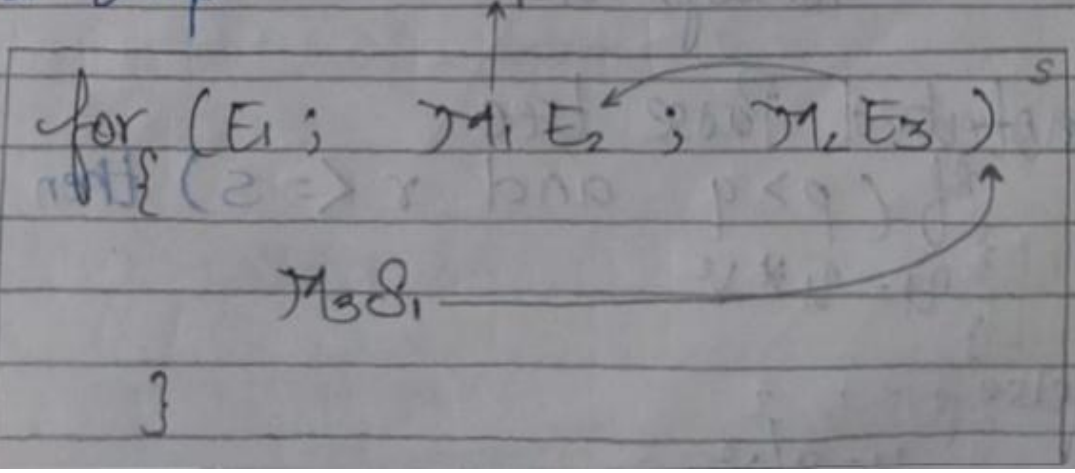
{

Backpatch (E.false, M_1.quad)

Backpatch (S_1.Next, M_2.quad)

S.Next := E.true

⑥ For loop



S → for (E1 ; M1 E2 ; M2 E2) M3 S1
{
Backpatch (E2.true, M3.quad)
Backpatch (S1.Next, M2.quad)
Backpatch (E2.Next, M1.quad)
S.Next := E2.false
gen (goto -

Problem Write SDTS and obtain IAC for the following

if (P>q and r<=s)
then
u = u*v
else
u = u/v

Draw Annotated parse tree

Soln: Generate Grammar

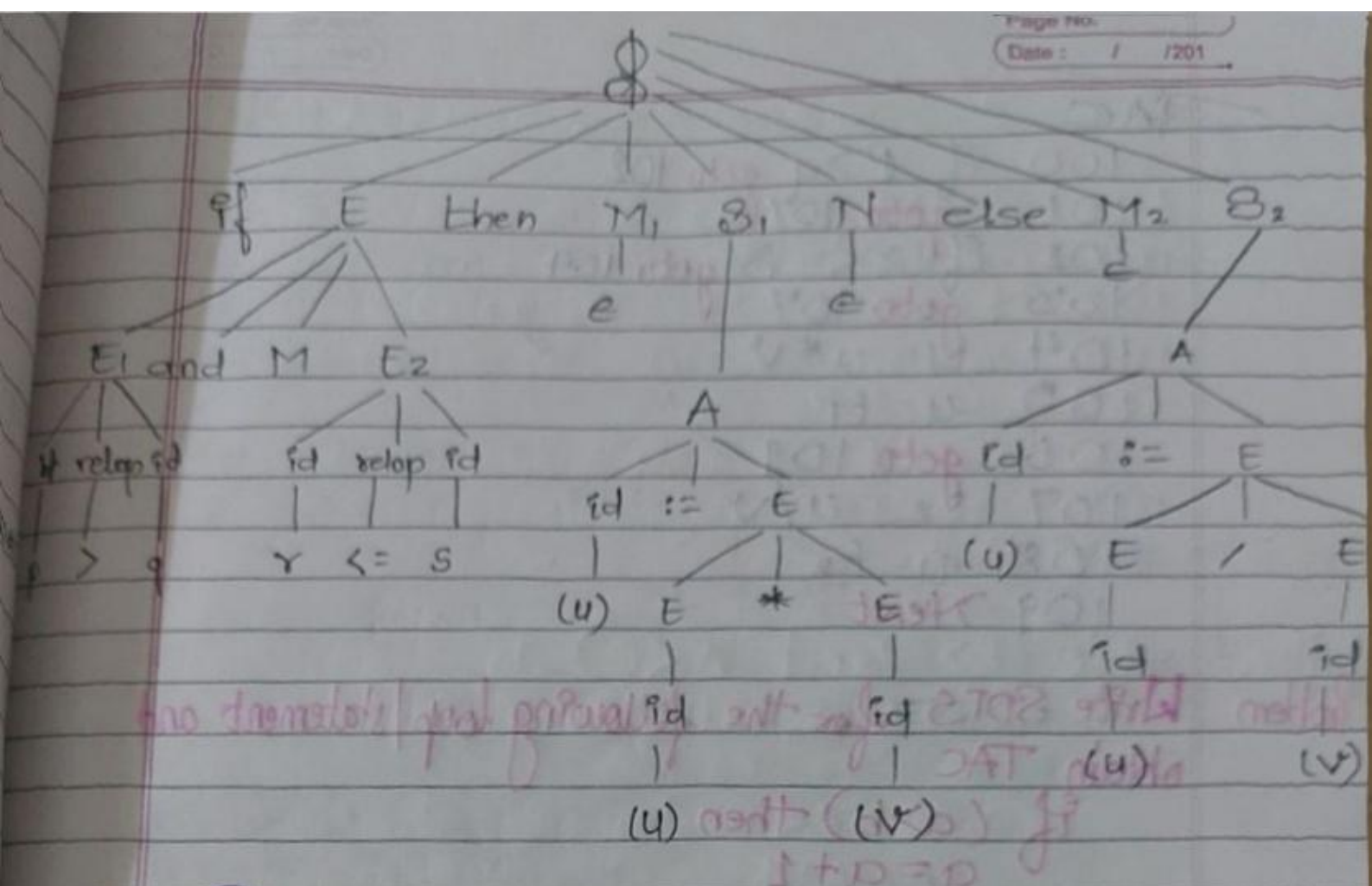
S → if E then M1 S1 N else M2 S2

E → E1 and M E2

E → id relop id

Annotated Parse tree

if (p>q and r<=s) then
{
u = u*v
}
else



(8) SDTS

$S \rightarrow$ if E then $M_1 S_1$ else $M_2 S_2$

{

Backpatch (E.true , M_1.quad)

Backpatch (E.false, M_2.quad)

S.Next := Merge (S_1.Next, S_2.Next, N.Next)

}

AND: $E \rightarrow E_1$ and M E_2

{

Backpatch (E_1.true , M.quad)

E.true := E_2.true

E.false := Merge (E_1.false , E_2.false)

}

$E \rightarrow$ id relop id

{

E.true := Next quad

E.false := Next quad+1

gencode(if id relop id goto -)

gencode(goto -)

}

TAC

```
100 if p>q goto 102
101 goto 107
102 if r<=s goto 104
103 goto 107
104 t1 = u*v
105 u = t1
106 goto 109
107 t2 = u/v
108 u = t2
109 Next
```

Problem Write SDTS for the following loop/statement & obtain TAC

```
if (a>b) then
    a=a+1
else
    b=b-1
```

