诚信保证(Integrity commitment)

本人知晓我校考场规则和违纪处分条例的有关规定，保证遵守考场规则，诚实做人。　　　　本人签字(Signature)：

编号：

西北工业大学考试试题（A 卷）

2020 － 2021　学年第 2 学期

开课学院 School of Computer Science

课程 Compiler Principles/编译原理(英) (U10M12008.01)

学时 48 hours

考试日期 2021.6.27　考试时间 2 hours 小时　　考试形式（ 闭 ）卷

Question	Q1/10	Q2/10	Q3/7	Q4/8	Q5/15	Q6/15	Q7/10	Q8/15	Q9/10	Sum of Scores
Scores										

考生班级（Class ID）		学号（Student ID）	2018380130	姓名（Name）	Khan Md Shahedul Islam

【Q1】 (10:3+7 scores)

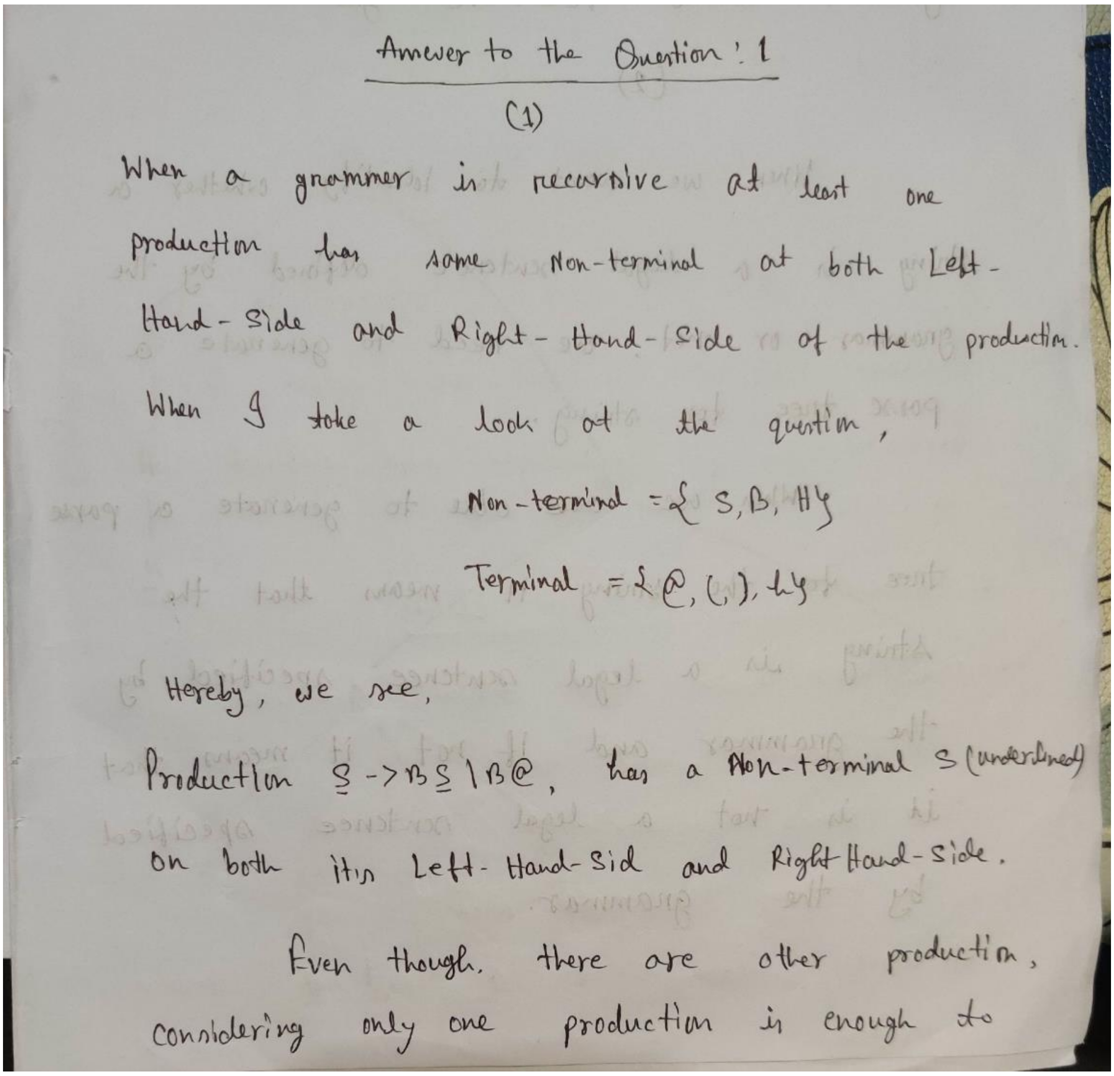
For grammar G[S]:

S→BS | B@，B→(BH)B | H，B→ε | b,　H→(H) | h | ε

Answer the following questions:

(1) Is this grammar a recursive grammar? Give the answer and the reason for your answer. (3 scores)

(2) Given a string “h(b(h))@” , is it a legal sentence defined by this grammar? Please draw the parsing tree for the input?　(7:4+3 scores).



Answer to the Question: 1

(1)

When a grammer is recarsive at least one production has same Non-terminal at both Left-Hand-Side and Right-Hand-Side of the production.

When I take a look at the question,

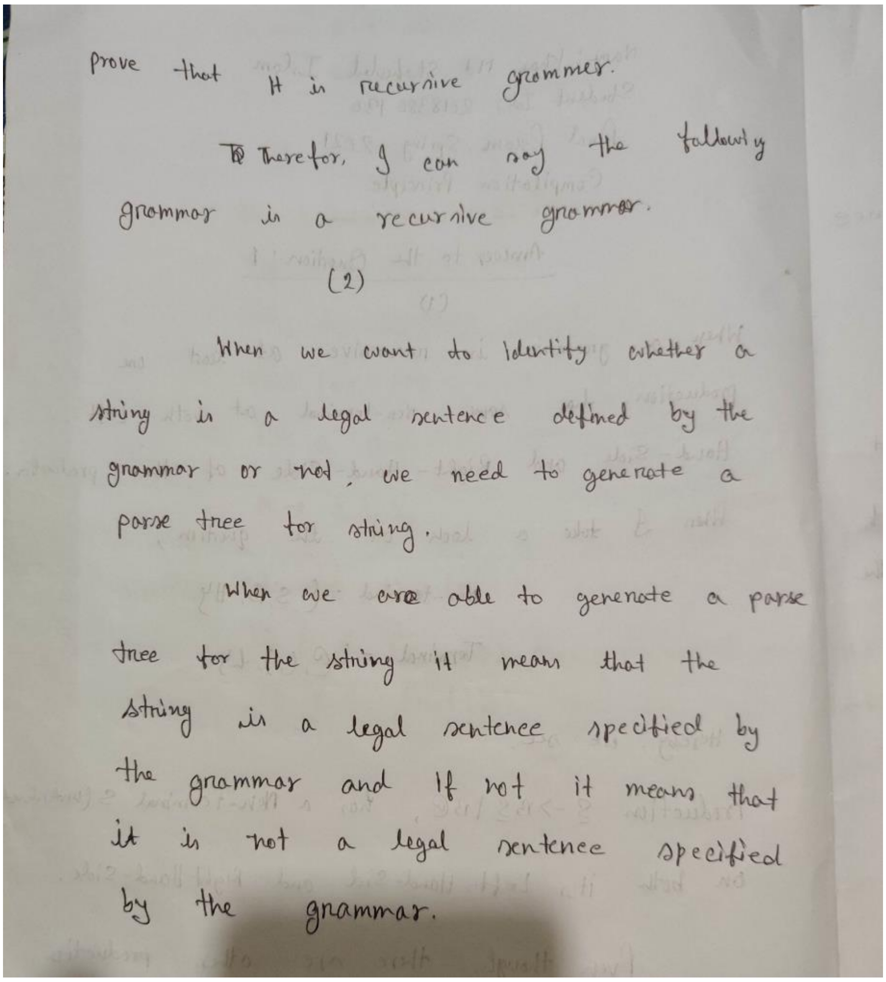
Non-terminal = {S, B, H}

Terminal = {@, (,), h}

Hereby, we see,

Production $\underline{S}$ -> B$\underline{S}$ | B@, has a Non-terminal S (underlined) on both its Left-Hand-Sid and Right Hand-Side.

Even though, there are other production, considering only one production is enough to



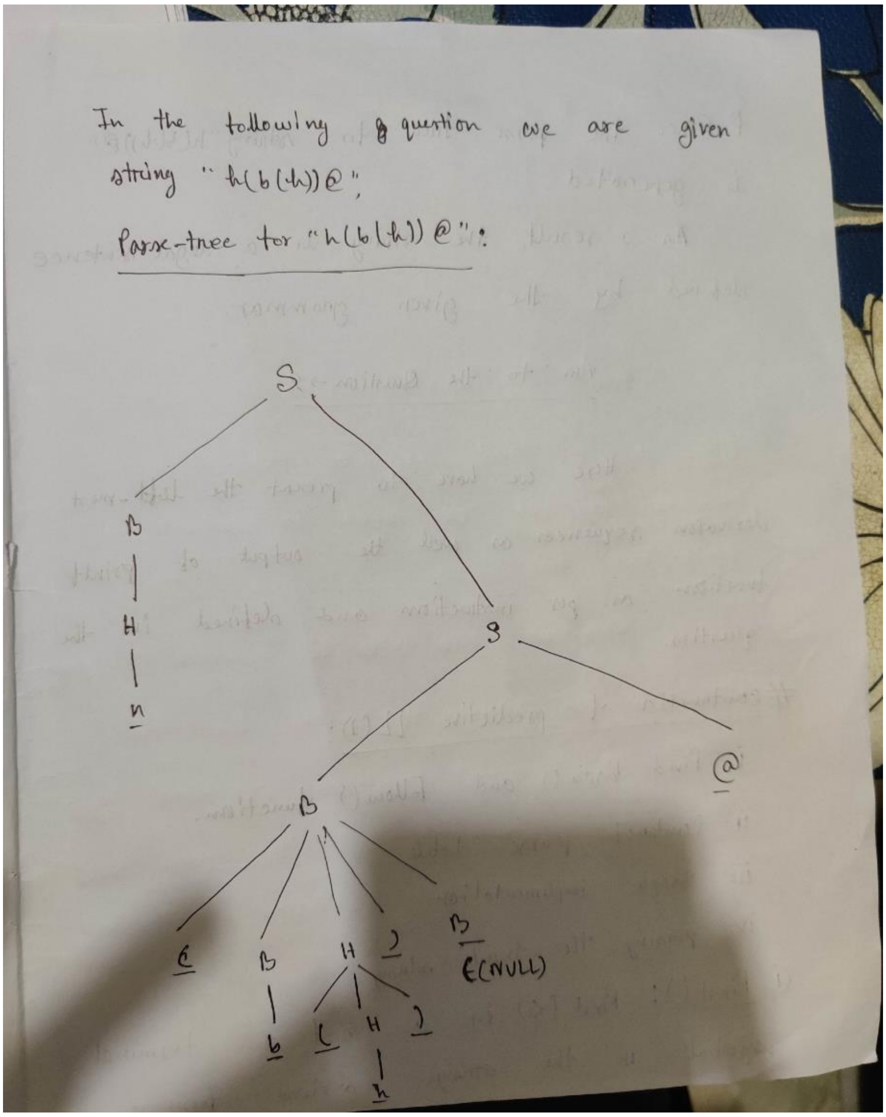
prove that It is recursive grommer.

~~T~~ Therefor, I can say the fallowing grammar is a recursive grammar.

(2)

When we want to Identify whether a string is a legal sentence defined by the grammar or not, we need to generate a parse tree for string.

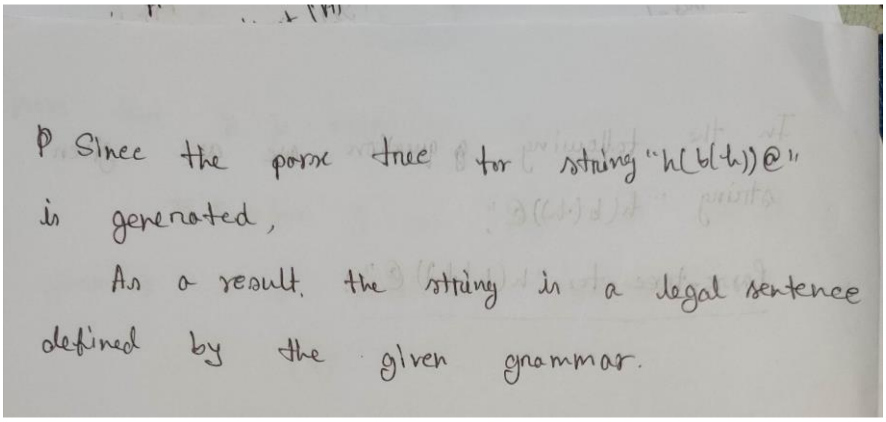
When we are able to generate a parse tree for the string it mean that the string is a legal sentence specified by the grammar and If not it means that it is not a legal sentence specified by the grammar.



