Rajiv Gandhi Proudyogiki Vishwavidyalaya

(University of Technology of Madhya Pradesh)
Airport Road, Gandhi Nagar, Bhopal-462033

END SEM EXAMINATION – DECEMBER-2020

(For Offline Open Book Examination only)

Course : B.E./B.Tech./B.Pharm/BE-PTDC/B.Arch./MCA/M.E./M.Tech/M.Pharm./M.Arch. / M.Tech(PT)/MCA DD/MBA INT./Pharm-D

Total No. of pages used: [ ]

Subject Code: [I][T][5][0][3][C][1][J]

Branch : Information Technology Subject Title: Theory of Computation

Enrolment No. : [0][1][0][3][I][T][1][8][1][0][4][4]

Date of Examination : 8/02/2021 Time of Examination : 10:00AM - 1:00PM

Signature of the Candidate (Must be as per Institute Record): Shweera

Date of Birth (Candidate need to fill) : [2][4][1][0][2][0][0][0]

Identification proof (Tick any one and write number)

PAN Card / Voter ID / Aadhar Card Number [4][9][6][6][9][2][5][5][9][8][9][8]

IMPORTANT INSTRUCTIONS :

1. Candidate must start writing from this page.
2. Any part of page / pages should not be left blank. This is necessary to reduce the size of the pdf file. Before uploading the pdf file please check and ensure that written text is readable. If you are not able to upload answer book, kindly submit original answer book at nearest collection centre only.
3. There should not be any cutting / overwriting in the enrollment number.
4. Answer must be precise and to the point.
5. Students should take the printout of this page much before the start of the exam and complete the entries.
6. The candidate must enter the total no. of pages used on the top of this cover page. Cover page is page no. 1.
7. Please read the Instruction for exam December-2020, which is already uploaded in the website www.rgpv.ac.in, and Students Portal.
8. Each page must be numbered and candidates must sign on the bottom of each page.
9. **Only one side writing is allowed on A4 size answer sheets and don't Scan blank pages.**

Start Writing from here :

1. (a) A grammar is a set of production rules which are used to generate strings of a language
It is defined by four tuples $G = \{V, T, P, S\}$ where,

V = non-terminal
T = Terminals
P = Production rule
S = Start Symbol

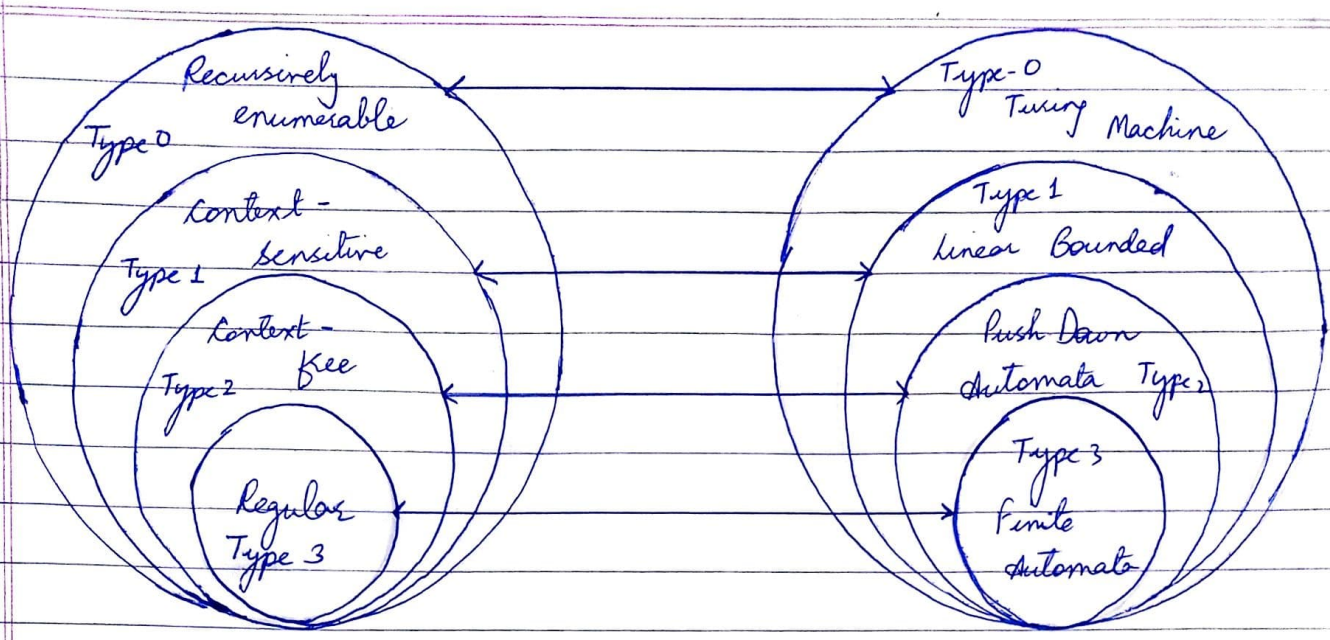
A production rule has the form $\alpha \rightarrow \beta$, where α and β are strings on $V_n \cup \Sigma$ and least one symbol of α belongs to V_n.