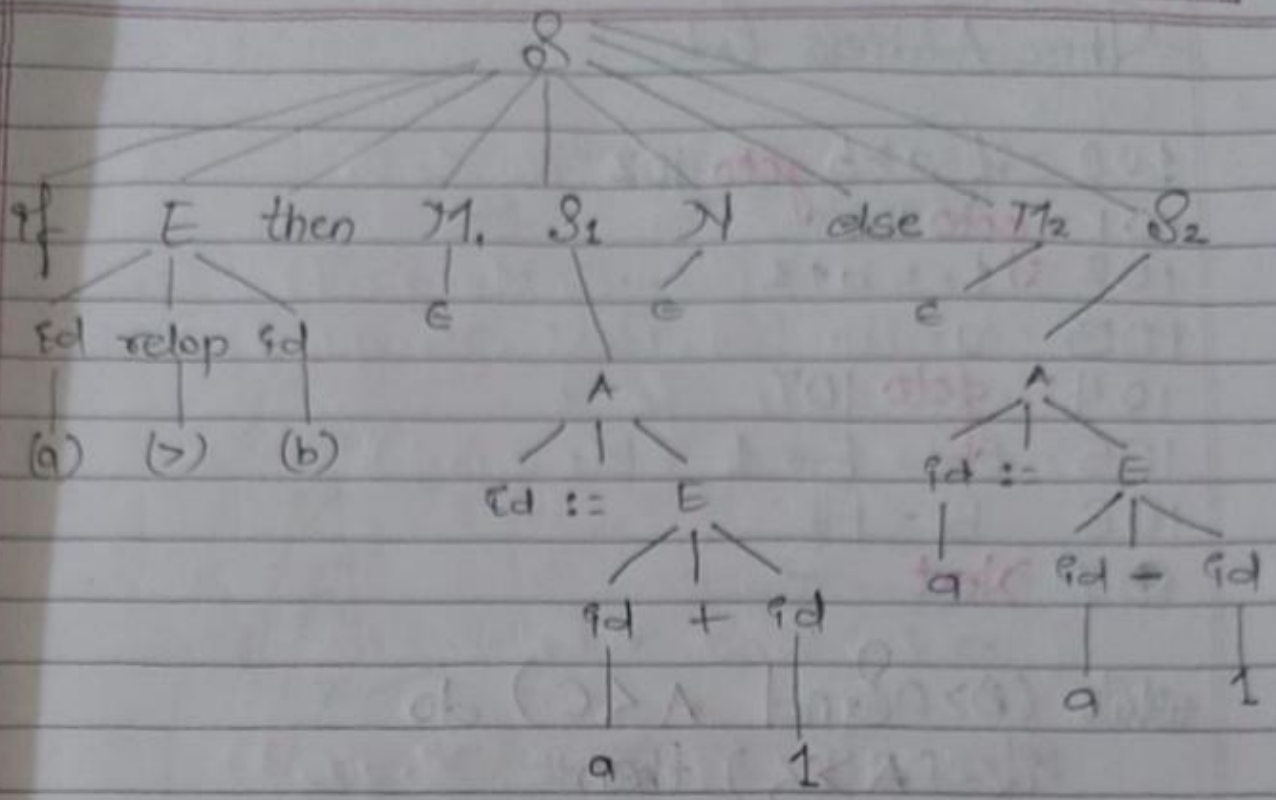
Ans. Generate Grammar

$S \rightarrow$ if E then $M_1 S_1$ N else $M_2 S_2$

$E \rightarrow$ id relop id

Annotation Parse Tree

```
if (a>b) then
{
    a+b
}
else
{
    b=b-1
}
```



SDTS

$S \rightarrow$ if E then $M_1 S_1 N$ else $M_2 S_2$

{
Backpatch (E.true, M.quad)
Backpatch (E.false, M_2.quad)
S.Next = Merge (S1.Next, N.Next)
}

$M_1 \rightarrow \epsilon$

{
M.quad := Next quad
}

$N \rightarrow \epsilon$

{
N.quad := Next quad
gen (goto _)
}

$E \rightarrow$ id relop id

{
E.true := Next quad
E.false := Next quad + 1
gencode (if id relop id goto _)
gencode (goto _)

Three Address Code:

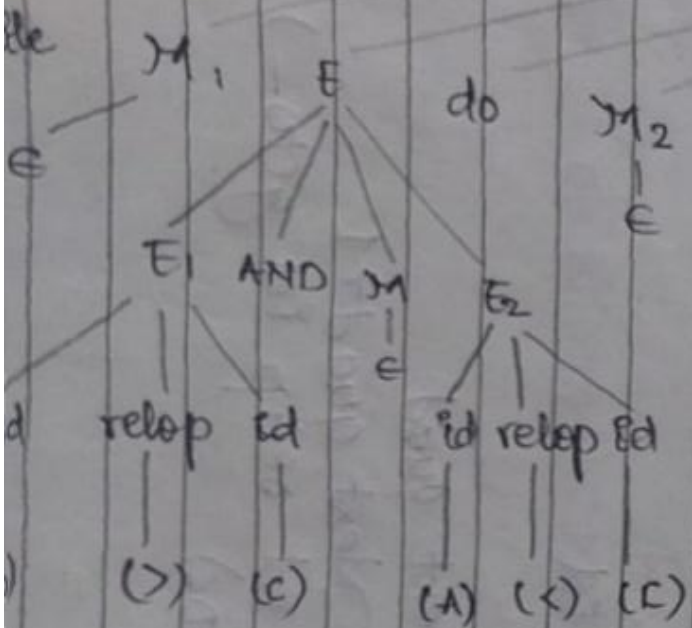
```
100 if a>b goto 102
101 goto 105
102 t1 = a+1
103 a = t1
104 goto 107
105 t2 = b-1
106 b = t2
107 Next
```

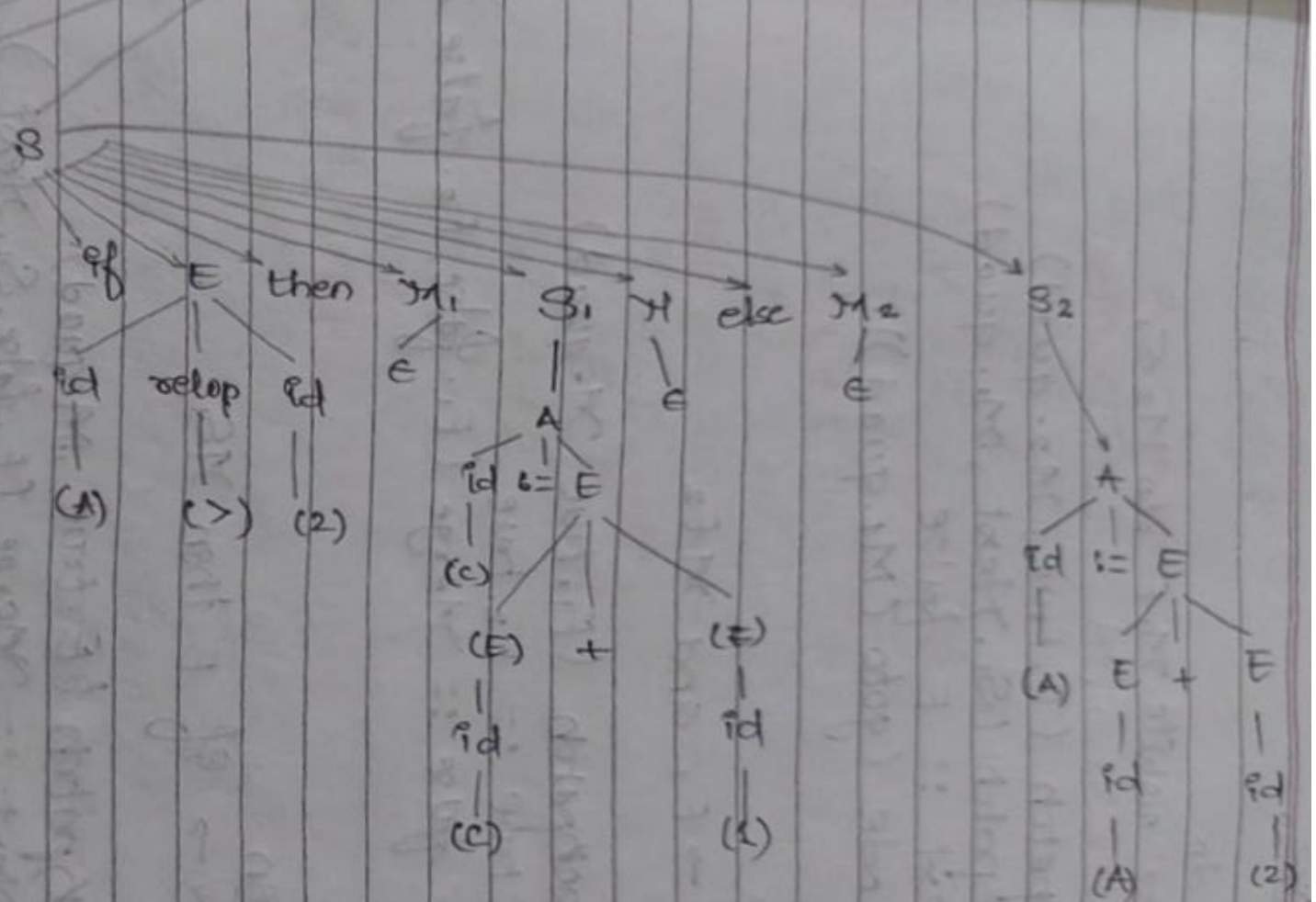
Problem: while (B>D and A<C) do
if (A>Z) then
else C = C+1
while (A<D) do
A = A+1

```
do while (B>D and A<C)
do
{
if (A>Z) then
  {
  C = C+1
  }
else
{
while (A<D)
do
{
A = A+1
}
```

Generate Grammar

S → while M1 E do M2 S
S → if E then M1 S1 N else
E → E1 AND ME2
E → id relop id


le
∈
M1
E
do
M2
∈
E1
AND
M
∈
E2
d
relop
id
(>)
(c)
id
relop
id
(A)
(<)
(C)


S
if
E
then
M1
∈
S1
N
∈
else
M2
∈
S2
id
relop
id
(A)
(>)
(2)
A
id := E
(c)
(E) +
id
(C)
(E)
id
(A)
A
id := E
(A)
E + E
id
(A)
id
(2)

SDTS.