In the following question we are given string "h(b(h))@";

Parse-tree for "h(b(h)) @":

S
B
H
h
S
@
B
(
B
H
)
B
b
(
H
)
h
ε(NULL)



Since the parse tree for string "h(b(h))@" is generated,

As a result, the string is a legal sentence defined by the given grammar.

【Q2】 (10 scores)

Given grammar G[S]:

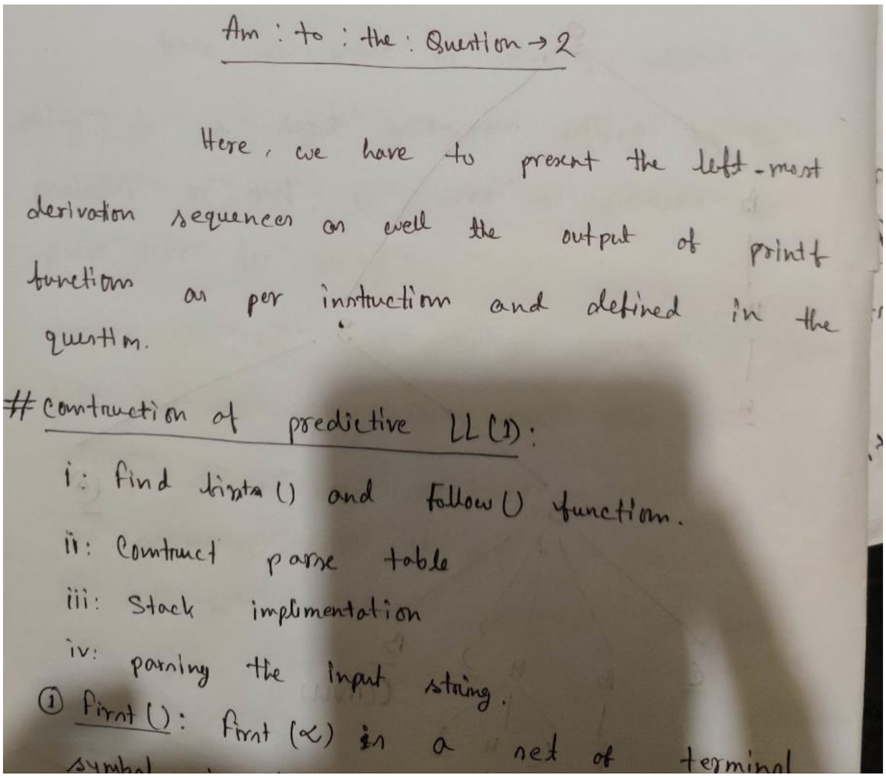
①S→A {printf(“1”);}　　②A→AaB {printf(“2”);}

③A→ε　{printf(“3”);}　　④B→hB {printf(“4”);}

⑤ B→ε　{printf(“5”);}

Given “aah” as the input string, and supposed that we use **LL(1)** similar parsing algorithm (derivation based analysis), please present the left-most derivation sequences as well as the output of printf functions defined in the question. (5 scores)

Notation: here we suppose to execute the printf when we use the rule to do derivation.



Ans : to : the : Question → 2

Here, we have to present the left-most derivation sequences as well the output of printf function as per instruction and defined in the question.

Contruction of predictive LL(1):

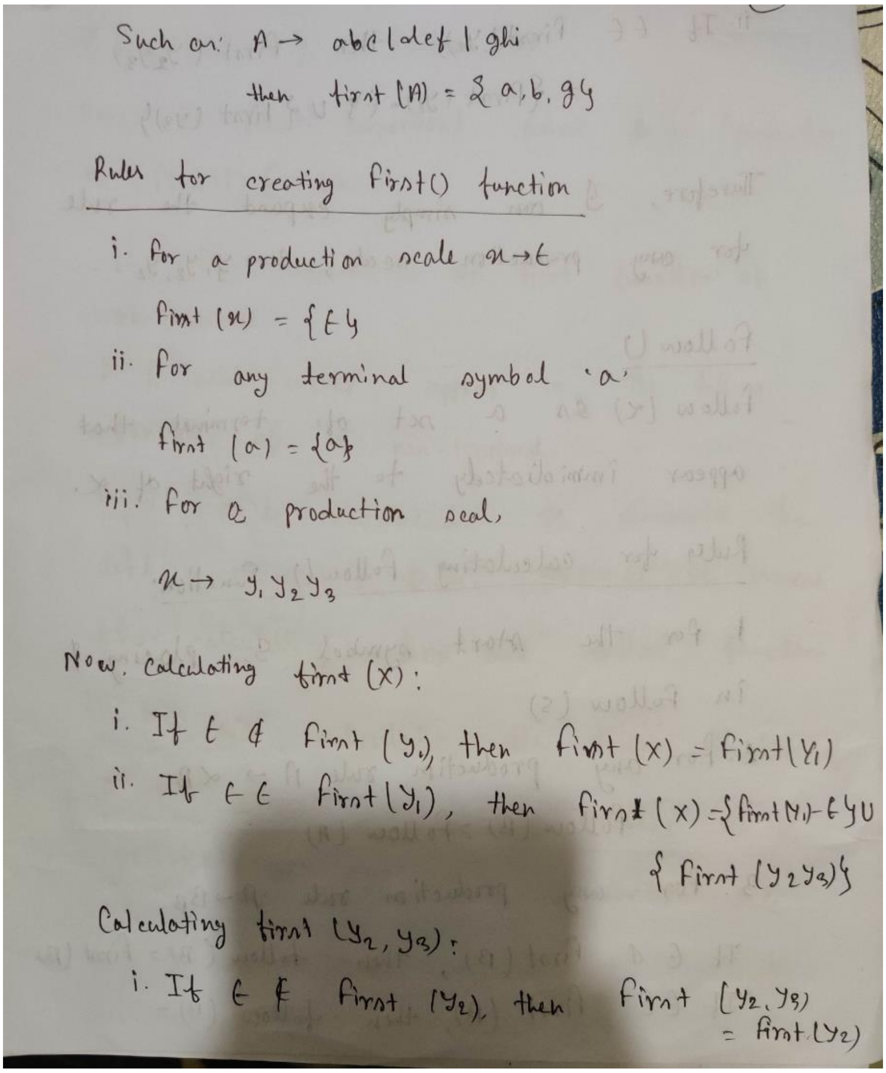
i: find firsta () and follow () function.

ii: Contruct parse table

iii: Stack implimentation

iv: parsing the input string.

① First (): First (α) is a set of terminal symbol



Such as: $A \rightarrow abc \mid def \mid ghi$

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

Rules for creating first() function

i. For a production rule $x \rightarrow \epsilon$

first $(x) = \{\epsilon\}$

ii. For any terminal symbol 'a'

first $(a) = \{a\}$

iii. For a production rule,

$x \rightarrow y_1 y_2 y_3$

Now, Calculating first (X):

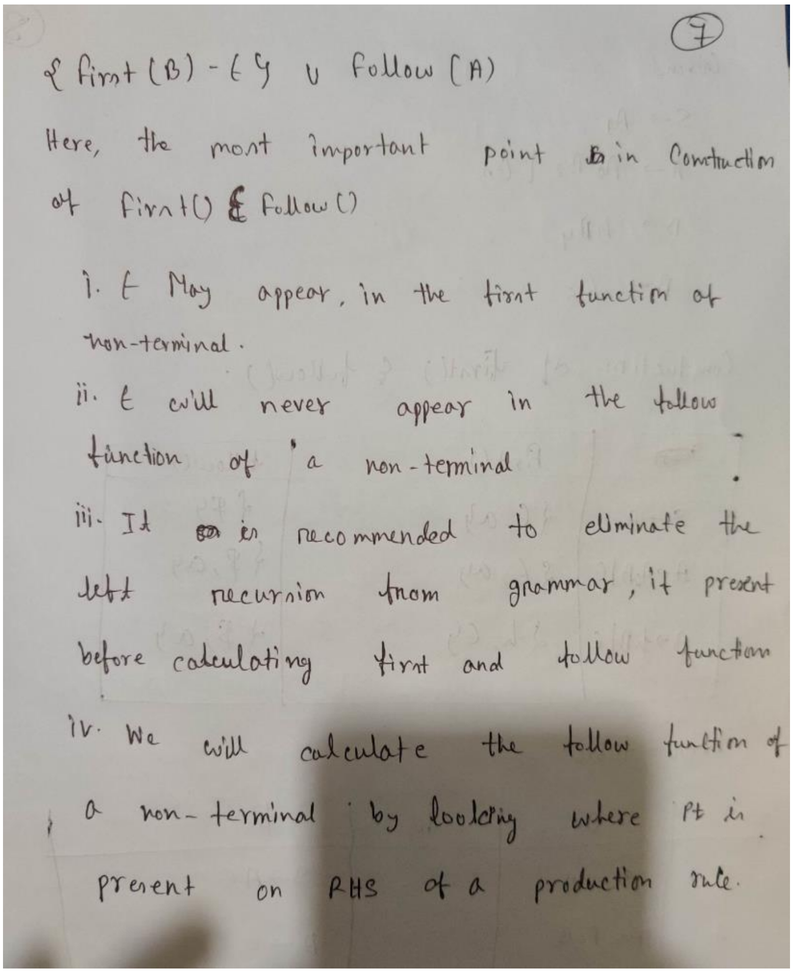
i. If $\epsilon \notin$ first (y_1), then first (X) = first(Y_1)

ii. If $\epsilon \in$ first(y_1), then first $(X) = \{$first$(Y_1) - \epsilon\} \cup$ $\{$ first $(y_2 y_3)\}$

Calculating first (y_2, y_3):

i. If $\epsilon \notin$ first (y_2), then first (y_2, y_3) = first (y_2)

6 / 30



$\{ \text{first}(B) - \epsilon \} \cup \text{follow}(A)$

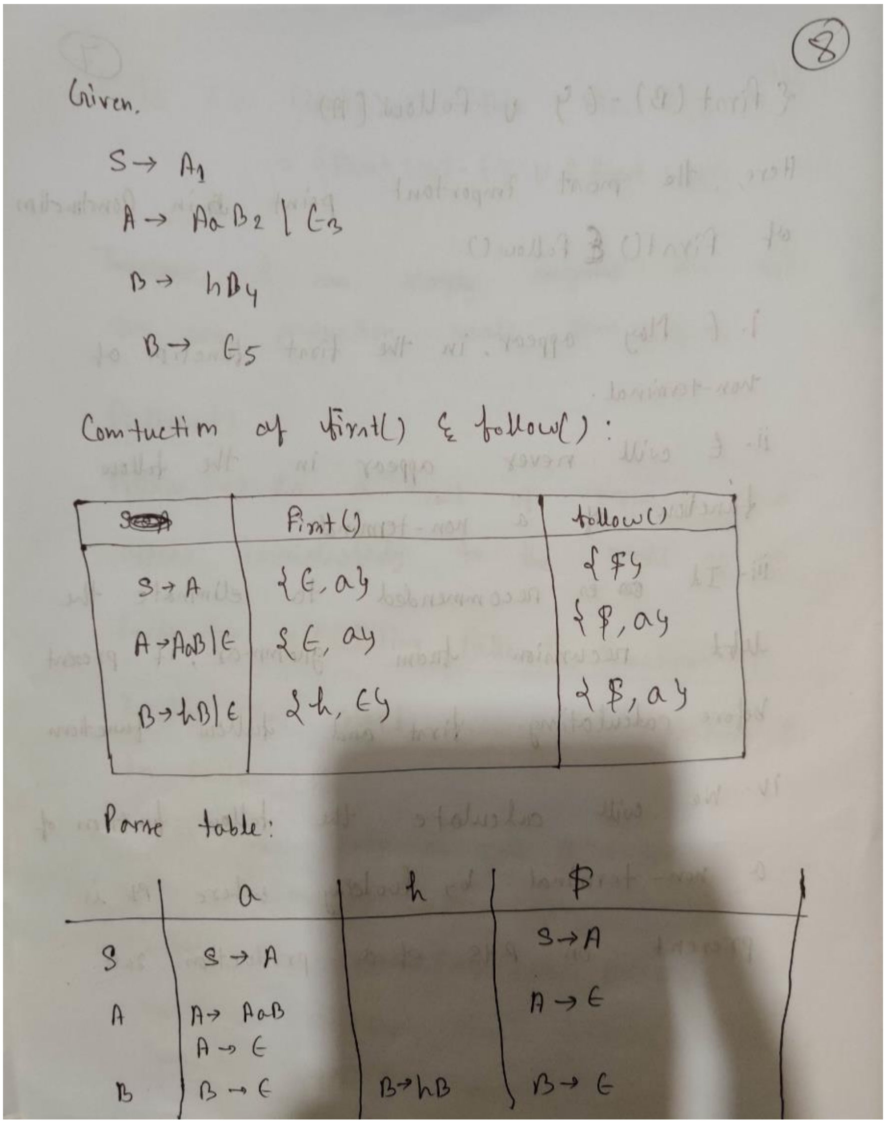
Here, the most important point in Construction of first() & follow()

i. ε May appear, in the first function of non-terminal.

ii. ε will never appear in the follow function of a non-terminal

iii. It is recommended to eliminate the left recursion from grammar, if present before calculating first and follow function

iv. We will calculate the follow function of a non-terminal by looking where it is present on RHS of a production rule.