According to Chomsky, there are four types of grammar - Type 0, Type 1, Type 2 and Type 3.

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable	Turing machin
Type 1	Context-Sensitive	Context-sensitive language	Linear bounde
Type 2	Context-free	Context-free language	Pushdown
Type 3	Regular grammar	Regular language	Finite state

Venn Diagram of Grammar Types :- P.T.O

Sign: Shweera



(i) Type-3 Grammar

→ Type-3 grammars generates regular languages.

→ Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single regular terminal or single terminal followed by a single non-terminal.

→ Regular language is accepted by Finite state automaton

→ The productions must be in the form :-

$X \rightarrow a$ or $X \rightarrow aY$, where $X, Y \in N$ (or V)

(Non-termina

and $a \in T$ (Terminal)

The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule.

Sign: Shreera

Example :-

$X \rightarrow \varepsilon$

$X \rightarrow a / aY$

$Y \rightarrow b$

Regular languages are used to define search patterns of programming languages.

(ii) Type - 2 Grammar

→ Type - 2 grammars generate context - free languages. The productions must be in the form $A \rightarrow \gamma$, where $A \in V$ and $\gamma \in (T \cup N)^*$

→ These languages generated by these grammars can be recognized by a non - deterministic push down automaton.

→ Context free languages are theoretical basis of for the syntax of most of the programming languages.

Example :-

$S \rightarrow Xa$

$X \rightarrow a / aX / abc / \varepsilon$

(iii) Type - 1 Grammar

→ Type - 1 grammars generate context - sensitive languages. The productions must be in the form

$\alpha A \beta \rightarrow \alpha \gamma \beta$

where, $A \in V$ and $\alpha \beta \gamma \in (TUV)^*$

The strings α and β may be empty but γ must not be empty.

→ The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right-side of any rule.

→ The languages generated by these grammars are recognized by a linear bounded automaton.

→ Examples:- $AB \rightarrow AbBc$, $A \rightarrow bcA$, $B \rightarrow b$

(iv) Type-0 Grammars

→ Type-0 grammars generate recursively enumerable languages. The productions have no restrictions.

→ These languages are recognized by a Turing machine.

→ The productions can be in the form $\alpha \rightarrow \beta$, where α is a string of terminals and non-terminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.

→ Type-0 grammars are too general to describe the syntax of programming languages and

Sign: Shreesa

natural languages.

→ Examples :- $AB \rightarrow A$, $AB \rightarrow aB$, etc.

(b) A language L on Σ is said to be recursive, if there exists a turing machine M that accepts L and that halts on every $w \in \Sigma^+$. In other words, a language is recursive iff there exists a membership algorithm for it or if there exists a Turing Machine that recognizes L.

Recursive languages are closed under complementation.

Theorem :- If L is recursive, then $\bar{L}$ will also be recursive.

Proof :- Let 'L' be a recursive language and 'M' be a turing machine that halts on all inputs and accepts L.