* While do

S → while M1 E do M2 S1
{
Backpatch (E.true, M2.quad)
Backpatch (S1.Next, M1.quad)
S.Next := E.false
gencode (goto (M1.quad))

* AND

E → E1 and M E2
{
Backpatch (E1.true, M.quad)
E.true := E2.true
E.false := Merge (E1.false, E2.fal
}

* If then

S → if E then M S1
{
Backpatch (E.true, M.quad)
S.Next := Merge (E.false, S1.Next)
}

* E → id relop id

{
E.true := Next quad
E.false := Next quad + 1
gencode (if id relop id goto _)
gencode (goto _)
}

three

100
101
102
103
104
105
106
107

Date: / /201

Three address code:

```
100  if B>D goto 102
101  goto 115
102  if A<C goto 104
103  goto 115
104  if A>Z goto 106
105  goto 109
106  t1 = C+1
107  C = t1
108  goto ---------- (ELES)
109  if A<D goto
110  goto 115
111  t2 = A+2
112  A = t2
113  goto 109
114  goto 100
115  NEXT
```

⑧ while (a<10 and c>D) do
if (a<b) then
a=a+b
else
b=a+b

Ans:

```
while (a<10 and c>D)
{
  do
  {
    if (a<b) then
    {
      a=a+b
    }
    else
    {
      b=a+b
    }
  }
```

Generate Grammar

$S \rightarrow$ while M_1 E do M_2 S_1

$S \rightarrow$ if E then M_1 S_1 else M_2 S_2

$E \rightarrow E_1$ and M E_2

$E \rightarrow$ id relop id

```
100 if a<10 goto 102
101 goto 112
102 if c>D goto 104
103 goto 112
104 if a<b goto 106
105 goto 107
106 t1=a+b
107 a=t1
108 goto 111
109 t2=a+b
110 b=t2
111 goto 100
112 Next
```

Ex while (A>B or C>B) do
if (D>20 AND NOT (B<C)) then
A= A+B
else
D= D−1
X= Y+1

SDTS for Switch Case:

Q Write SDTS for switch statement.

Syntax

```
Switch (condition)
  begin
    {
      Case V1 : statement 1
      Case V2 : Statement 2
      Case V3 : statement 3
      .
      .
      .
      Case Vn : statement n
    }
END.
```

S → switch EN begin Caselist end
Caselist → Case V : S
Caselist → Caselist1 Case V : S
Caselist → Caselist1 default : S
E → Ed relop Ed
M → ϵ
N → ϵ

Q Translate the following switch to SDTS

```
switch (a+b)
{
  case 2 : { x = x+1 ; break }
  case 3 : { x = y+1 ; break }
  case 4 : { y = y+1 ; break }
```

Ans: Generate Grammar:-

S → switch E M begin Caselist end

Caselist → Case V : S

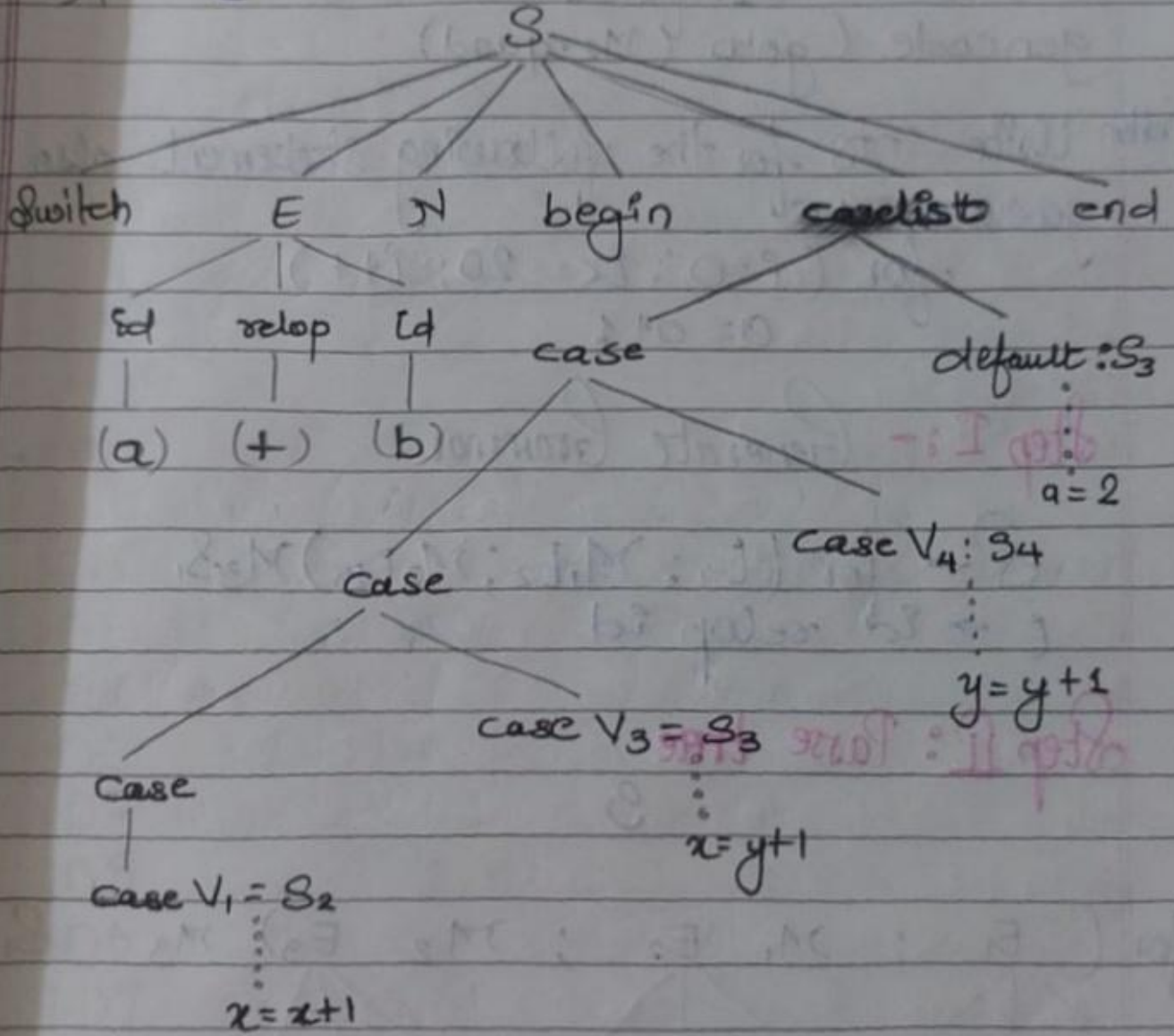
Caselist → Caselist$_1$ Case V : S

Caselist → Caselist$_2$ default : S

E → id relop id

M → ∈

N → ∈



100	$t_1 = a + b$	108	$t_4 = y + 1$		
101	goto 113	109	$y = t_4$	116	goto 111
102	$t_2 = x + 1$	110	goto 117	117	Next
103	$x = t_2$	111	$a = 2$		
104	goto 117	112	goto 117		
105	$t_3 = y - 1$	113	if $t_1 = 2$ goto 102		
106	$x = t_3$	114	if $t_2 = 3$ goto 105		
107	goto 117	115	if $t_1 = 4$ goto 108		

For:

$S \rightarrow$ for $(E_1 ; M_1 E_2 ; M_2 E_3) M_3 S$

{

Backpatch (E_3.true, M_3.quad)

Backpatch (S_1.Next, M_2.quad)

Backpatch (E_3.Next, M_1.quad)

S.Next := E_2.false

gencode (goto (M_2.quad)

Ques Write SDTS for the following statement, generate TAC

```
for (i=0; i<= 20; i++)
    a= a+1
```