8

Given,

$S \rightarrow A_1$

$A \rightarrow AaB_2 \mid \epsilon_3$

$B \rightarrow hB_4$

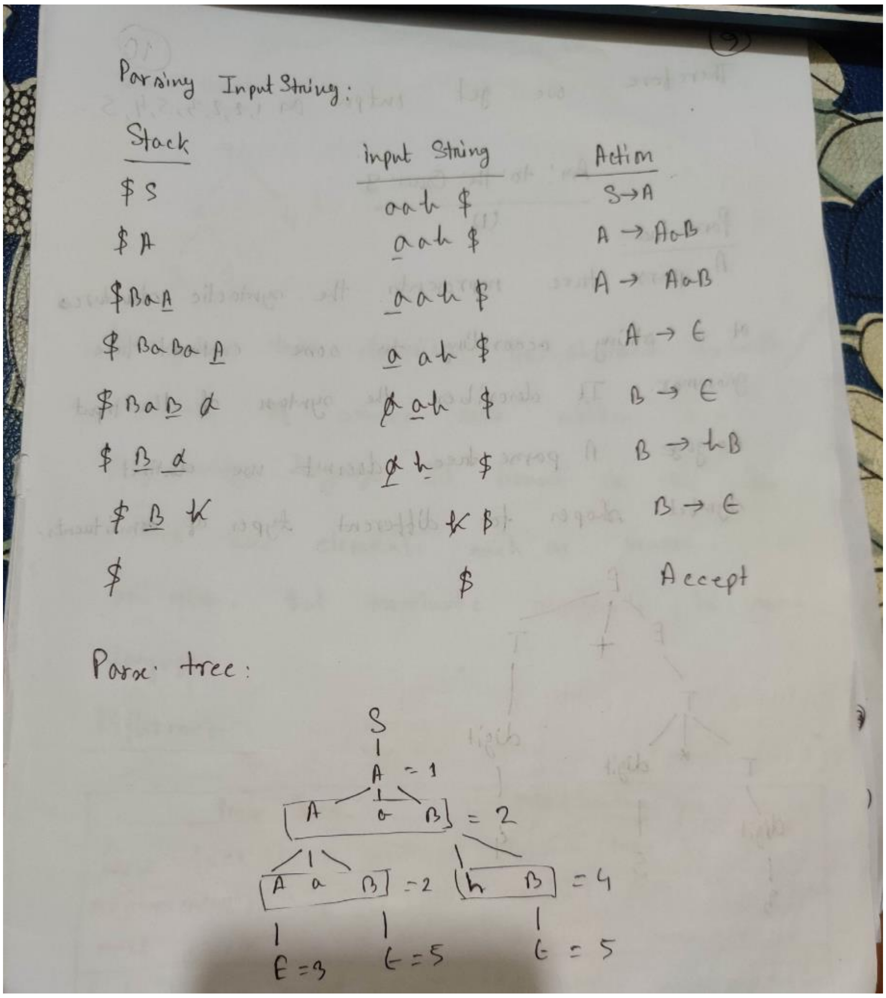
$B \rightarrow \epsilon_5$

Contuction of first() & follow():

	First()	follow()
S → A	{ε, a}	{$}
A → AaB | ε	{ε, a}	{$, a}
B → hB | ε	{h, ε}	{$, a}

Parse table:

	a	h	$
S	S → A		S → A
A	A → AaB A → ε		A → ε
B	B → ε	B → hB	B → ε



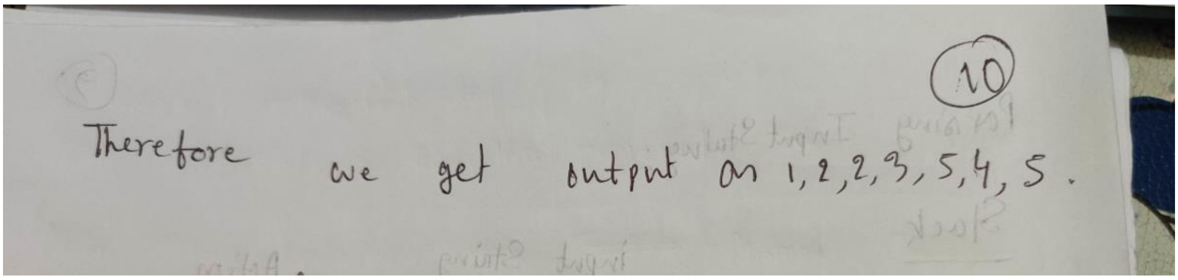
9

Parsing Input String:

Stack	Input String	Action
$S	aab $	S→A
$A	aab $	A → AaB
$BaA	aab $	A → AaB
$ BaBaA	a ab $	A → ε
$BaB ~~a~~	~~a~~ ab $	B → ε
$ B ~~a~~	~~a~~ b $	B → bB
$ B ~~b~~	~~b~~ $	B → ε
$	$	Accept

Parse tree:

S
A = 1
A a B = 2
A a B = 2
b B = 4
ε = 3
ε = 5
ε = 5



10

Therefore we get output as 1,2,2,3,5,4,5.

注：1. 答题请写在该试卷上相应位置。

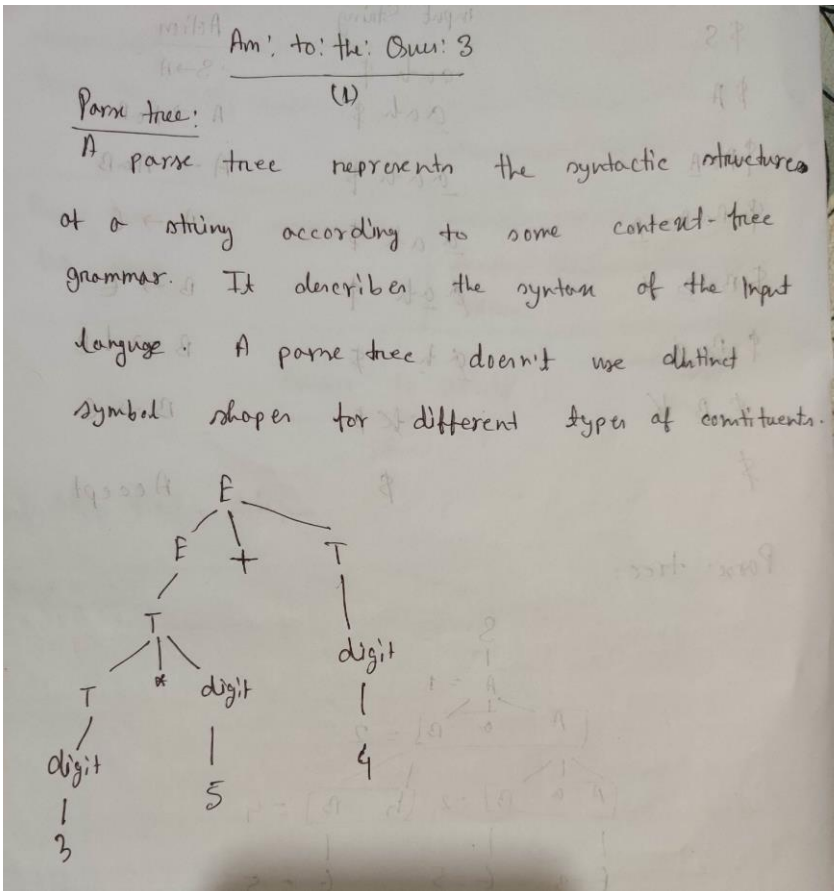
2. 命题教师和审题教师姓名应在试卷存档时填写。

教务处印制

【Q3】Answer the following questions: (7: 3.5+3.5 scores)

(1) Describe the difference between parsing tree and abstract syntax tree.

(2) Explain why intermediate representation (IR) were used in many compilers (explain the bennefits!).



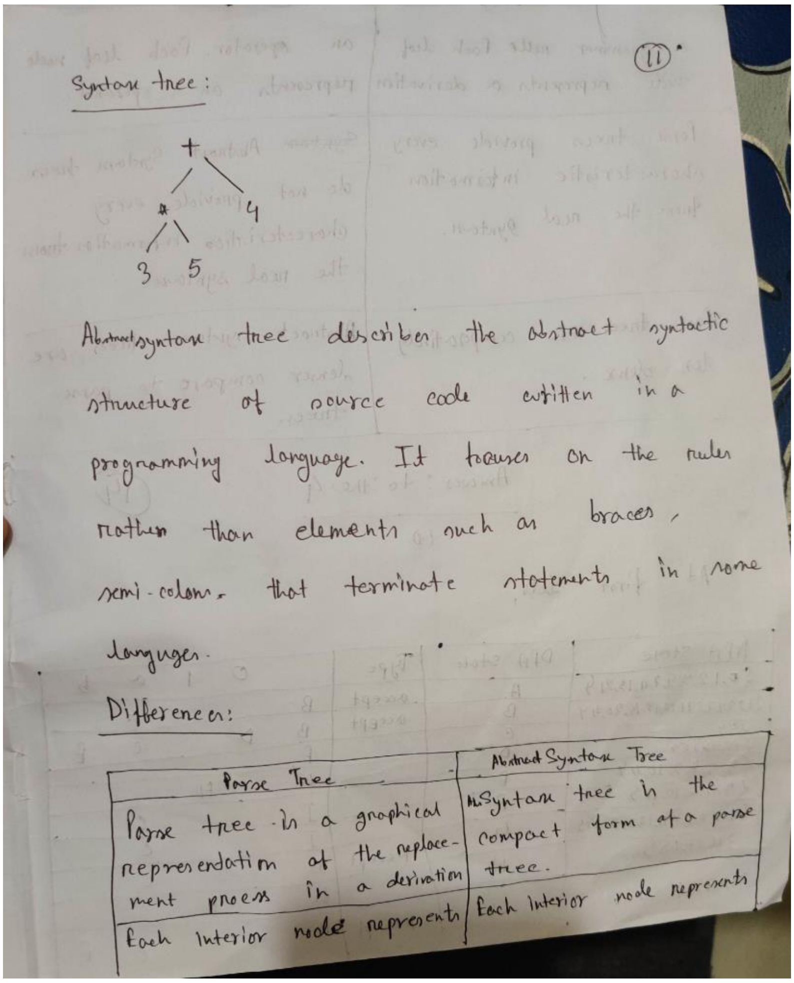
Ans: to: the: Ques: 3

(1)

Parse tree:

A parse tree represents the syntactic structures of a string according to some context-free grammar. It describes the syntax of the Input languge. A parse tree doesn't use distinct symbol shapes for different types of constituents.