Let 'M'' be a turing machine constructed from 'M' such that if M enters a final state on input w, then M' halts without

Sign: Shreeya

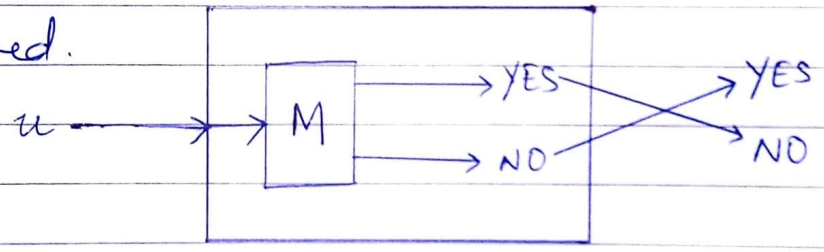
accepting.

If M halts without accepting then M' enters a final state. Clearly, one of these two event occurs ∴ M' is an algorithm.

L(M') is the complement of L and thus the complement of L is a recursive language.

∴ If L is recursive, then $\bar{L}$ will also be recursive.

Hence proved.



8.(a) Top Down Passing

Passing is used to derive a string using the production rules of the grammar. It is used to check the acceptability of a string.

Passing is divided into two types :
Top-Down passing and Bottom-up passing

Sign: Shreera

When the passer starts constructing the parse tree from the start symbol and then tries to transform the start symbol to input, it is called top-down parsing

Top-down passing attempts to find the left most derivations for an input string and so this technique uses Left Most Derivation.

Its main decision is to select what production rule to use in order to construct the string.

Recursive-descent passing : It is a common form of top-down passing. It is called recursive as it uses recursive procedures to process the input. It suffers from backtracking.

Limitations of top-down passing -

(i) Backtracking : It is a method of expanding non-terminal symbol where one alternative could be selected until any mismatch occurs otherwise another alternative is checked.

(ii) Left recursion : Top down ~~re~~ passer could enter

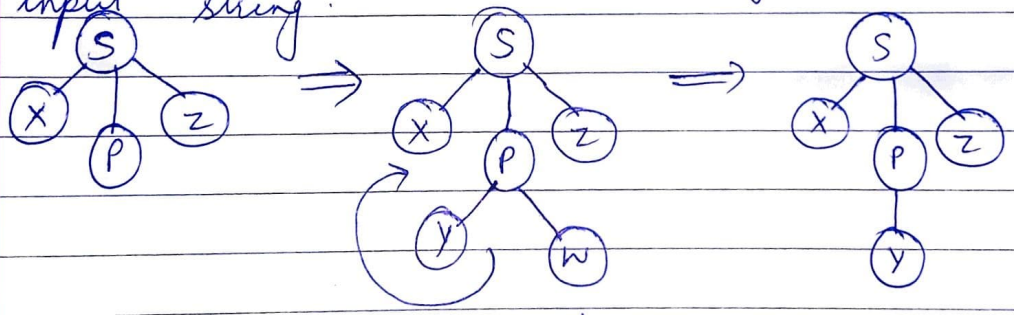
Sign : Skveera

an infinite loop.

(iii) Left factoring: It is used when the suitability of the two alternatives is checked while expanding the non-terminals.

(iv) Ambiguity: Ambiguous grammar creates more than one parse tree of the single string which is not acceptable in top-down parsing and need to be eliminated.

Example:- Let G be the grammar -
$S \rightarrow XPZ$, $P \rightarrow yw/y$ & 'xyz' be the input string.



Backtracking

(b) Halting problem

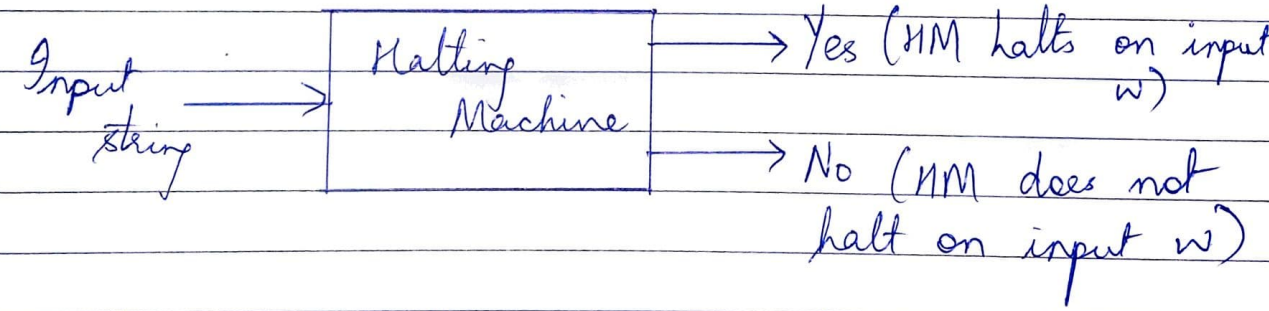
The halting problem is the problem of finding out whether, with the given input, a program will halt at some time or