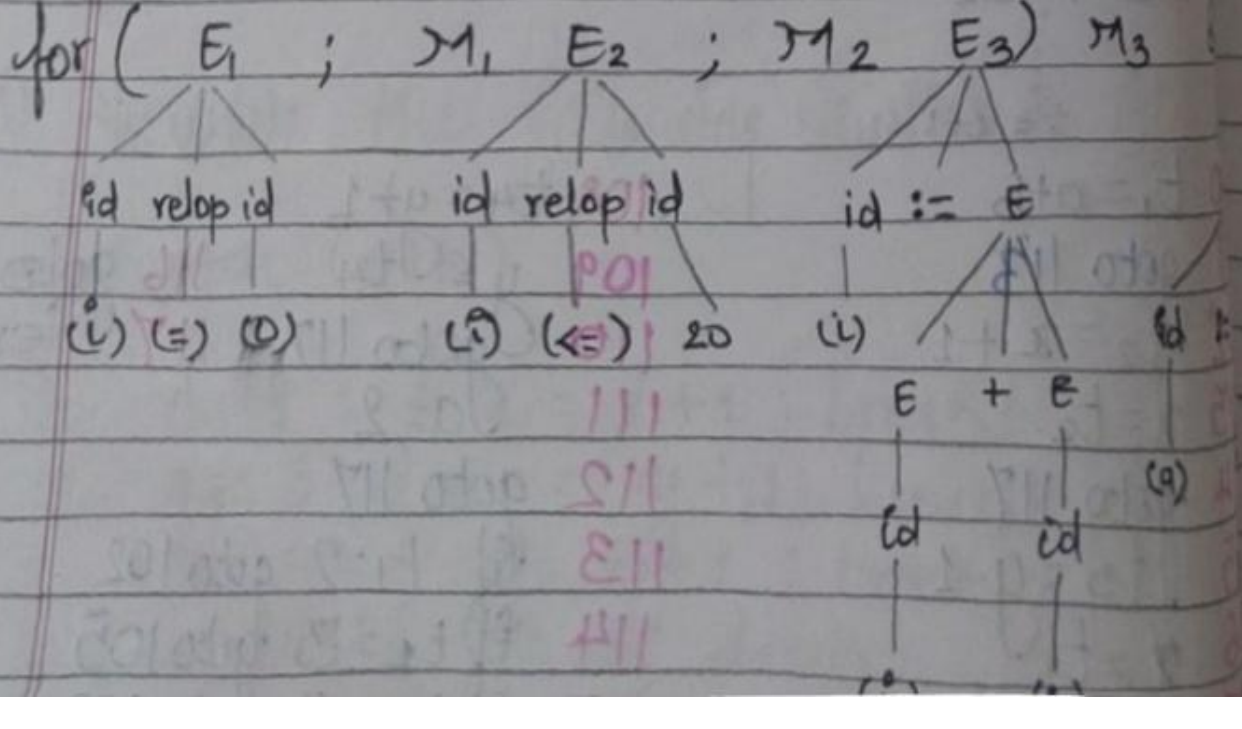
Sol

Step I:- Generate Grammar

$S \rightarrow$ for $(E_1 ; M_1 E_2 ; M_2 E_3) M_3 S_1$

$E \rightarrow$ id relop id

Step II: Parse tree



S → for (E_1; M_1 E_2; M_2 E_3) M_3 S.

{

Backpatch (E_3.true, M_3.quad)

Backpatch (S_1.Next, M_2.quad)

Backpatch (E_3.Next, M_1.quad)

S.Next := E_2.false

gencode (goto (M_2.quad))

}

E → id relop id

{

E.true = Next.quad

E.false = Next.quad

gen (if id relop id goto _)

gen (goto _)

}

Step 4: TAC.

```
100 i = 0
101 if i <= 20 goto 103
102 goto 108
103 t1 = a + 1
104 a = t1
105 t2 = i + 1
106 i = t2
107 goto 101
108. Next.
```

```
do
  if (D>20 and not (B<C)) then
  else
    {
      A = A+B
    }
  else
    {
      D = D-1
      X = Y+Z
    }
```

dig S → while M_1 E do M_2 S_1

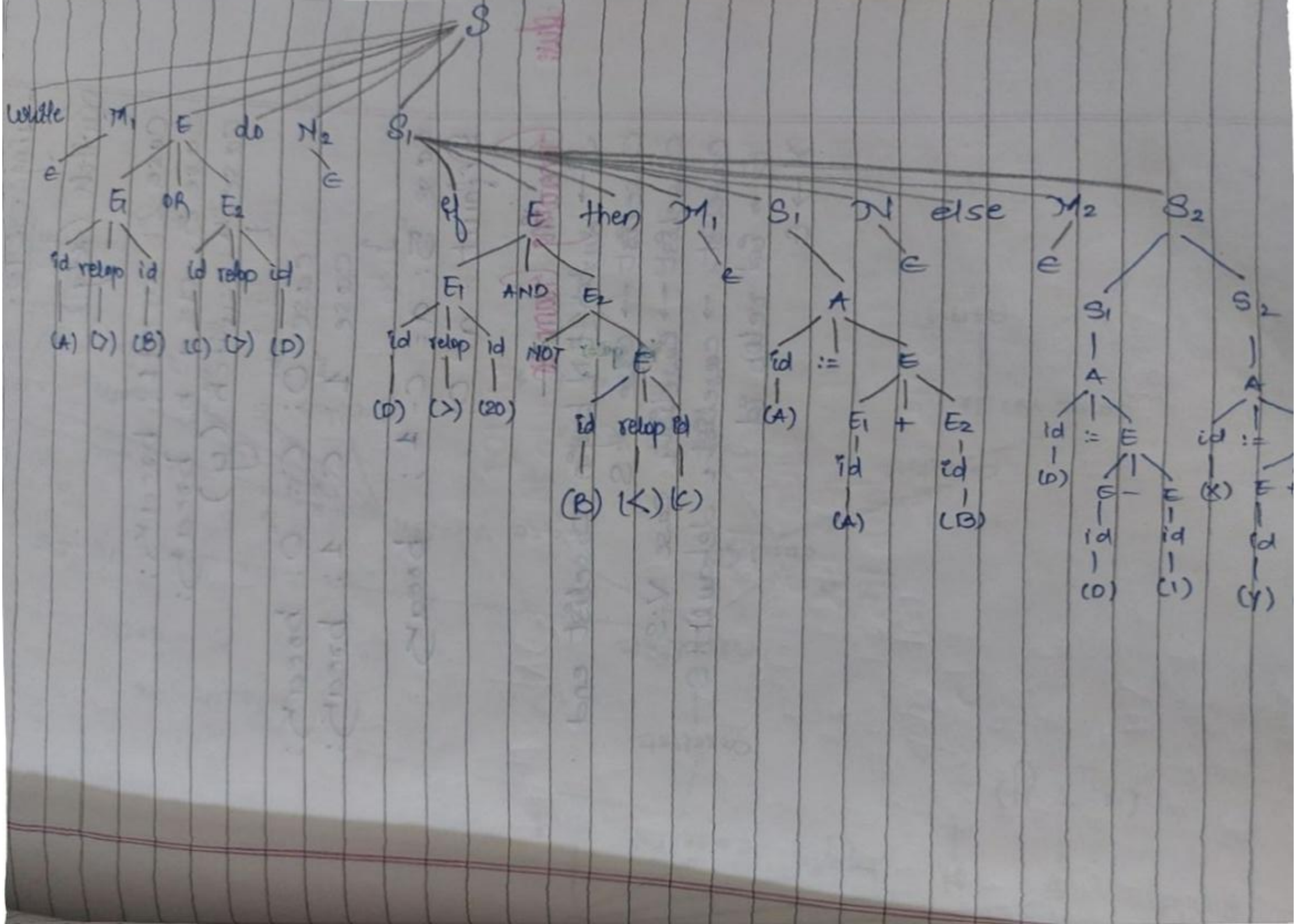
Pro S → if E then M_1 S_1 N else M_2 S_2

E → E_1 and E_2

E → not E_1

E → id relop id

```
100  if A>B goto 104
101  goto 102
102  if C>D goto 104
103  goto 116
104  if D>20 goto 106
105  goto 111
106  if B<D goto 111
107  goto 108
108  b = A+B
109  A = t1
110  goto 115
111  t2 = D-1
112  D = t2
113  t3 = Y+2
```


S
while M1 E do M2 S1
ε
E1 OR E2
id relop id id relop id
(A) (>) (B) (C) (>) (D)
ε
if E then M1 S1 N else M2 S2
E1 AND E2
id relop id NOT E
(D) (>) (20)
id relop id
(B) (<) (C)
ε
A
id := E
(A) E1 + E2
id id
(A) (B)
ε
S1 S2
A A
id := E id := E
(D) E − E (X) E +
id id id
(D) (1) (Y)

* Translate the following code Intermediate Code using SDTS.

```
switch (x+y)
{
case 2: a = a+1; break;
case 3: a = a-b; break;
case 4: switch (y)
        {
            case 0: c = 0; break;
            case 1: c = 1; break;
        }
case 5: a = c-1; break
Default: a = 0;
```