E
E
+
T
T
*
digit
T
digit
3
5
digit
4



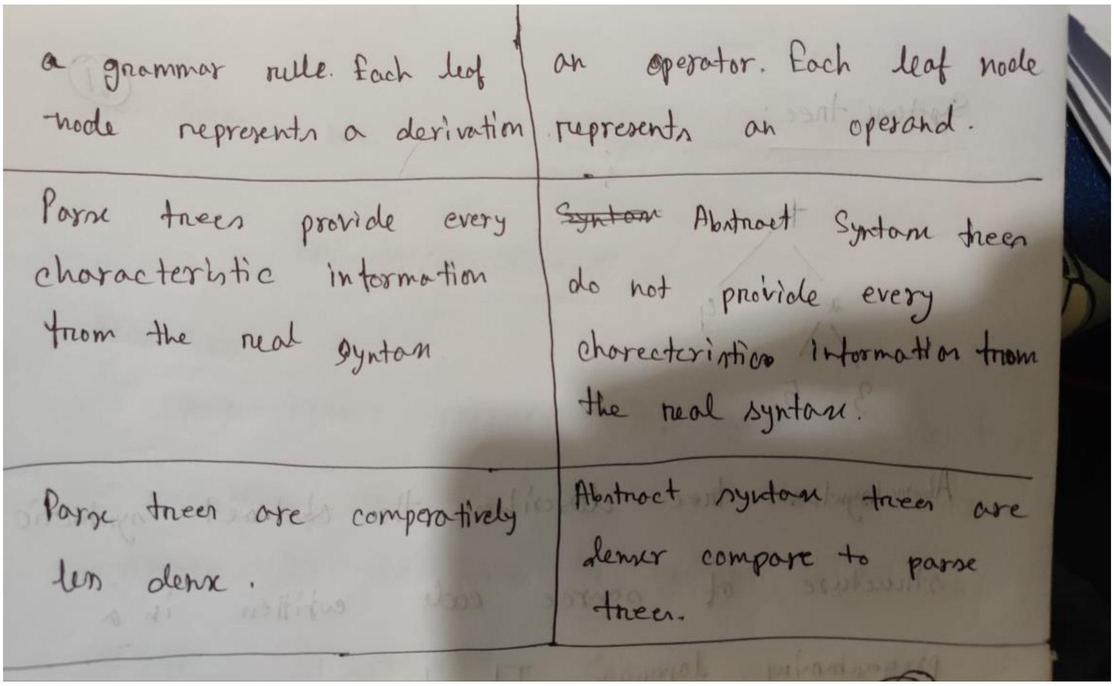
11

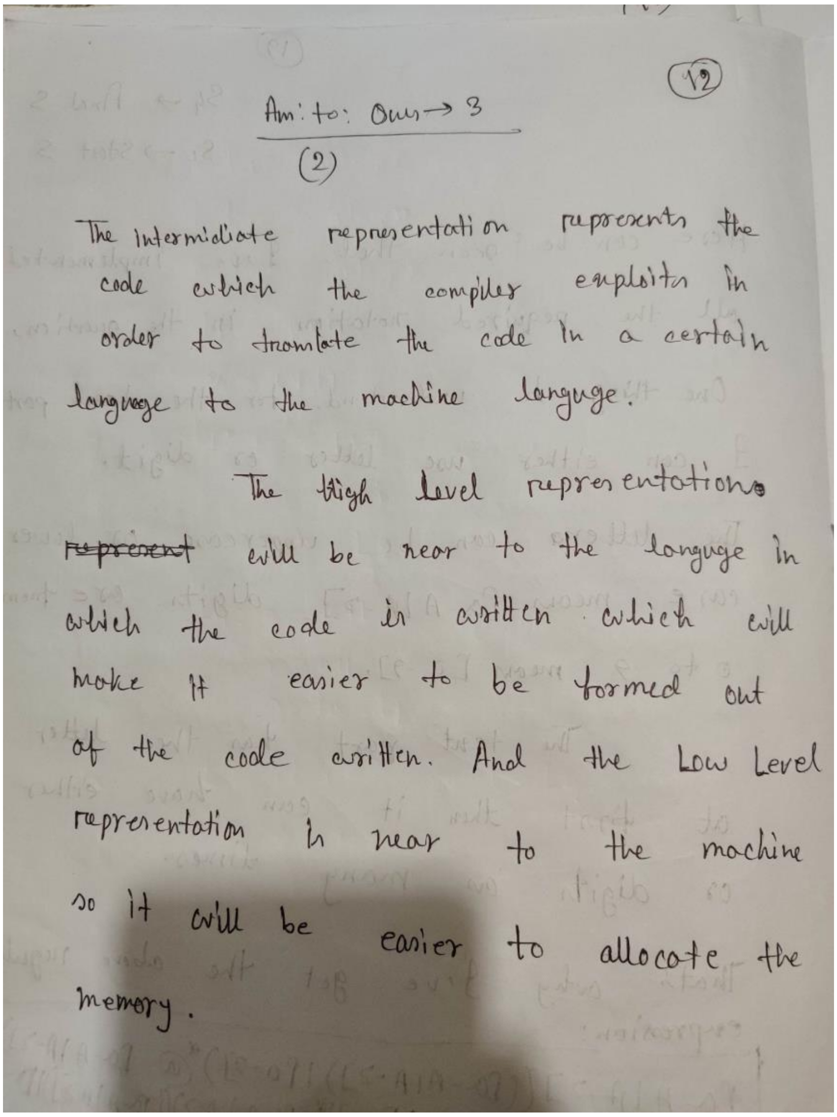
Syntax tree :

Abstract syntax tree describes the abstract syntactic structure of source code cwritten in a programming language. It focuses on the rules rather than elements such as braces, semi-colons. that terminate statements in some languges.

Differences:

Parse Tree	Abstract Syntax Tree
Parse tree is a graphical representation of the replacement process in a derivation	Syntax tree is the compact form of a parse tree.
Each interior node represents a grammar rule. Each leaf node represents a derivation	Each interior node represents an operator. Each leaf node represents an operand.
Parse trees provide every characteristic information from the real syntax	~~Syntax~~ Abstract Syntax trees do not provide every charecteristics information from the real syntax.
Parse trees are comperatively less dense.	Abstract syntax trees are denser compare to parse trees.





(12)

Ans: to: Ques → 3

(2)

The intermidiate representation represents the code which the compiler exploits in order to tromlate the code in a certain language to the machine languge.

The High level representations ~~represent~~ will be near to the languge in which the code is written which will make it easier to be formed out of the code written. And the Low Level representation is near to the machine so it will be easier to allocate the memory.

【Q4】Please answer the following questions: (8: 4+4 scores)

(1) Given regular expression ((a | b)* | 01*)*, please draw the NFA.

(2) Write down the regular expression or NFA or DFA for the following language:

A language could send email to serveral email addresses at the same time, e.g: abcd@abc.com，xyz@nnn.edu，uvw@eee.edu.cn

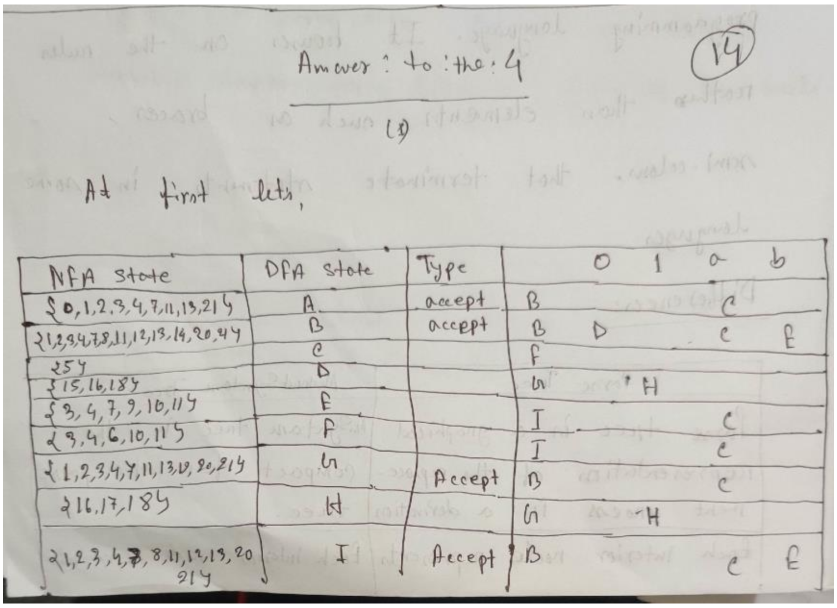
The whole expression could be representated as:

{ abcd, xyz, uvw } @ {abc.com, nnn.edu, eee.edu.cn}

Notations:

All text parts (except @) could be defined only with letters and digits while started with at least one letter.

After @, at least one ‘.’ must be included. That is abc@x.com is ok, but abc@ and abc@a and abc@a. are illegal.



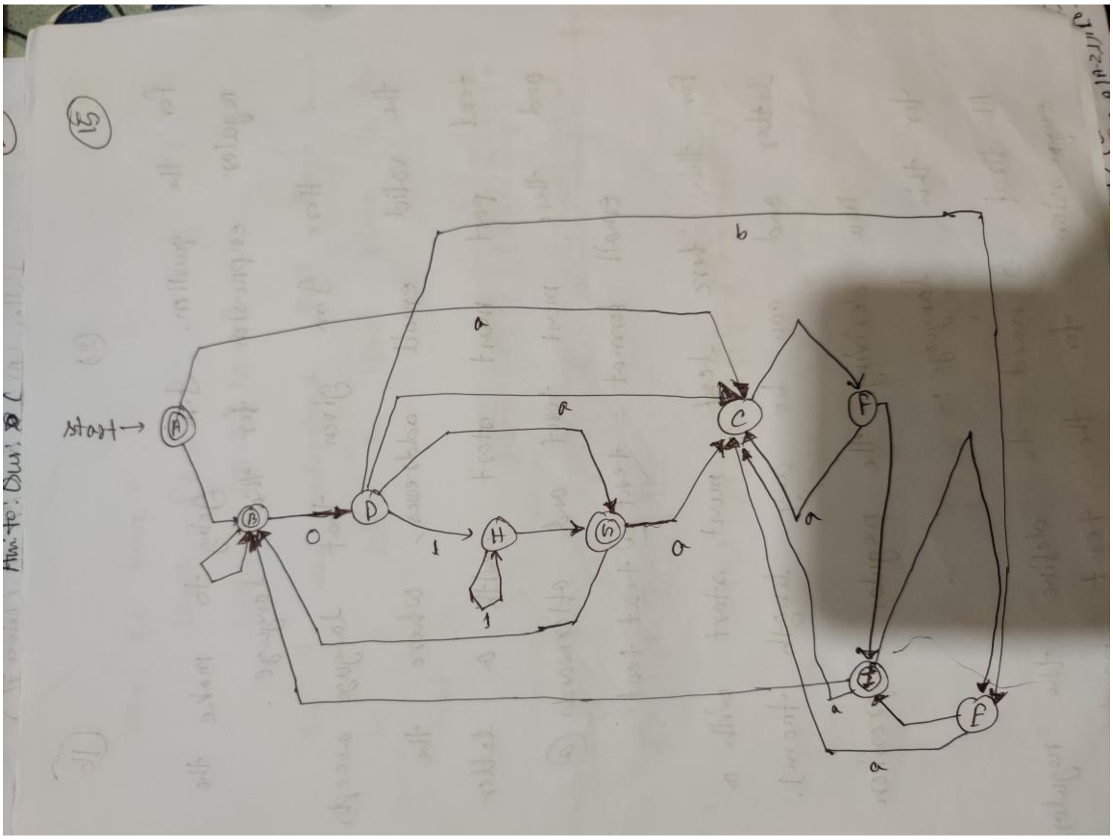
Answer : to : the : 4

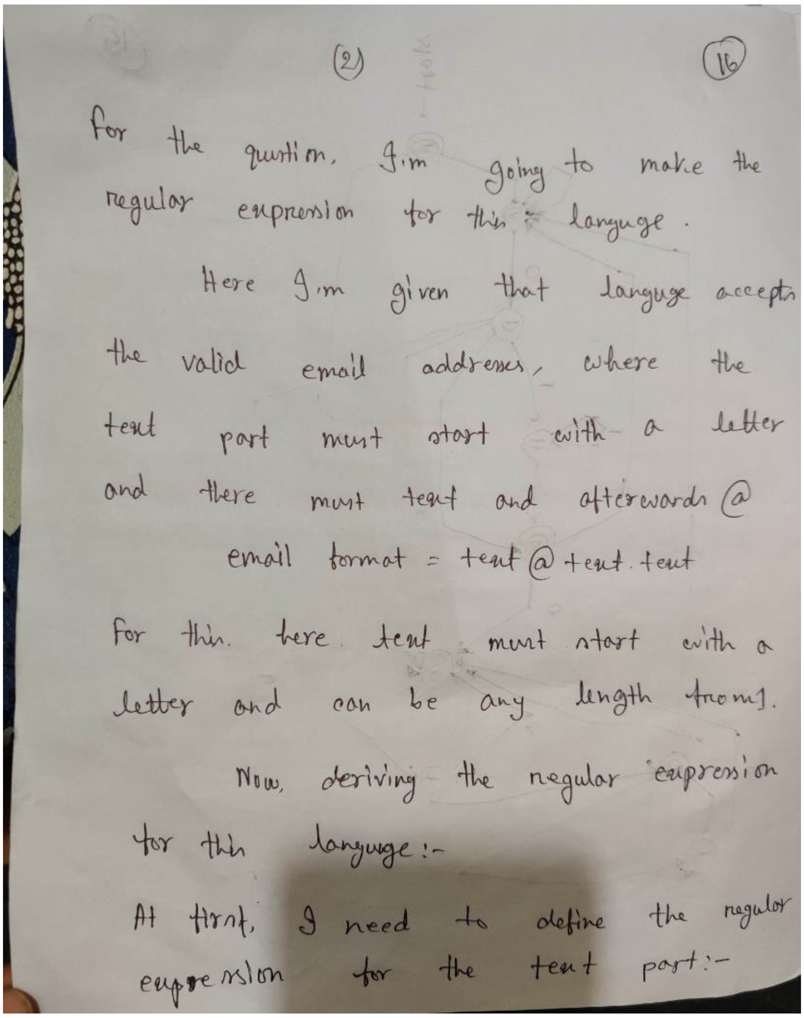
14

(1)

At first lets,

NFA state	DFA state	Type		0	1	a	b
{0,1,2,3,4,7,11,13,21}	A	accept	B			C	
{1,2,3,4,7,8,11,12,13,14,20,44}	B	accept	B	D		C	E
{25}	C		F				
{15,16,18}	D		G		H		
{3,4,7,9,10,11}	E		I			C	
{3,4,6,10,11}	F		I			C	
{1,2,3,4,7,11,13,17,20,21}	G	Accept	B			C	
{16,17,18}	H		G		H		
{1,2,3,4,~~7~~,8,11,12,13,20 21}	I	Accept	B			C	E





(2) (16)

for the qustion, I.m going to make the regular expression for this languge.