Sign: Shreera

continue to run indefinitely.

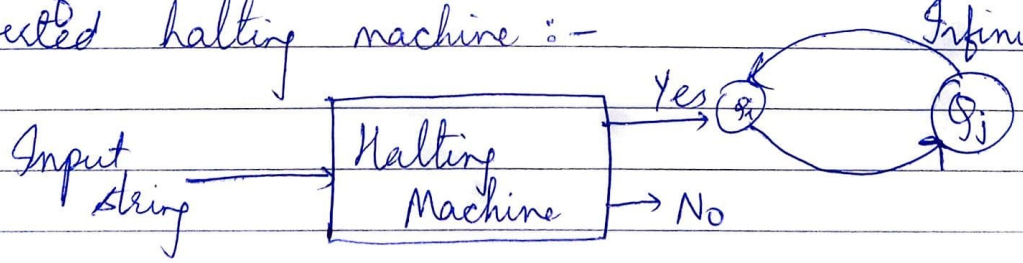
The halting problem of a turing machine is undecidable and it can be proved by the theorem of contradiction.

Let HM be the turing machine that produces a 'yes' or 'no' in a finite amount of time. when given an input string w
If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'.



Let (HM)' be the inverted halting machine as:
→ If H returns YES, then loop forever.
→ If H returns NO, then halt.

Inverted halting machine :-



Sign: Ekveera

Further, a machine (HM)$_2$ which input itself is constructed as follows :-
→ If (HM)$_2$ halts on input, loop forever
→ Else, halt
This is the contradiction, hence the halting problem is undecidable.

Example :-
Consider the following python program :-

```
x = input()
while x:
    pass
```

It reads the input and if the input is not empty, the program will loop forever. Thus, if the input is empty, the program will terminate.

(c) Greibach Normal Form

A context free grammar is in GNF if all the production rules satisfy one of the following conditions :-

(i) A start symbol generating ε. Eg :- $S \rightarrow \varepsilon$
(ii) A non-terminal getting a terminal. Eg :- $A \rightarrow a$
(iii) A non-terminal generating a terminal which is

Sign: Ekveera

followed by any number of non-terminals.
Eg :- $S \rightarrow aASB$

For example :-

$G_1 = \{ S \rightarrow aAB \mid aB,\ A \rightarrow aA \mid a,\ B \rightarrow bB \mid b \}$
$G_2 = \{ S \rightarrow aAB \mid aB,\ A \rightarrow aA \mid \varepsilon,\ B \rightarrow bB \mid \varepsilon \}$

The production rules of grammar G1 satisfy the rules specified by GNF norms so G1 is in GNF.
The production rules of grammar G2 does not satisfy the rules as $A \rightarrow \varepsilon$ & $B \rightarrow \varepsilon$ contains ε. ∴ G2 is not in GNF.

Steps for conversion of CFG into GNF :-

(i) Convert the grammar into Chomsky Normal Form (CNF).

(ii) Change the names of the non-terminal symbols into some A_i in ascending order of i.

(iii) Alter the rules so that non-terminals are in ascending order, such that if the production is of the form $A_i \rightarrow A_j \gamma$, then

Sign : Shreeee

$i < j$ & should never be $i \geq j$.

(iv) Remove left recursion by introducing a new variable.

(d) Cellular Automata

A cellular automata (CA) is a collection of cells arranged in a grid, such that each cell changes state and a function of time ~~defined~~ according to a defined set of rules that includes the state of neighbouring cells.