Sol: Generating Grammar

S → switch EN begin caselist end

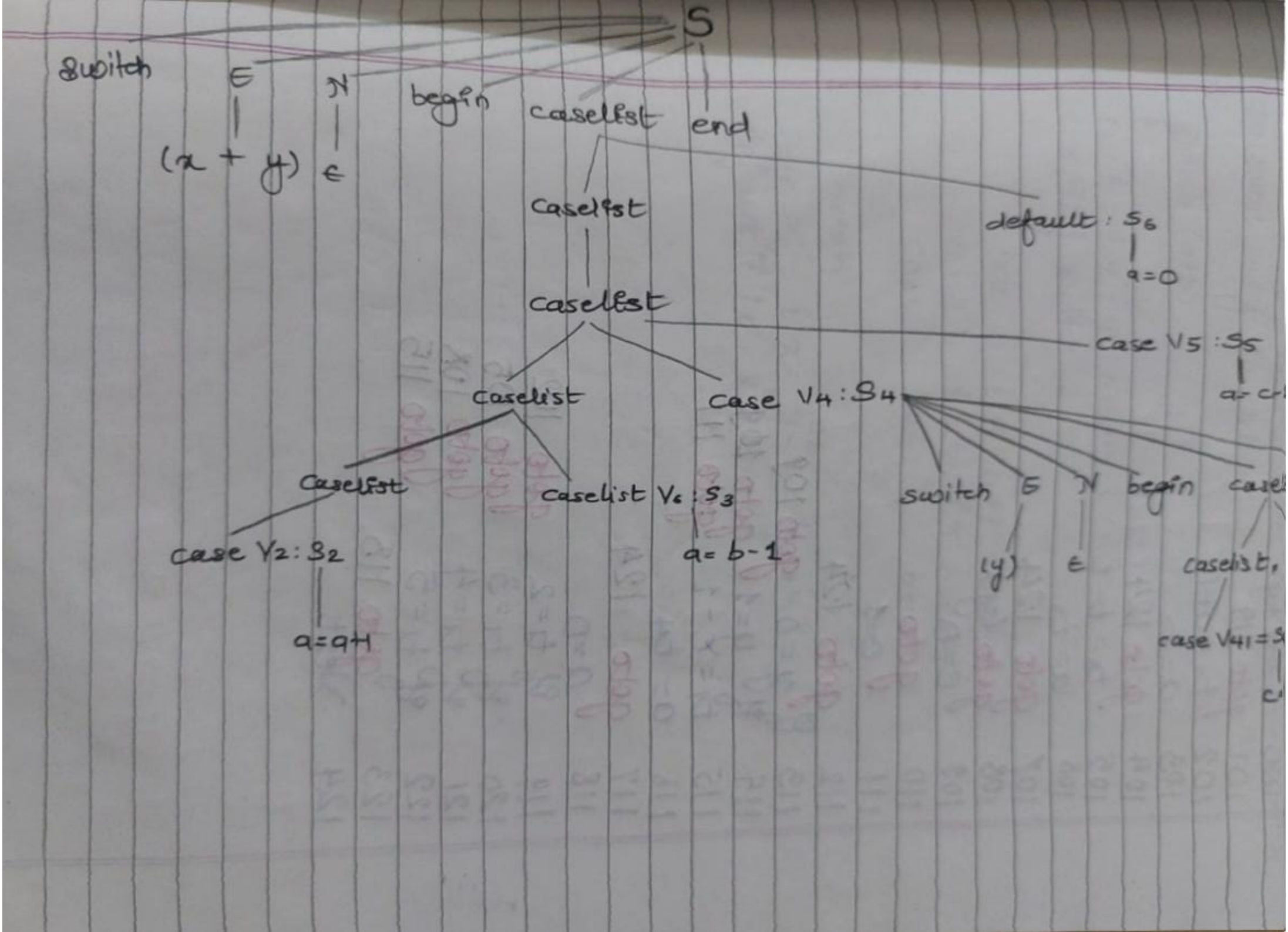
caselist → case V:S

caselist → caselist$_1$ case V:S

caselist → caselist$_1$ default:S

E → id relop id

N → ∈


S
switch
E
(x + y)
N
ε
begin
caselist
end
caselist
caselist
caselist
Caselist
Caselist
case V2 : S2
a = a+1
caselist Vc : S3
a = b-1
default : S6
a=0
case V5 : S5
case V4 : S4
switch
E
(y)
N
ε
begin
caselist,
case V41 = S

```
100  t1 = x+y
101  goto 119
102  t2 = a+1
103  a = t2
104  goto 124
105  t3 = b-1
106  a = t3
107  goto 124
108  goto (y)
109  c = 0
110  goto
111  c = 1
112  goto 124
113  if y=0 goto 109
114  if y=1 goto 111
115  t4 = c-1
116  a = t4
117  goto 124
118  a = 0
119  if t1=2 goto 102
120  if t1=3 goto 105
121  if t1=4 goto 108
122  if t1=5 goto 115
123  goto 118
124  next
```

HW

Date : / /201

Que Give the translation scheme for switch case statement used in C language. Translate the following switch statement using the scheme

```
switch (a+b)
{
case 2 : { x = y ; break ; } V2 (S1)
case 3 : { switch x V5
    {
      case 0 : { a = b+1 ; break ;   V0 S2
      case 1 : { a = b+3 ; break ;   V1 S3
      case 2 : { a = 2 ; break ;     V4 S4
    }
  break
case 3 : { x = y - 1 ; break }   V3 S5
default : { a = 2 ; break }      V6 S6
```

SDTS for Array Reference

Grammar (left side → right side)

array → A → L = E

L → elist] / id

elist → elist , / id [E

E → E+E | (E) | L

If not given assume bpw of 4 bytes per word

SDTS:

A → L = E

{

if L.offset = Null then

Gen (L.place = E.place

else

Gen (L.place [L.offset] = E.place

}

L → elist]

{

T = Newtemp()

U = Newtemp()

Gen (T = addr (elist.name) - C)

Gen (U = elist, place * bpw)

L.place = T

L.offset = U

}

L → id

{

L.place = id.place

L.offset = Null

}

Page No.
Date : / /201

elist → elist1, E
{
elist.name = elist1.name
elist.NDIM = elist1.NDIM + 1
T = Newtemp()
T = Limit(elist.name, elist1.NDIM + 1)
Gen(T = elist1.place * T)
Gen(T = T + E.place)
elist.place = T
}

elist → id[E
{
elist.place = E.place
elist.NDIM = 1
elist.name = id.name
}

E → E1 + E2
{
T = Newtemp()
Gen(T = E1.place + E2.place
E.place = T
}

E → (E1)
{
E.place = E1.place
}

E → L
{
if L.offset = Null then
E.place = L.place
else
begin
T = Newtemp()
Gen(T = L.place[L.offset])
E.place = T
end
}

61
Q

A = B[i]
A[i] = B[i] + c[j]
A[I, j] = B[I, J]
A[i, j, K] = B[I, J, K]
L R

Formula to FIND TAC

① for One dimensional array.
eg. A[i]

TAC: $t_1 = i * bpw$
$t_2 = addr(A)$
$t_3 = t_2[t_1]$

② For two dimensional array

eg. A[i, j] or A[i][j]
$d_1 * d_2$

TAC:
$t_1 = i * d_2$
$t_2 = t_1 + J$
$t_3 = t_2 * bpw$
$t_4 = addr(A) - C$
$t_5 = t_4[t_3]$

where
$C = (d_2 + 1) * bpw$

③ For three dimensional array

eg A[i, j, K] or A[i][j][K]
$d_1 * d_2 * d_3$

TAC

$t_1 = i * t_2$

$t_2 = t_1 - J$

$t_3 = t_2 * d_3$

$t_4 = t_3 + K$

$t_5 = t_4 * bpw$

$t_6 = addr(A) - C$

$t_7 = t_6 [t_5]$

where

$C [(d_2 * d_3) + 1) * bpw$

Problem: Translate the following assignment statement to intermediate code using array reference scheme