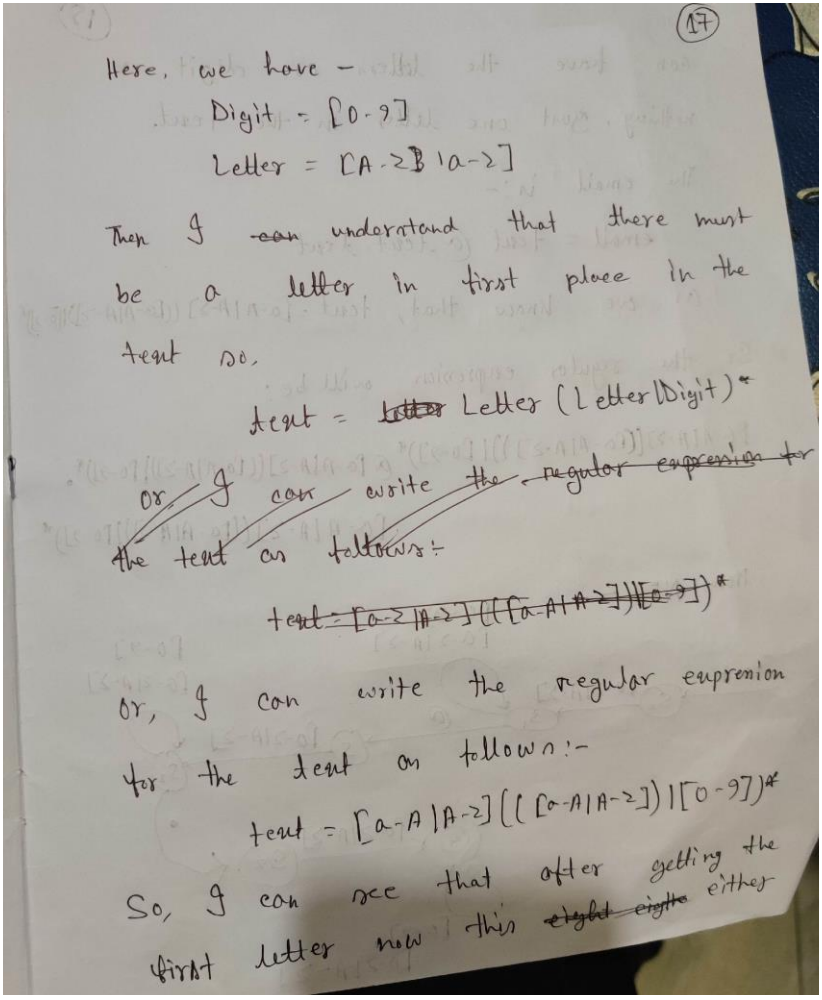
Here I.m given that languge accepts the valid email addresses, where the text part must start with a letter and there must text and afterwards @

email format = text @ text.text

For this. here text must start with a letter and can be any length from 1.

Now, deriving the regular expression for this languge:-

At first, I need to define the regular expression for the text part:-



Here, we have –

Digit = [0-9]

Letter = [A-Z | a-z]

Then I ~~can~~ understand that there must be a letter in first place in the text so,

text = ~~Letter~~ Letter (Letter|Digit)*

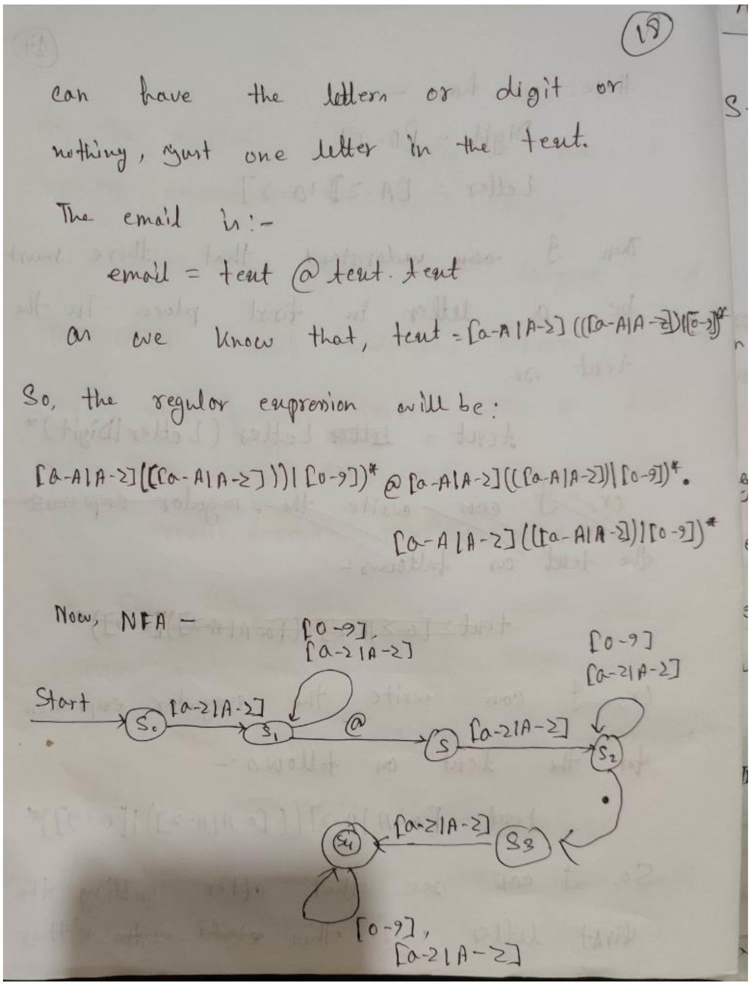
~~Or, I can write the regular expression for the text as follows:–~~

~~text = [a-z |A-Z] (([a-A | A-Z]) | [0-9])*~~

Or, I can write the regular expression for the text as follows:–

text = [a-A |A-Z] (([a-A|A-Z]) | [0-9])*

So, I can see that after getting the ~~eight eigths~~ either first letter now this



can have the letters or digit or nothing, just one letter in the text.

The email is:-

email = text @ text. text

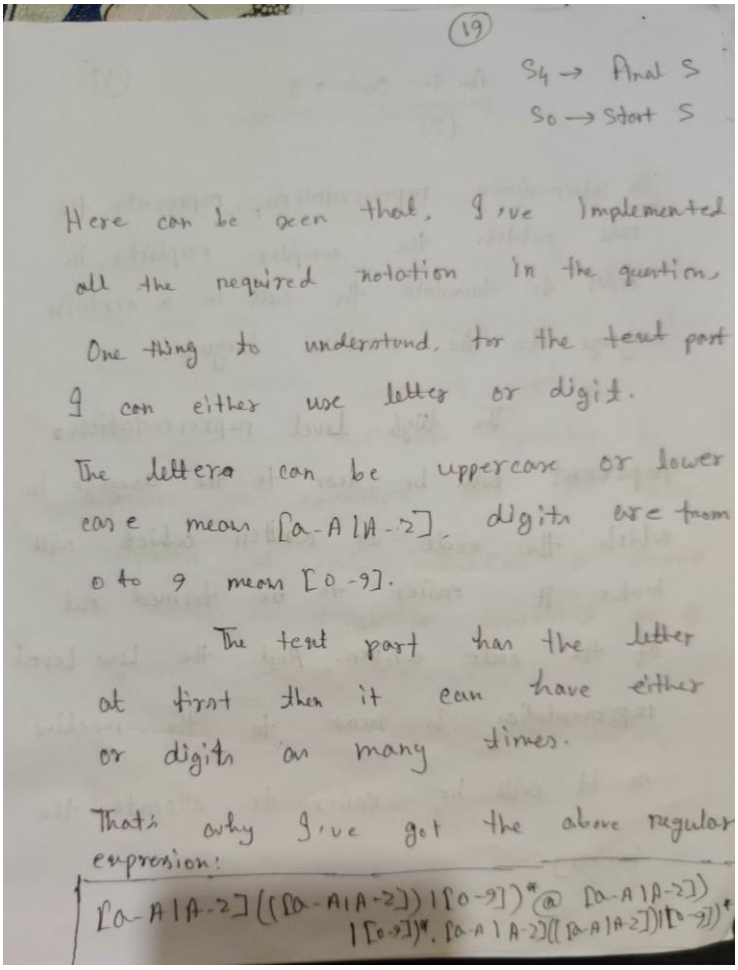
as we know that, text = [a-A|A-Z](([a-A|A-Z])|[0-9])*

So, the regulor expression will be:

[a-A|A-Z]([[a-A|A-Z]))|[0-9])* @ [a-A|A-Z](([a-A|A-Z])|[0-9])*. [a-A|A-Z](([a-A|A-Z])|[0-9])*

Now, NFA -

[0-9],
[a-z|A-Z]
[0-9]
[a-z|A-Z]
Start
S₀
[a-z|A-Z]
S₁
@
S
[a-z|A-Z]
S₂
•
S3
[a-z|A-Z]
S₄
[0-9],
[a-z|A-Z]



(19)

S4 → final S

S0 → Start S

Here can be seen that, I've implemented all the required notation in the question. One thing to understand, for the text part I can either use letter or digit. The letters can be uppercase or lower case mean [a-A|A-2]. digits are from 0 to 9 mean [0-9].

The text part has the letter at first then it can have either or digits as many times.

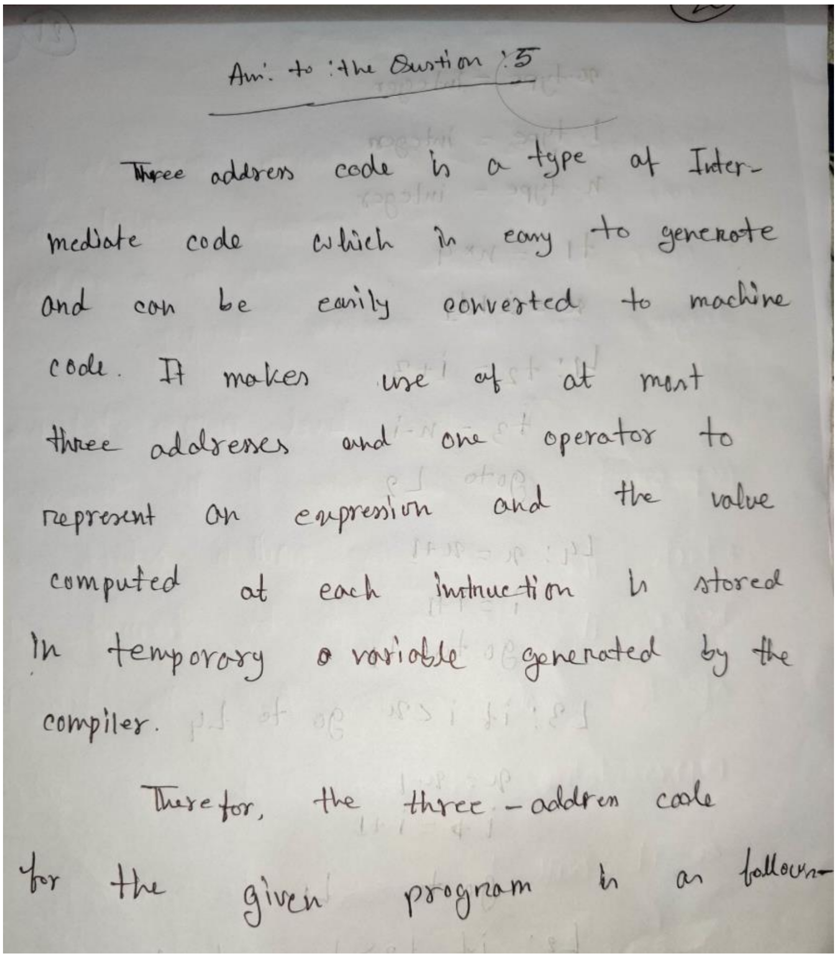
That's why I've got the above regular expression:

[a-A|A-2](([a-A|A-2])|[0-9])*@ [a-A|A-2])
|[0-9])*. [a-A|A-2](([a-A|A-2])|[0-9])*

【Q5】 (15 scores) Given the following C program:

```
int x, i, n;
x=i+n*4;
while (i+3 > n-i)
{
    if (i<x) { x =x+1; }
    else
    {
        x=x-1;
    };
    i++;
}
```

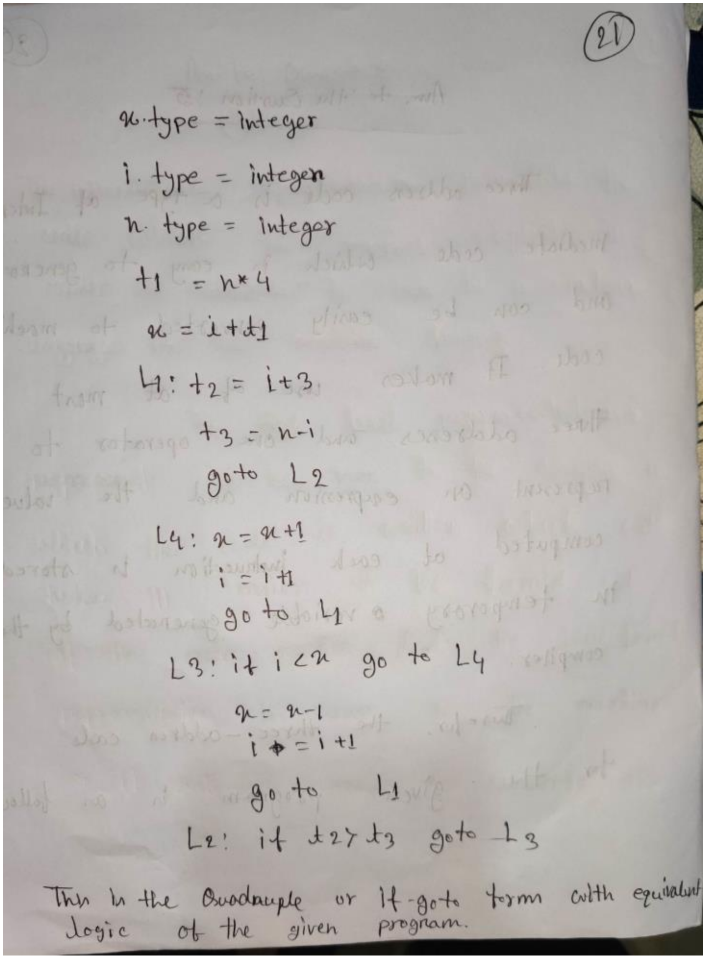
Please present the Quadruple (three-address code) or if-goto forms with equivalent logic to above program. (15 scores)



Ans. to :the Qustion :5

Three address code is a type of Intermediate code which is easy to generate and can be easily converted to machine code. It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary a variable generated by the compiler.