A cellular automaton is a model of system of cell objects with the following characteristics:-

→ The cell lie on a grid.
→ Each cell has a state. The number of state possibilities is typically finite. The simplest example has two possibilities 0 and 1.
→ Each cell has a neighborhood. This can be defined in any number of ways, but it is typically a list of adjacent cells.

Example :-

Sign: Shreera

Conway's game of life is an example of an outer totalistic cellular automaton with cell values 0 and 1.

3. (a) The finite state machine is used to recognize patterns.

→ Finite automata machine takes the symbol as input and changes its state accordingly. In the input, when a desired symbol is found then the transition occurs.

→ While transition, the automata can move either to the next state or stay in the same state.

→ FA has two states : accept state and reject state. When the input string is successfully processed and the automata reached its final state then it will accept

A finite automata consists of following :-

Q : finite set of states
Σ : finite set of input symbol
q_0 : initial state
F : Final state
δ : Transition function

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

Sign : Shreera

FA is characterized in two ways :-

(i) DFA (deterministic FA)

→ It refers to the uniqueness of computation. In DFA, the input character goes to one state only.

→ DFA doesn't accept NULL move that means DFA cannot change state without any input character.

DFA has five tuples $\{Q, \Sigma, q_0, F, \delta\}$

(ii) NDFA (Non-deterministic FA)

→ It is used to transit the any number of states for a particular input.

→ NDFA accepts the NULL move that means it can change state without reading symbols.

NDFA also has five tuples same as DFA but NDFA has a different transition function.

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

Limitations of Finite Automata :-

(i) The expected character of deterministic finite state machines can be not needed in some

Sign: Ekveera

areas like computer games.

(ii) It is not applicable for all domains.

(iii) The implementation of huge systems using FSM is hard for managing without any idea of design.

(iv) The orders of state conversions are inflexible.

(v) Head movement is only in one direction.

(vi) The only memory it has is, state to state.

Applications of Finite state machines :-

(i) FSMs are most recognized for being utilized in artificial intelligence.

(ii) Also used in execution of ~~navigation~~ navigating parsing text

(iii) Lexical Analyzer

(iv) Switching circuit diagram.

(v) Digital logic design (Mealy & Moore machines)

(vi) Softwares with finite states like vending machines, weighing machines, traffic lights, toll machines, etc.

(b) A language is said to be regular if it can be represented by using a finite automata or if a regular expression can be generated from it.

Sign: Shreera

If a language can be expressed by :-
- FA (Finite Automata) or
- TG (Transition graph) or
- RE then it can also be expressed by other two as well.

(i) Kleen's Theorem Part I

If a language can be accepted by an FA then it can also be accepted by a transition graph (TG) as well.

(ii) Kleen's Theorem Part II

If a language can be accepted by TG then it can be expressed by 'regular expression' as well.

(iii) Kleen's Theorem Part III

If a language can be expressed by a regular expression then it can be accepted by an FA as well.

Example :- Kleen's theorem part - I can be used to generate a FA for the given regular expression :- $(ab+a)^*$

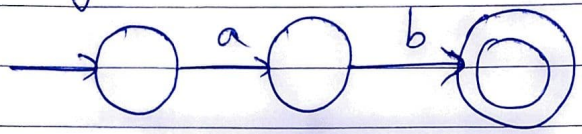
If r_1 and r_2 are two regular expressions

Sign: Ekveera

then :-

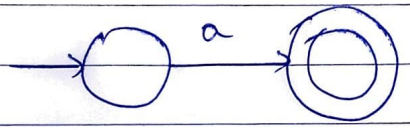
$r_1 + r_2$, $r_1 \cdot r_2$ and r_1^* are also regular expressions corresponding to the language $L(r_1) \cup L(r_2)$; $L(r_1) \cdot L(r_2)$ and $L(r_1)^*$ respectively.