$B = A[I, J]$

A is an array of size 10×20

Assume bpw = 4

Sol: Generate Grammar

I $A \rightarrow L = E$

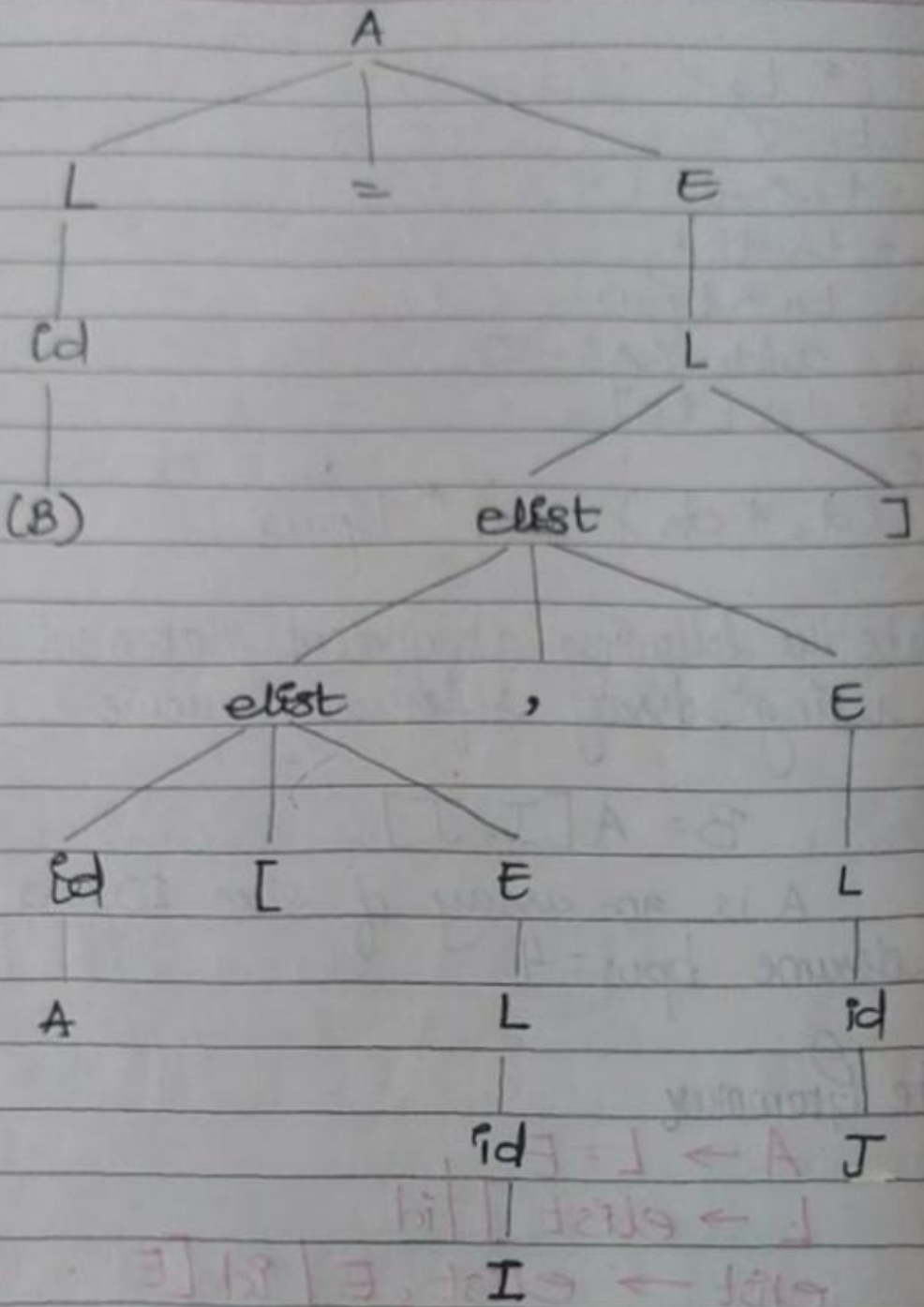
$L \rightarrow elist\ ] \mid id$

$elist \rightarrow elist, E \mid id\ [E$

$E \rightarrow E + E \mid (E) \mid L$

Parse Tree.

P.T.O.



(3) TAC

$t_1 = i * d_2$

$t_2 = t_1 + J$

$t_3 = d_2 * bpw$

$t_4 = addr(A) - C$

$t_5 = t_4[t_3]$

where

$C = (d_2 + 1) * bpw$

TAC

100 $t_1 = I * 20$
101 $t_2 = t_1 + J$
102 $t_3 = t_2 * 4$
103 $t_4 = addr(A) - C$ (84)
104 $t_5 = t_4[t_3]$
105 $B = t_5$

where

$C = (d_2 + 1) * bpw$
$= (20 + 1) * 4$

$\boxed{C = 84}$.

Q2. $A[i,j] = B[i,j]$

A → Two dimensional assay of size 10×10
B → Two dimensional assay of size 10×20

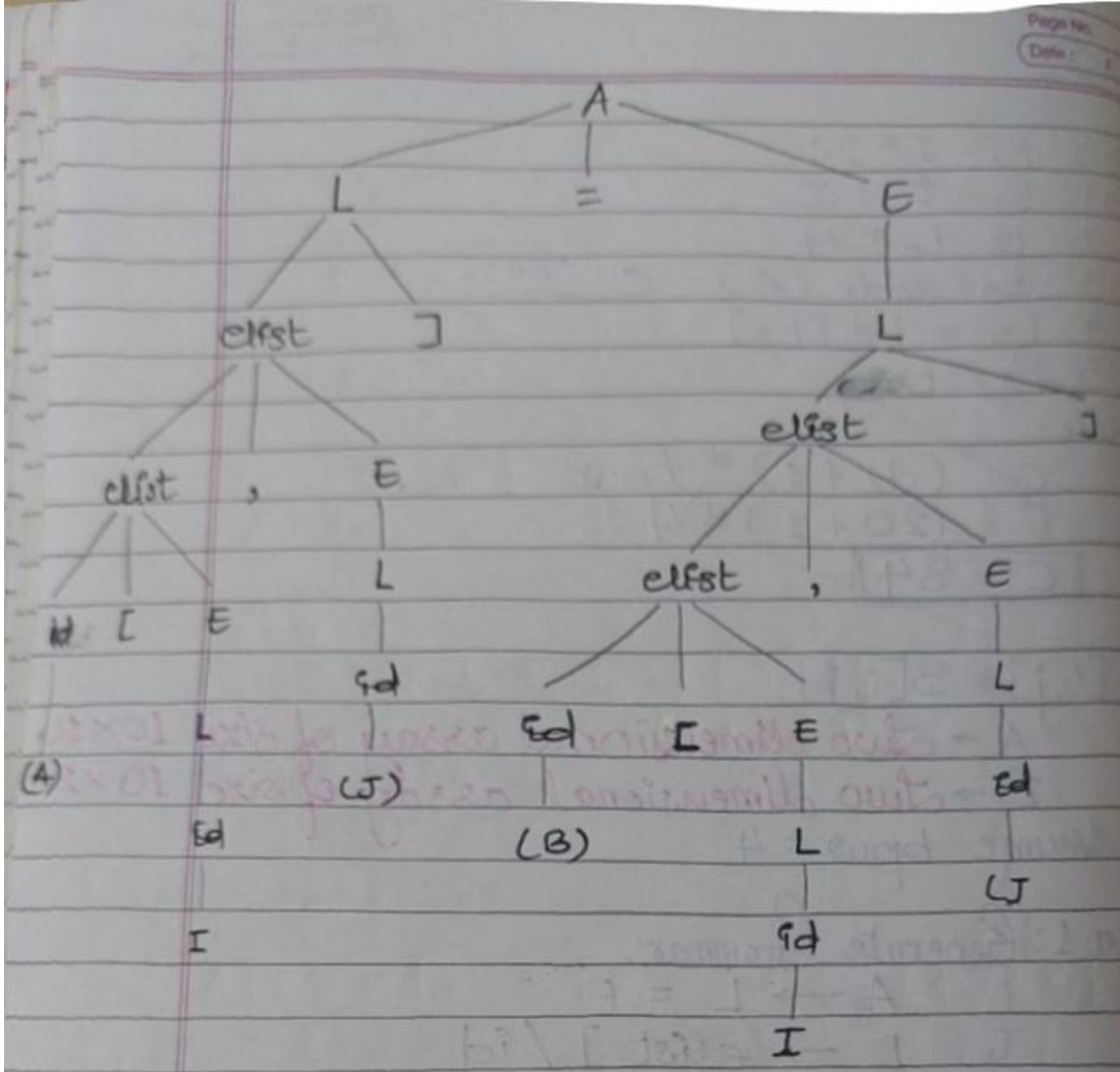
Assume bpw = 4

Ans Step 1: Generate Grammar.