Therefor, the three-address code for the given program is as follows-



21

```
x.type = integer
i. type = integen
n. type = integor
t1 = n*4
x = i+t1
L1: t2 = i+3
    t3 = n-i
    goto L2
L4: x = x+1
    i = i +1
    go to L1
L3: if i<x go to L4
    x = x-1
    i = i +1
    go to L1
L2: if t2>t3 goto L3
```

This is the Quadruple or If-goto form with equivalent logic of the given program.

【Q6】 (15 scores)

Given G[S] as following:

S → A=E

A → H id | id

H→ * | ε

E → id + E | A

(1) Present the First set for each production rule. (3 scores)

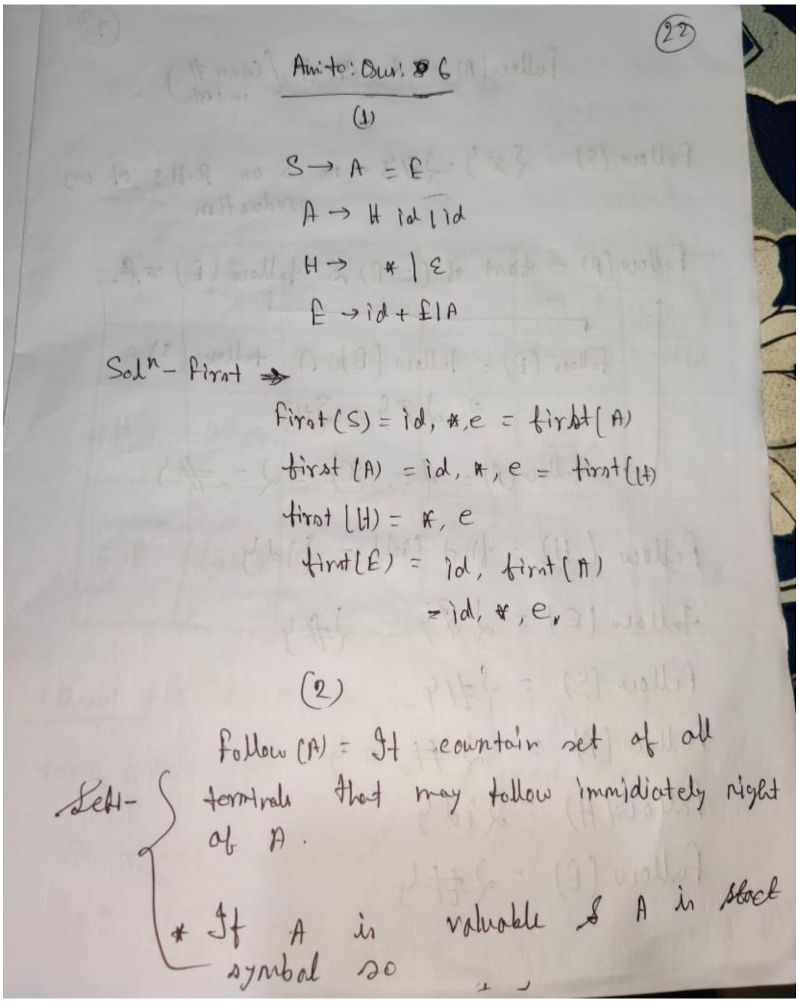
(2) Present the Follow Set for each non-terminal symbol. (3 scores)

	First	Follow
S → A=E		
A → H id		
A → id		
H → ε		
H → *		
E → id + E		
E → A		

(3) Present the LL(1) table. Is this grammar LL(1)? (5 +2 scores)

	Id	=	+	*	#
S					
A					
H					
E					

(4) Given input string “a=b+*c”, present the derivation sequences according to the LL(1) table: (2 scores)



22

Ans to: Qus: 6

(1)

$S \rightarrow A = E$

$A \rightarrow H\ id \mid id$

$H \rightarrow * \mid \varepsilon$

$E \rightarrow id + E \mid A$

Soln – first ⇒

first(S) = id, *, e = first(A)

first(A) = id, *, e = first(H)

first(H) = *, e

first(E) = id, first(A)

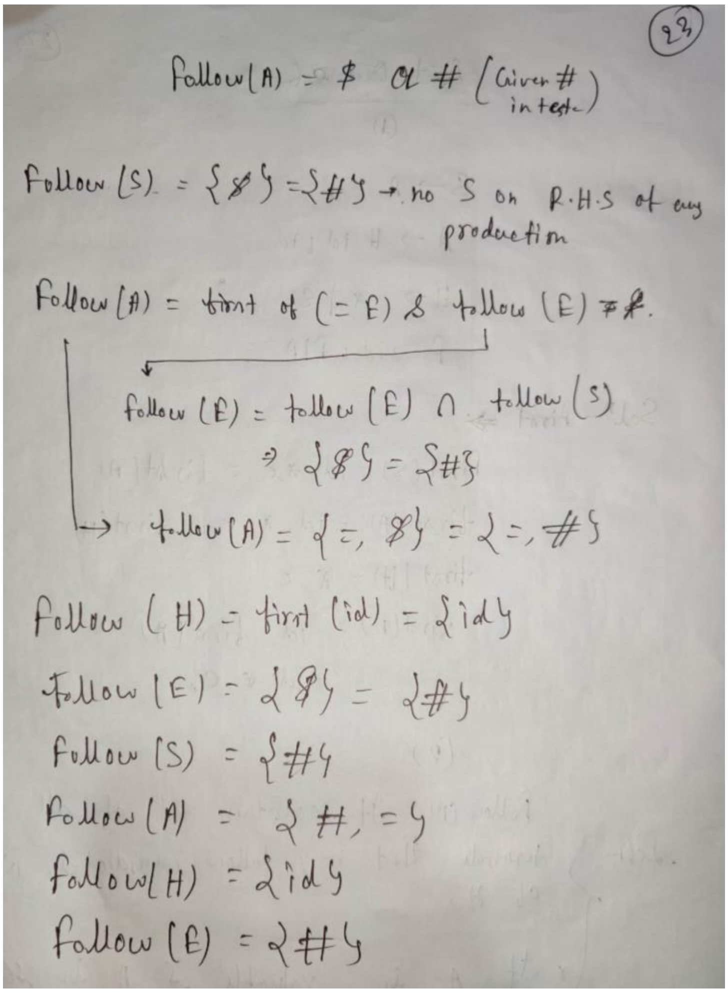
= id, *, e,

(2)

follow(A) = It countain set of all terminals that may follow immidiately right of A.

Left-

* If A is valuable & A is start symbol so



(23)

follow(A) = $ or # (Given # in test)

Follow (S) = {$} = {#} → no S on R.H.S of any production

Follow (A) = first of (= E) & follow (E) ≠ $.

follow (E) = follow (E) ∩ follow (S)

⇒ {$} = {#}

→ follow (A) = {=, $} = {=, #}

follow (H) = first (id) = {id}

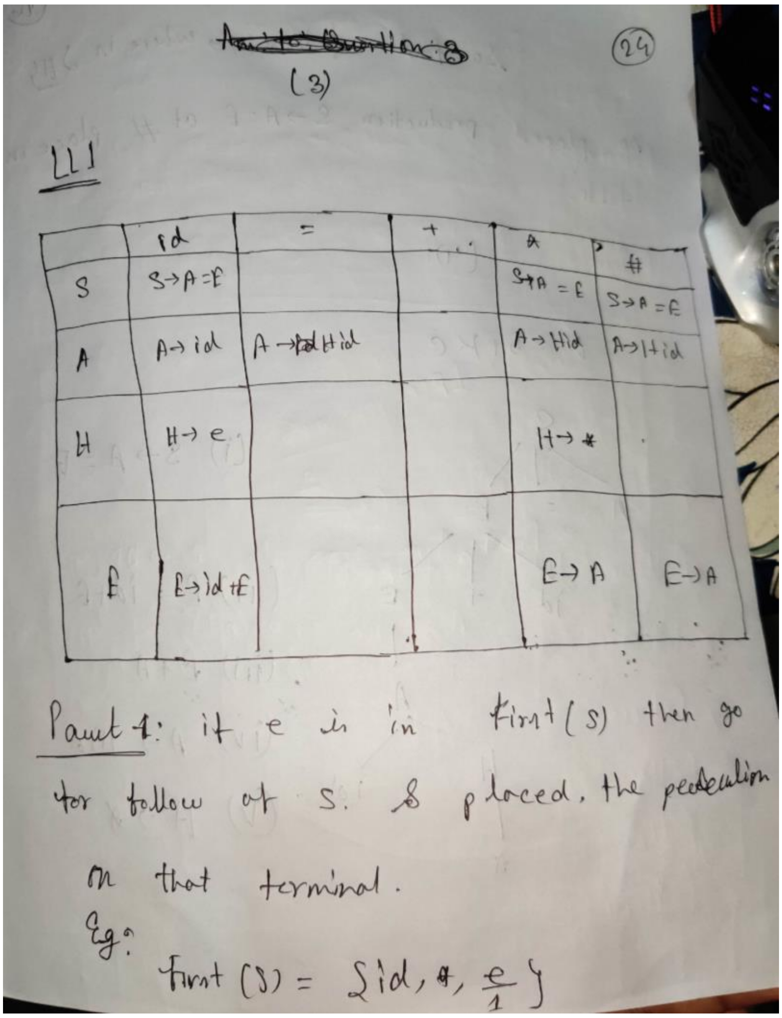
Follow (E) = {$} = {#}

follow (S) = {#}

Follow (A) = {#, =}

follow (H) = {id}

follow (E) = {#}



~~Ans to Question 3~~ 24

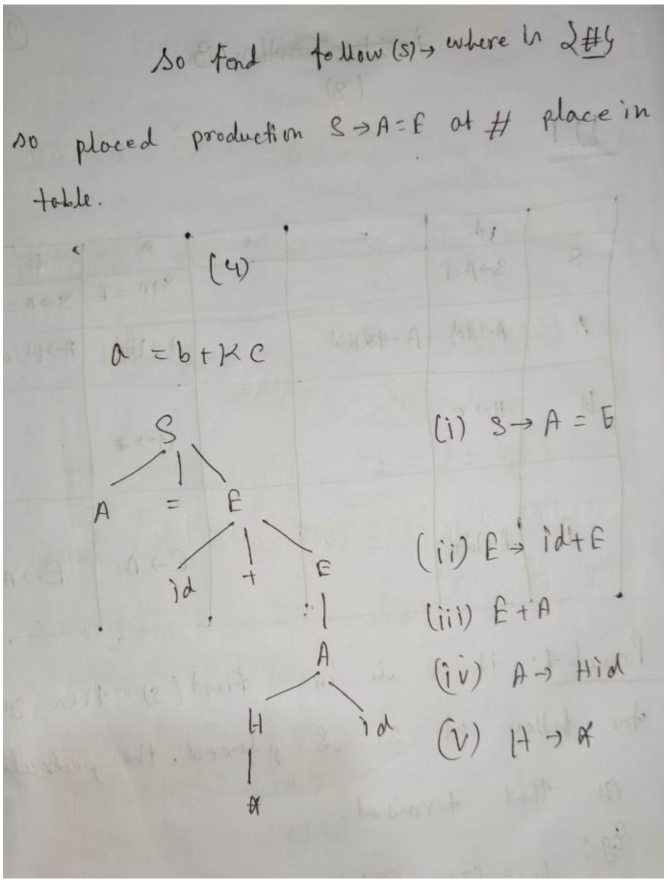
(3)

LL1

	id	=	+	*	#
S	S→A=E			S→A = E	S→A = E
A	A→ id	A → ~~id~~ H id		A → Hid	A→H id
H	H→ e			H→ *	
E	E→id+E			E→ A	E→A

Part 1: if e is in First(S) then go for follow of S. & placed, the production on that terminal.

Eg: First (S) = {id, *, $\underset{1}{e}$}



So find follow(S) → where is {#}

so placed production $S \to A = E$ at # place in table.