(i)



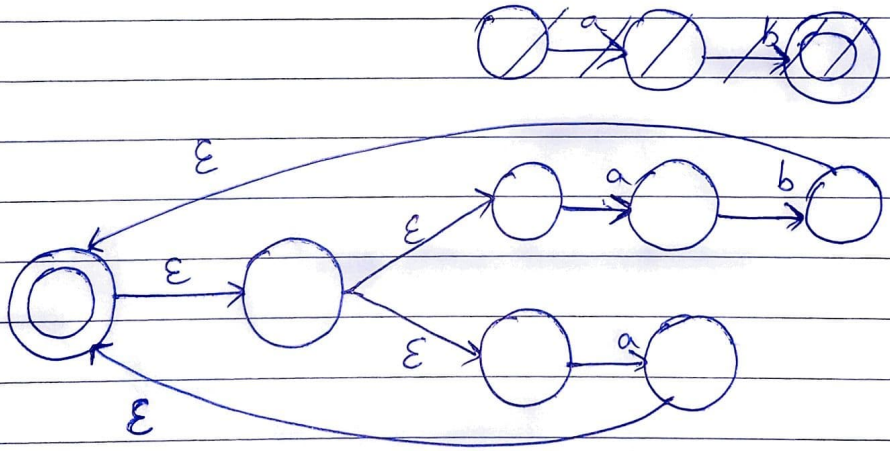
ab

(ii)



a

(iii)



$(ab + a)^*$

Sign : Shreesa

4.(a) A Turing machine is an accepting device which accepts the recursively enumerable languages generated by Type 0 grammars

A Turing machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape.

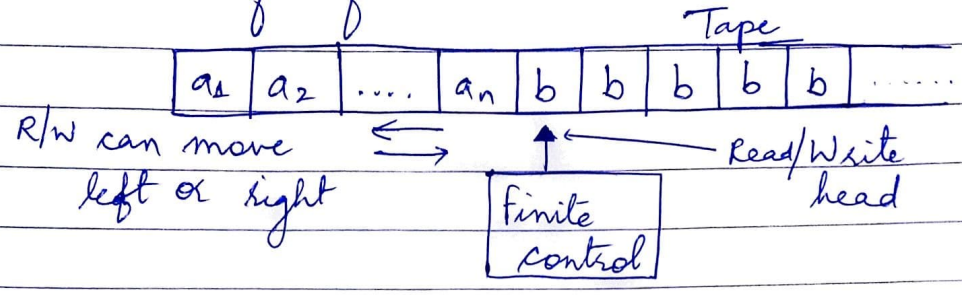
A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, its internal state is changed and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

A TM can be described as a 7-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where -

Q: Finite set of states.
X: Tape alphabet
Σ: Input alphabet
δ: Transition function; $\delta: Q \times X \rightarrow Q \times X \times$ (Left_shift, right_shift)

Sign :- Ekveera

q_0 : Initial state
B : Blank Symbol
F : set of final states.



Variants of TM:-

(i) Multitape Turing machine: The TM is allowed to have 'k' tapes. It has a separate head in each of the 'k' tapes. In each transition step, each tape replaces the bit under its respective head and moves left / right. All the 'k' tapes can also be simulated by a single tape. The content of different tapes are written side by side, separated by a marker.

(ii) Two stacks: The two stacks store the contents of the tape on the left side and right side of the tape head respectively.

(iii) Queue: It can be simulated using two stacks.

Sign: Skveera

NDTM (non deterministic TM)

The TM can move to multiple configurations simultaneously.

Transition function :- $\delta : Q \times T \longrightarrow 2^{Q \times T \times (L,R)}$

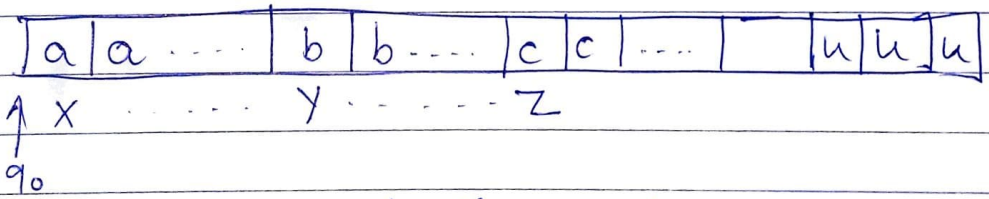
(b) $L = \{a^n b^n c^n \mid n >= 1\}$

If n = 1, abc

n = 2, aabbcc

$\therefore W = \{abc, aabbcc, \ldots\}$

Turing m/c confirm in shell, we store the value with n of a & n of n.