$A \longrightarrow L = E$
$L \rightarrow elist\ ] \mid id$
$elist \rightarrow elist, E \mid id\ [E$
$E \longrightarrow E+E \mid (E) \mid L$

Step 2:

Parse Tree:-



③ Three address Code

100 $t_1 = i * 10$
101 $t_2 = t_1 + J$
102 $t_3 = t_2 * 4$
103 $t_4 = addr(A) - C_A$ (4a)
104 $t_5 = t_4 [t_3]$
105 $t_6 = i * 20$
106 $t_7 = t_6 + J$
107 $t_8 = t_7 * 4$
108 $t_9 = addr(B) - C_8$
109 $t_{10} = t_9 [t_8]$
110 $t_5 = t_{10}$

Value

$C_A = (d_2 + 1) * bpw$

$= (10 + 1) * 4$

$= 44.$

$C_B = (d_2 + 1) * bpw$

$= (20 + 1) * 4$

$= 84.$

Problem: $A = B[I, J, K]$ $\quad d_1 \times d_2 \times d_3$ $\quad 10 \times 20 \times 30$ $\quad bpw = 4.$

Sol: Generate Grammar

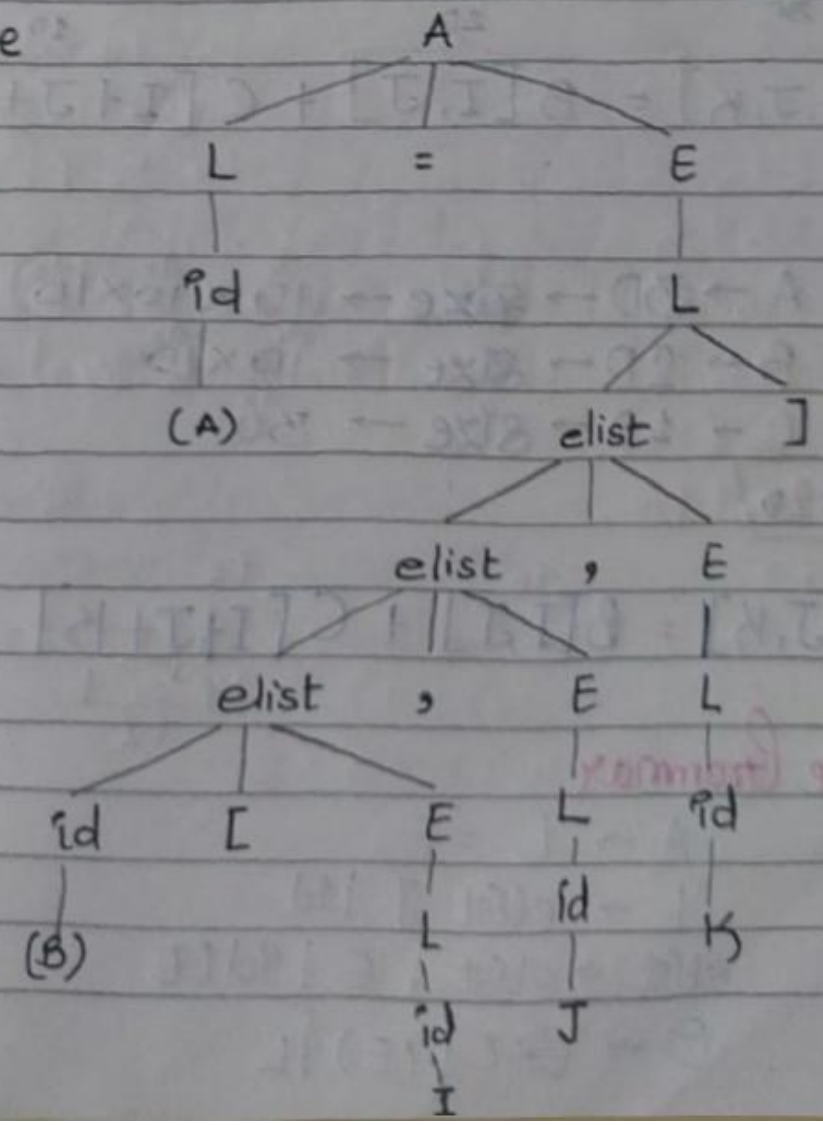
$A \rightarrow L = E$

$L \rightarrow elist\ ] \mid id$

$elist \rightarrow elist\ ,\ E \mid id\ [\ E$

$E \rightarrow E + E \mid (E) \mid L$

Parse Tree



3 | IAC

100 $t_1 = i * 20$
101 $t_2 = t_1 + J$
102 $t_3 = t_2 * 30$
103 $t_4 = t_3 + K$
104 $t_5 = t_4 * 4$
105 $t_6 = addr(B) - C$
106 $t_7 = t_6[t_5]$
107 $A = t_7$

where

$C = ((d_2 * d_3) + 1) * bpw$

$C = ((20 \times 30) + 1) * 4$

$C = \underline{2404}$

Problem (Very Imp)

$$\overbrace{A[I,J,K]}^{3D} = \overbrace{B[I,J]}^{2D} + C\overbrace{[I+J+K]}^{1D}$$

$A \rightarrow 3D \rightarrow size \rightarrow 10^{d_1} \times 10^{d_2} \times 10^{d_3}$