(4)

$a = b + Kc$

(i) $S \to A = E$

(ii) $E \to id + E$

(iii) $E + A$

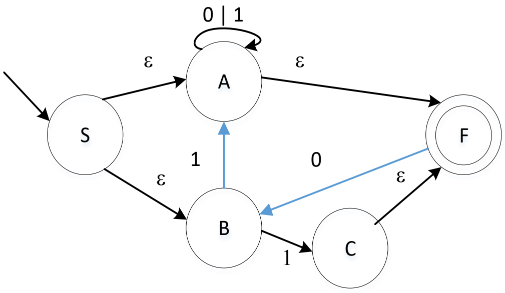
(iv) $A \to H\,id$

(v) $H \to \alpha$

S
A
=
E
id
+
E
A
H
id
α

【Q7】 (10 scores)

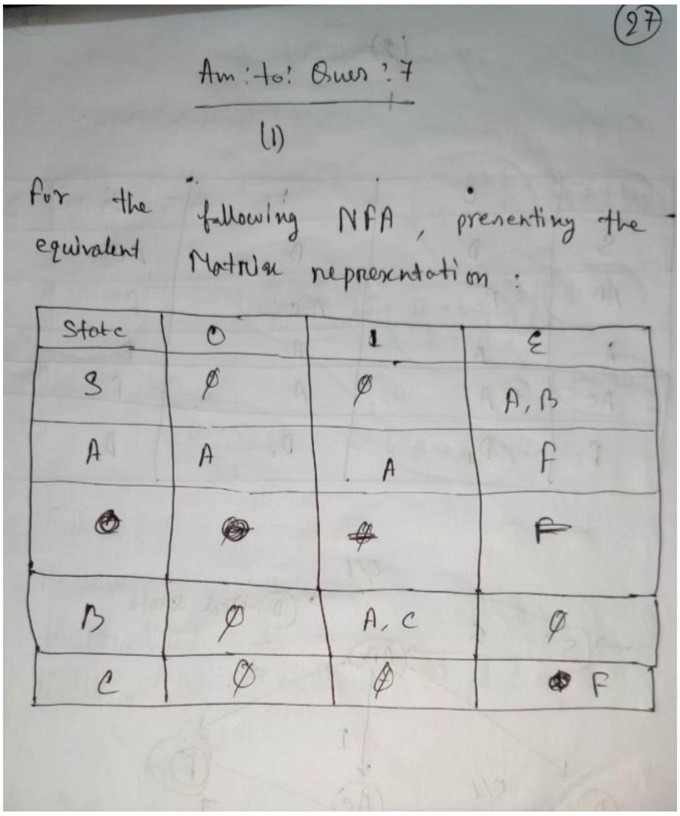
Given the following NFA:



(1) Present the equivalent Matrix representation of this NFA. (8 scores)

	0	1	ε
S			
A			
B			
C			

(2) Transform the NFA to DFA. (7 scores)



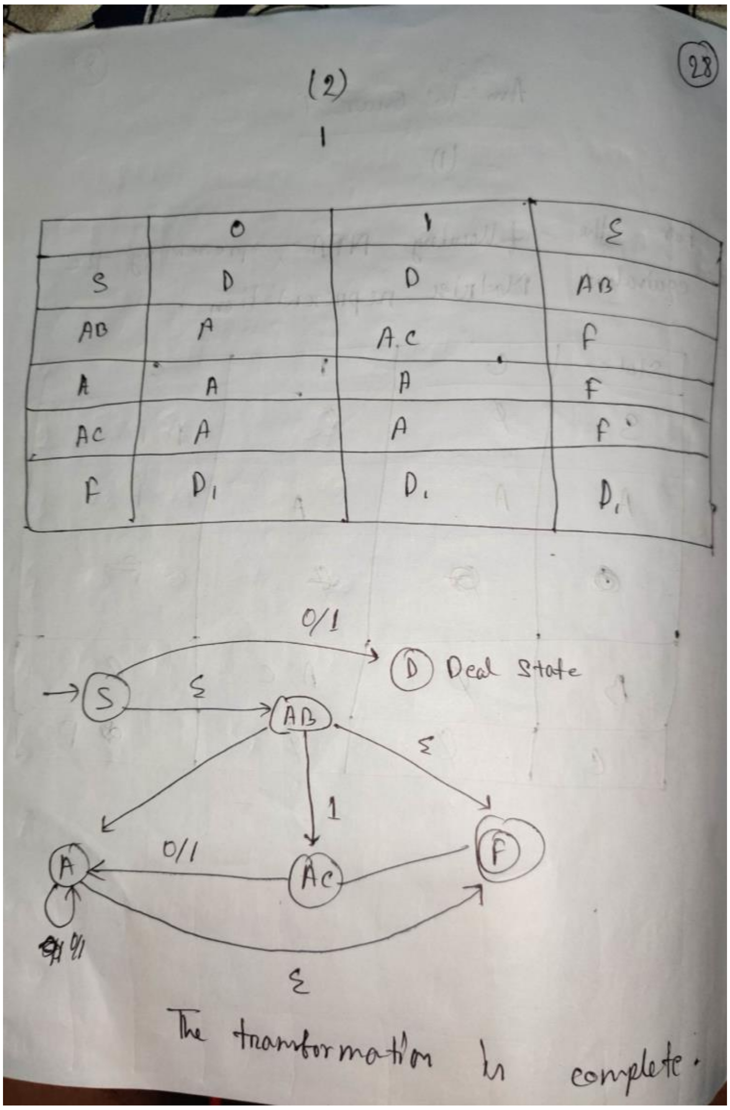
27

Ans: to: Ques: 7

(1)

For the following NFA, presenting the equivalent Matrix representation :

State	0	1	ε
S	∅	∅	A, B
A	A	A	F
		~~∅~~	~~F~~
B	∅	A, C	∅
C	∅	∅	F



(28)

(2)

1

	0	1	ε
S	D	D	AB
AB	A	A.C	F
A	A	A	F
AC	A	A	F
F	D_1	D_1	D_1

0/1
D
Deal State
S
ε
AB
ε
1
0/1
A
AC
F
0/1
ε

The transformation is complete.

【Q8】 (15 scores)

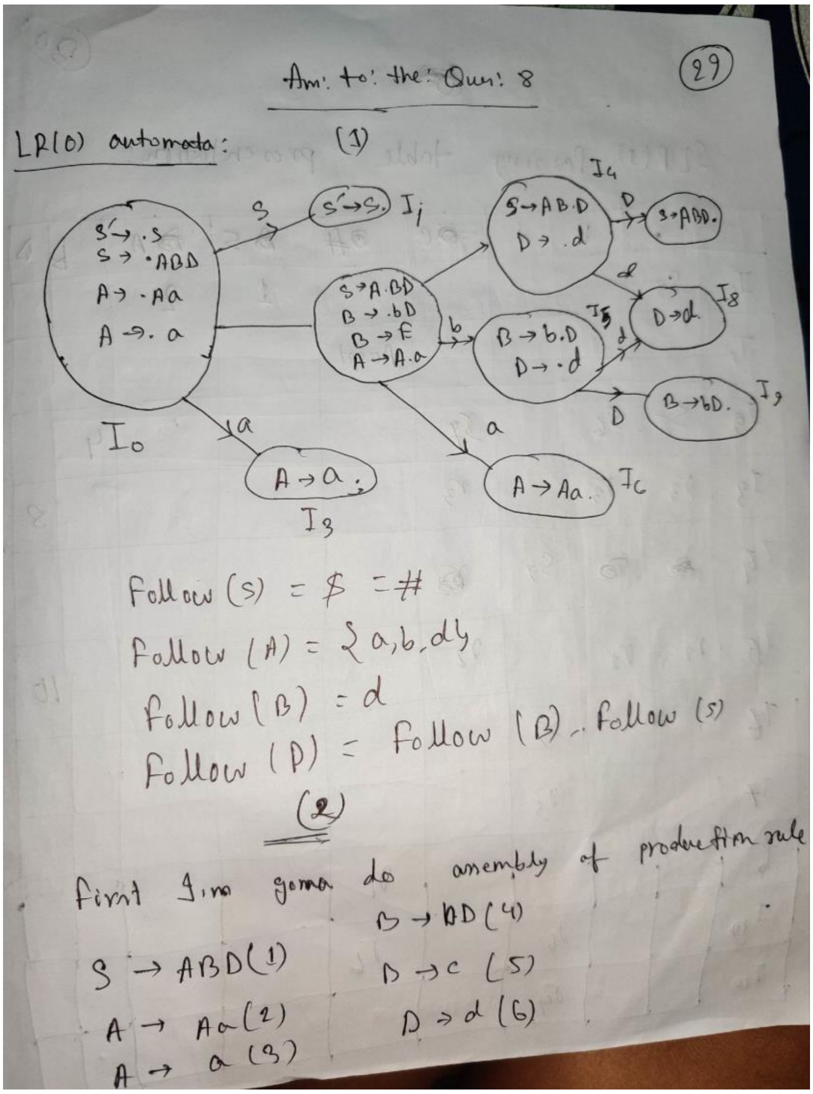
Given the following grammar G[S]:

(1) S→ABD
(2) A→Aa | a
(3) B→bD | ε
(4) D→d

(1) Please write down the LR(0) automata. (6 scores)

(2) Please present the SLR(1) parsing table. (7 scores)

	a	b	d	#	S	A	B	D
I0								
I1								
…	…	…	…	…	…	…		



Ans: to: the: Qus: 8 (29)

LR(0) automata: (1)

S′→.S
S→ •ABD
A→ ·Aa
A→. a
I0
S
S′→S.
I1
S→A.BD
B→ .bD
B→E
A→A.a
S→AB.D
D→ .d
I4
D
S→ABD.
d
D→d.
I8
b
B→b.D
D→·d
I5
d
D
B→bD.
I9
a
A→a.
I3
a
A→Aa.
I6

Follow (S) = \$ = #

Follow (A) = {a,b,d}

follow (B) = d

Follow (D) = Follow (B). follow (S)

(2)

first I,m gonna do assembly of production rule

S → ABD(1)

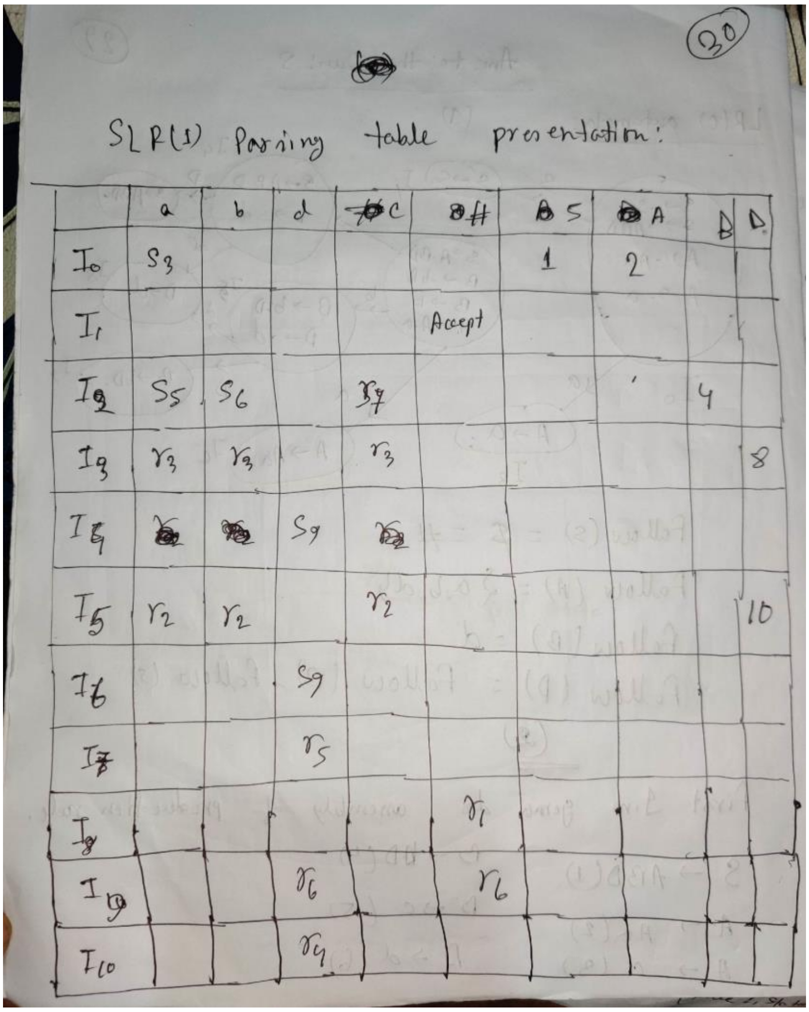
A → Aa(2)

A → a (3)

B → bD (4)

D → c (5)

D → d (6)