$B \rightarrow 2D \rightarrow size \rightarrow 10^{d_1} \times 10^{d_2}$

$C \rightarrow 1D \rightarrow size \rightarrow 30^{d_3}$

BPW: 2

$$A^{t_4}[I,J,K] = B^{t_6}[I,J] + C[\underbrace{I+J}_{t_1}+K]^{t_2}$$

① Generate Grammar

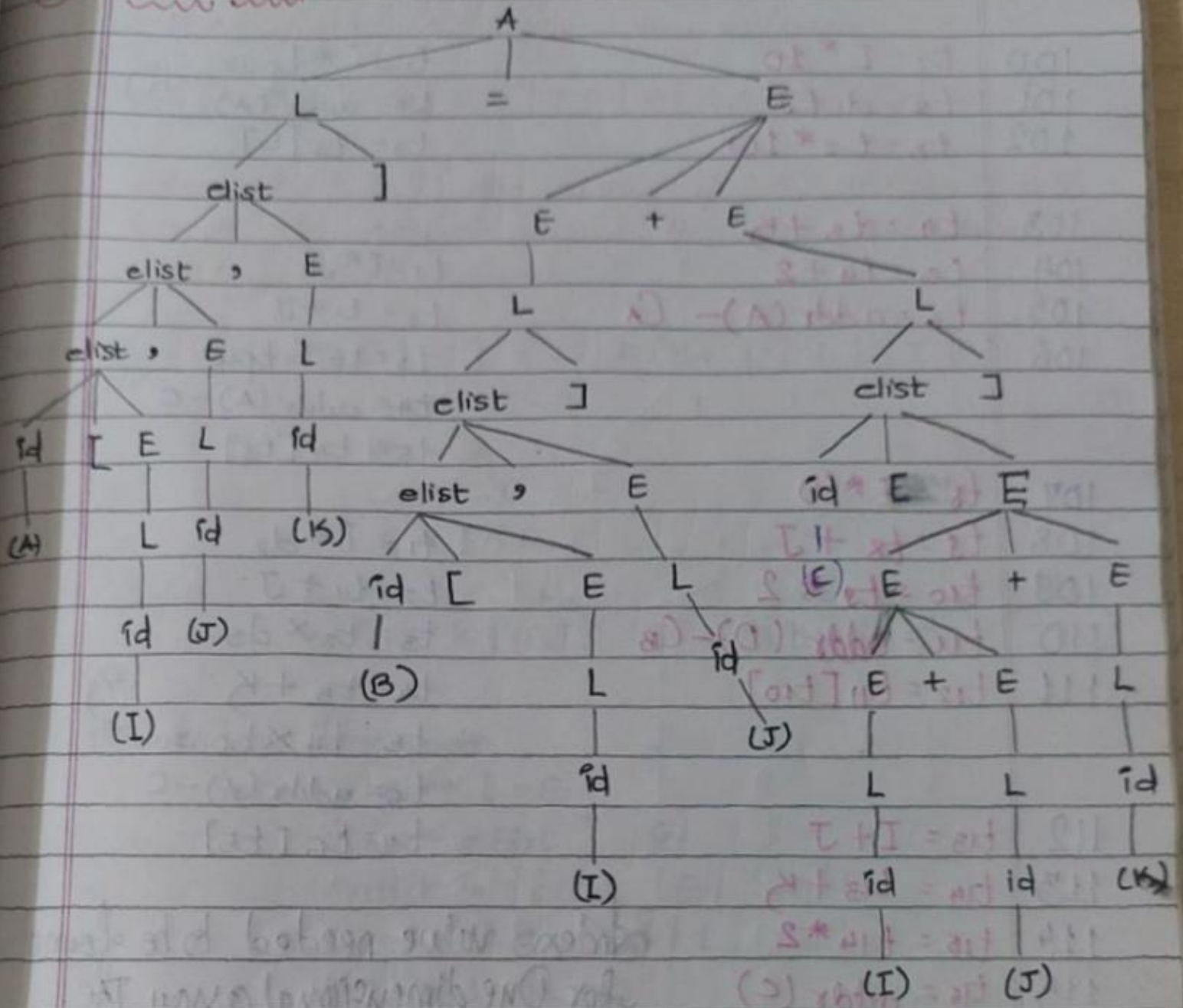
$A \rightarrow L = E$

$L \rightarrow elist\ ] \mid id$

$elist \rightarrow elist, E \mid id[E$

Page No.
Date : / /201

② Parse tree



1 dimensional = 3 variable
2 dimensional = 5 variable
3 dimensional = 7 variable

6M
Q

TAC

100	$t_1 = I * 10$	$t_1 = i * bpw$
101	$t_2 = d_1 + J$	$t_2 = addr(A)$
102	$t_3 = t_2 * 10$	$t_3 = t_2[t_1]$
103	$t_4 = t_3 + K$	
104	$t_5 = t_4 + 2$	$t_1 = i * d_2$
105	$t_6 = addr(A) - C_A$	$t_2 = t_1 + J$
106		$t_3 = t_2 * bpw$
		$t_4 = addr(A) - C$
		$t_5 = t_4[t_3]$
107	$t_8 = I * 10$	
108	$t_9 = t_8 + J$	$t_1 = i * d_2$
109	$t_{10} = t_9 * 2$	$t_2 = t_1 + J$
110	$t_{11} = addr(D) - C_B$	$t_3 = t_2 \times d_3$
111	$t_{12} = t_{11}[t_{10}]$	$t_4 = t_3 + K$
		$t_5 = t_4 \times bpw$
		$t_6 = addr(A) - C$
112	$t_{13} = I + J$	$t_7 = t_6[t_5]$
113	$t_{14} = t_{13} + K$	
114	$t_{15} = t_{14} * 2$	
115	$t_{16} = addr(C)$	
116	$t_{17} = t_{16}[t_{15}]$	
117	$t_{18} = t_{12} + t_{17}$	
118	$t_7 = t_{18}$	

Index value needed to be store for One dimensional assay TAC

$$C_A = ((d_2 \times d_3) + 1) * bpw$$
$$= ((10 \times 10) + 1) * 2$$
$$\boxed{C_A = 202}$$

$$C_B = (d_2 + 1) * bpw$$
$$= (10 + 1) * 2$$

Date : / /201

Problem Translate the following assignment statement of intermediate code using array reference

$$A[I,J] = B[I,J] + C[A[K,L]] + D[I+J]$$

(8 M)

where A, B, C, D are array of 2×3, 4×5, 6 & 7 respectively. Assume bpw = 4. Draw Annotated Parse Tree for the same.

Ans $A[I,J] = B[I,J] + C[A[K,L]] + D[I,J]$

A → 2×3

B → 4×5

C → 6

D → 7

logic $A[I,J] = B[I,J]^{t_B} + C[A[K,L]]^{t_C} + D[I+J]^{t_D}$

Generate Grammar

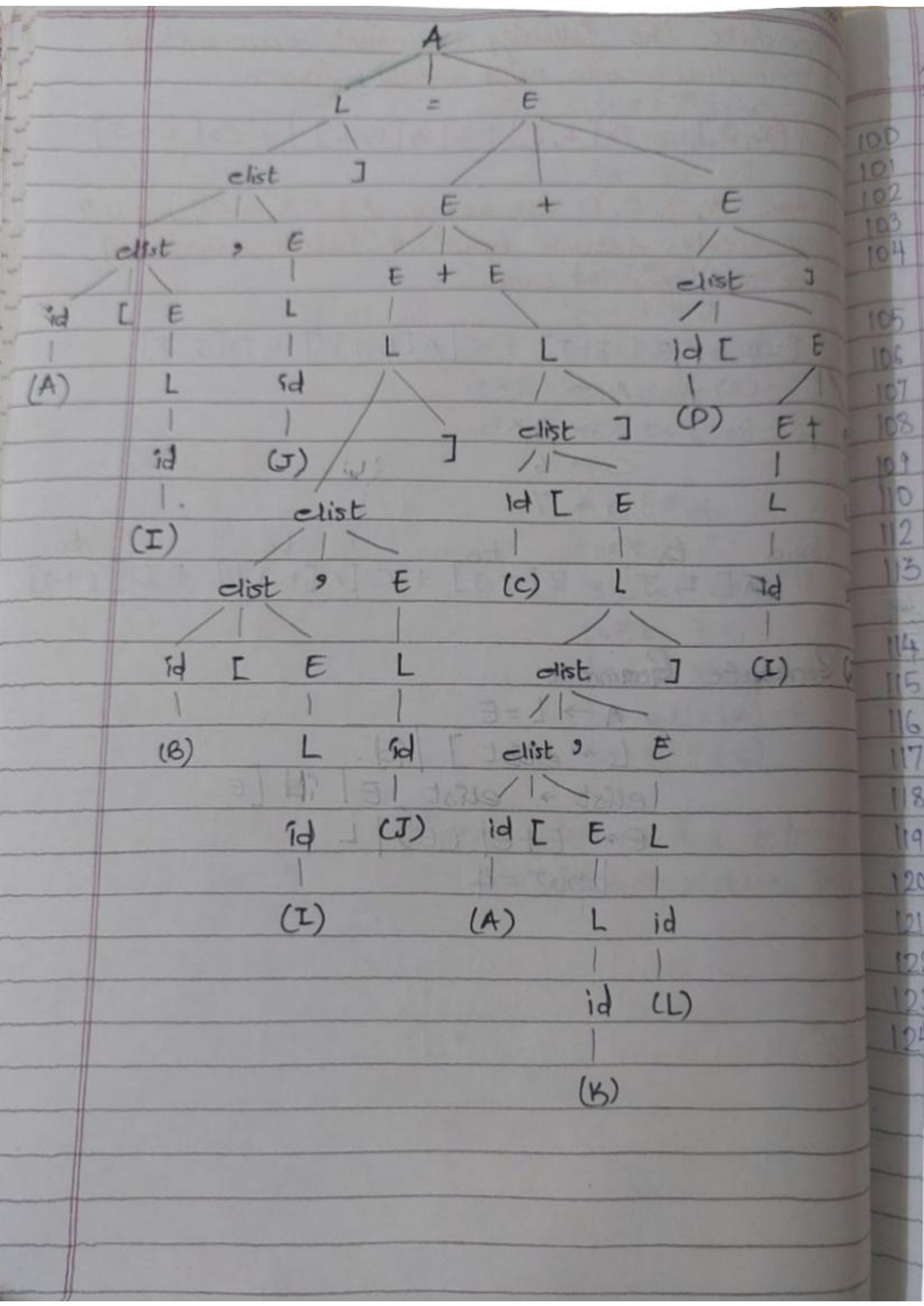
$A \rightarrow L = E$

$L \rightarrow elist\ ] \mid id$

$elist \rightarrow elist, E \mid id\ [E$

$E \rightarrow E+E \mid (E) \mid L$

bpw = 4


A
L = E
elist]
elist , E
id [E
(A)
L
id
(I)
elist
elist , E
id [E
(B)
L
id
(I)
id
(A)
E + E
L L
elist]]
id (J)
id [E
(C)
L
id
(I)
id [E L
(A)
L id
id (L)
(K)
E + E
elist]
id [E
(D)
E +
L
id
(J)

TAC

```
100  t1 = I * 3
101  t2 = t1 + J
102  t3 = t2 * 4
103  t4 = addr (A) - CA
104  t5 = t4 [t3]

105  t6 = I * 5
106  t7 = t6 + J
107  t8 = t7 * 4
108  t9 = addr (B) - CB
109  t10 = K * 3
110  t12 = t11 + L                    (J value is L)
112  t13 = t12 * 4
113  t14 = addr (A) - CA

114  t15 = t14 [t13]
115  t16 = t15 * 4
116  t17 = addr (C)
117  t18 = t17 [t16]
118  t19 = I + J
119  t20 = t19 * 4
120  t21 = addr (D)
121  t22 = t21 [t20]
122  t23 = t10 + t18
123  t24 = t23 + t22
124  t5 = t24
```

where

$C_A = (d_2 + 1) * bpw$

$= (3 + 1) * 4$

$= 16$

$C_B = (d_2 + 1) * bpw$

$= (5 + 1) * 4$

$= 6 * 4$

$= 24$

H.W.

Problem: $A[B[i,j], c[k]] = k$

$A \rightarrow 10 \times 20$

$B \rightarrow 10 \times 20$

$C \rightarrow 30$

Problem: $C[i,j,k] = a[b[i,j], k] + a[i,j]$

$a = 10 \times 20$

$C \rightarrow 10 \times 20 \times 30$

BPW = 4