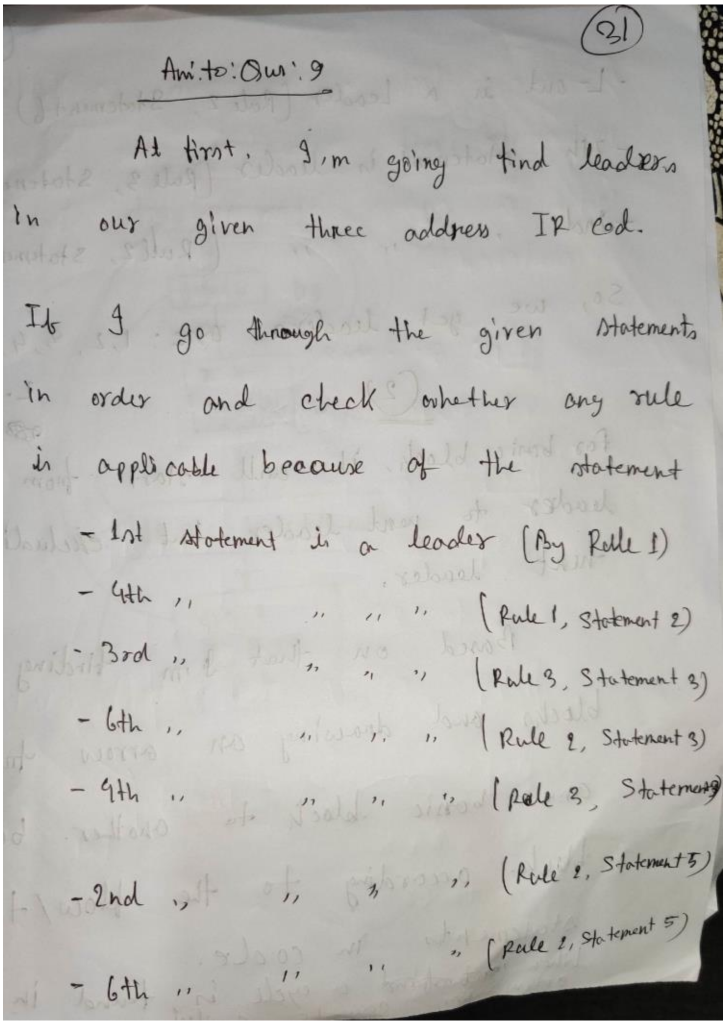
30

SLR(1) Parsing table presentation:

	a	b	d	c	#	S	A	B	D
I_0	S_3					1	2		
I_1					Accept				
I_2	S_5	S_6		r_7				4	
I_3	r_3	r_3		r_3					8
I_4			S_9						
I_5	r_2	r_2		r_2					10
I_6			S_9						
I_7			r_5						
I_8					r_1				
I_9			r_6		r_6				
I_{10}			r_4						

【Q9】For the following three addresses IR code, write down the basic blocks and draw the control flow graph of it. (10 scores).

```
(1) X := X+2
(2) if X>=10 goto (4)
(3) goto (6)
(4) X := X-1
(5) goto (2)
(6) if A<>4 goto (L_out) //L_out is outside of this code sequence
(7) X := X-2
(8) Y := X+5
(9) goto (2)
```



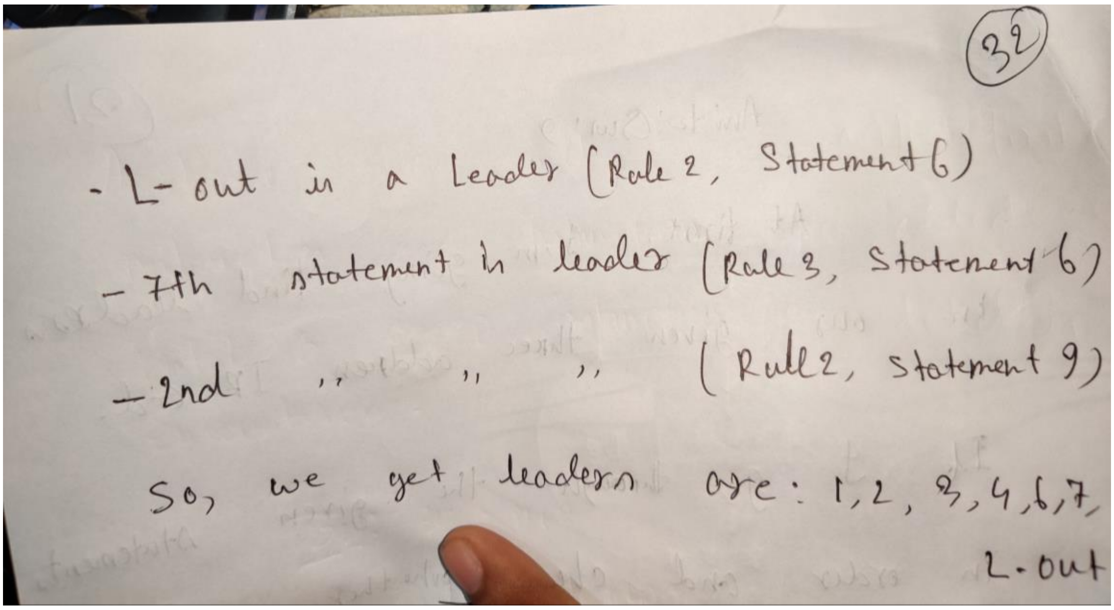
(31)

Ans.to:Qus:9

At first, I,m going find leaders in our given three address IR cod.

If I go through the given statements in order and check whether any rule is applicable because of the statement

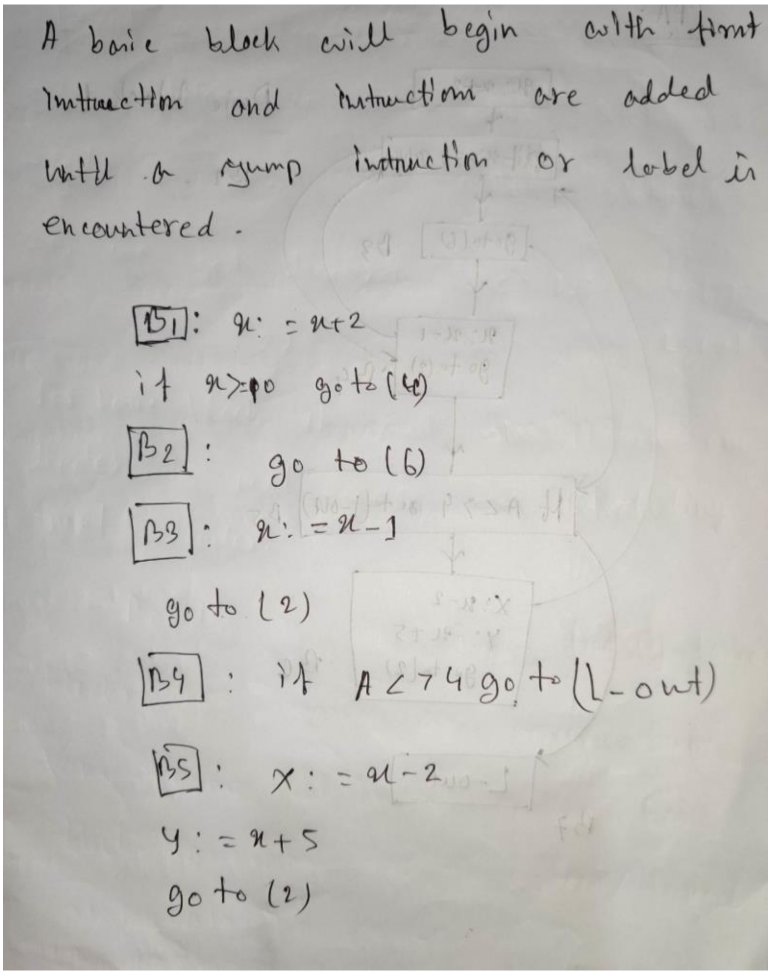
- 1st statement is a leader (By Rule 1)
- 4th ,, ,, ,, ,, (Rule 1, Statement 2)
- 3rd ,, ,, ,, ,, (Rule 3, Statement 3)
- 6th ,, ,, ,, ,, (Rule 2, Statement 3)
- 9th ,, ,, ,, ,, (Rule 3, Statement3)
- 2nd ,, ,, ,, ,, (Rule 2, Statement 5)
- 6th ,, ,, ,, ,, (Rule 2, Statement 5)



(32)

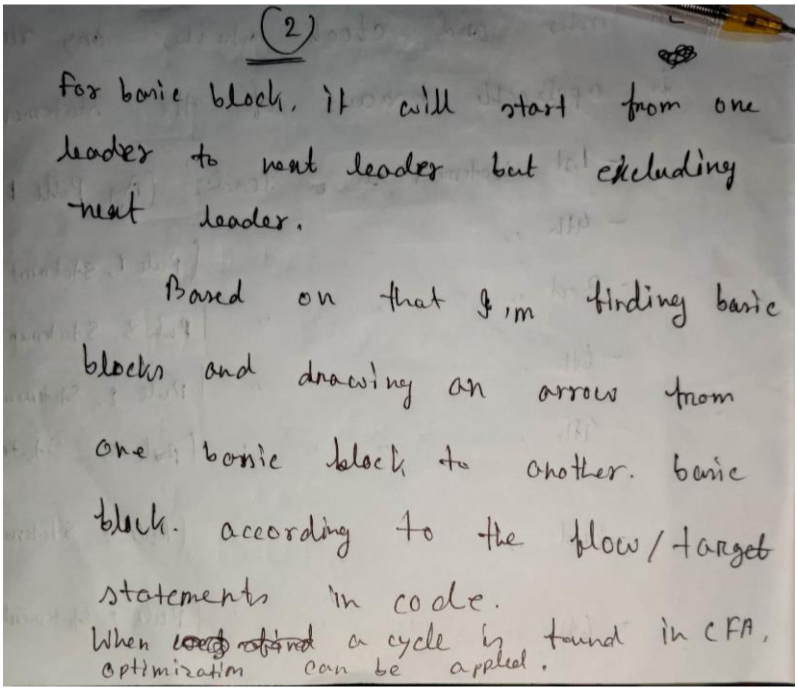
- L-out is a Leader (Rule 2, Statement 6)
- 7th statement is leader (Rule 3, Statement 6)
- 2nd ,, ,, ,, (Rule 2, statement 9)

So, we get leaders are: 1, 2, 3, 4, 6, 7, L-out



A basic block will begin with first instruction and instruction are added until a jump instruction or label is encountered.

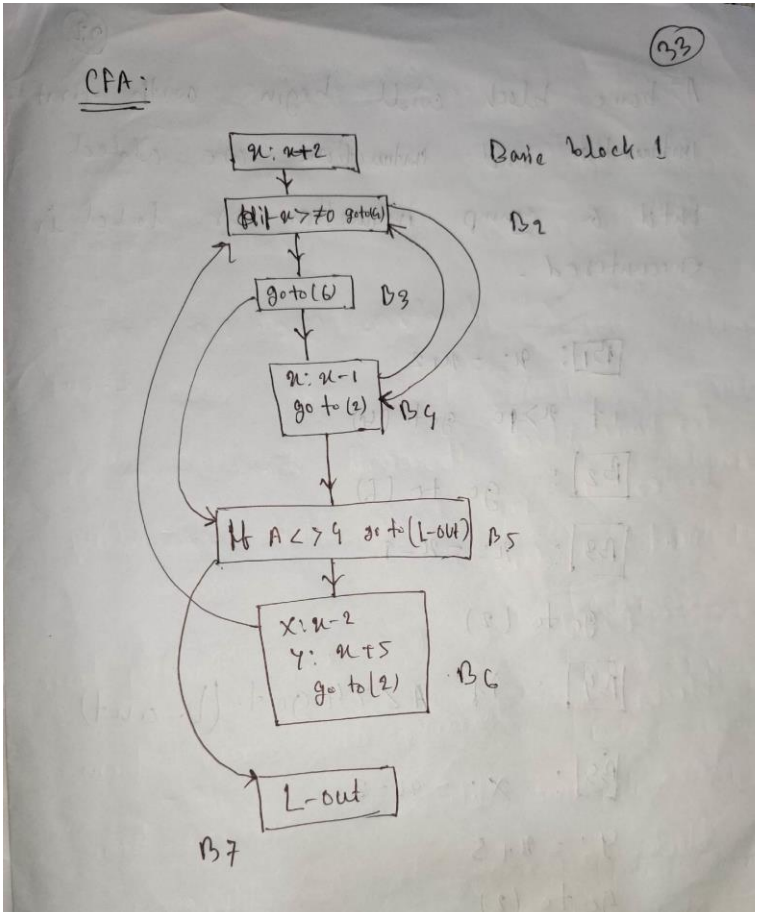
```
B1: x: = x+2
if x>=10 go to (4)
B2:  go to (6)
B3: x: = x-1
go to (2)
B4 : if A<>4 go to (L-out)
B5: x: = x-2
y: = x+5
go to (2)
```



(2)

for basic block, it will start from one leader to next leader but excluding next leader.

Based on that I'm finding basic blocks and drawing an arrow from one basic block to another. basic block. according to the flow/target statements in code.
When [illegible] a cycle is found in CFA, optimization can be applied.


33
CFA:
x: x+2
Basic block 1
if x>=0 goto(6)
B2
goto(6)
B3
x: x-1
go to (2)
B4
If A<>4 go to (L-out)
B5
X: x-2
Y: x+5
go to (2)
B6
L-out
B7