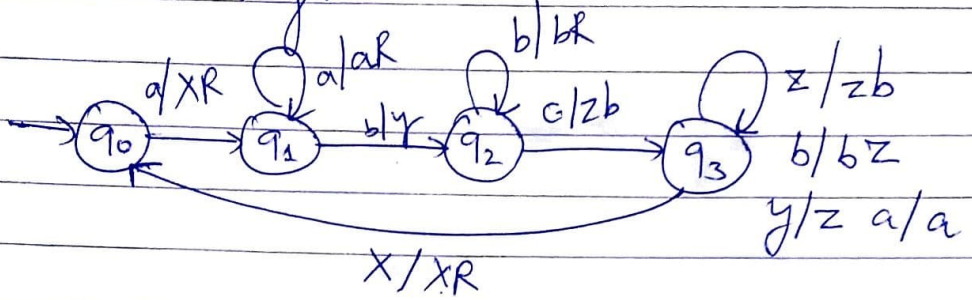
Left most side is length and right most side is infinite and it contains some whitespace

Turing machine starts from left to right in our language as there are equal no. of a, b & c.

In q_0, transition state from q_0, R is moving right. In q_0, input a is cancel used symbol X, as like for b 'Y', C for Z,

Sign :- Ekveera

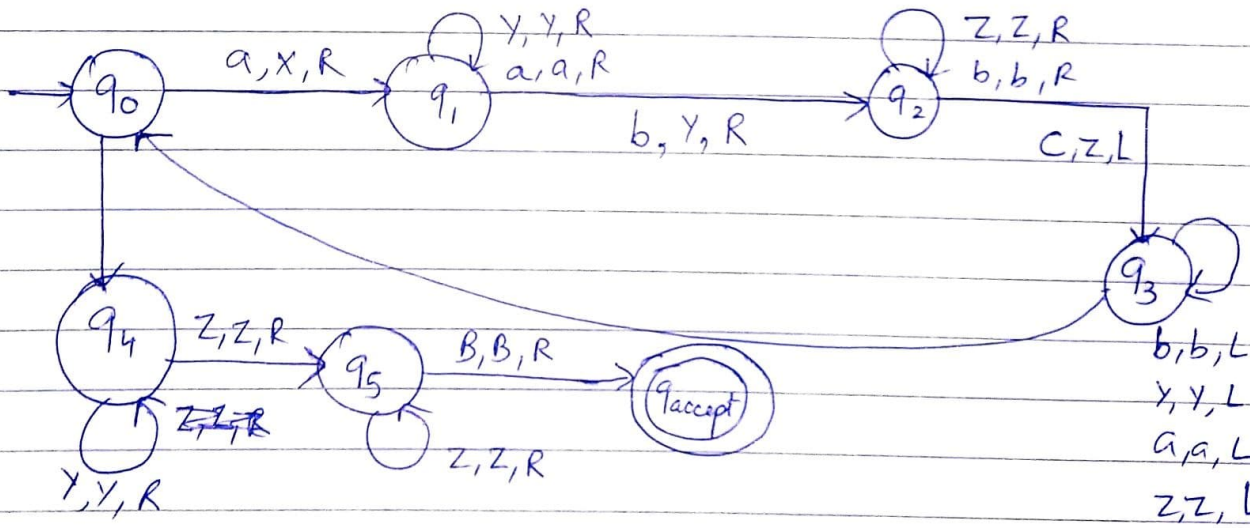
R moves right.



We move until 'a' doesn't end and same is with b & c. Now, we will go back to left to 'a' by using same method to left to reach the X & then start with same procedure in loop for calculation of 'a'.

So required is :-

B	B	X	X	Y	Y	Y	Z	Z	Z	B	B



$\delta(q_0, b) = (q, \text{reject } C, R)$
$\delta(q_4, c) = (q, \text{reject } C, R)$
$\delta(q, z) = (q, \text{reject } C, R)$
$\delta(q, c) = (q, \text{reject } C, R)$
$\delta(q, u) = (q, \text{reject } C, R)$

Rejection state.

7. (a) A context free grammar is a formal grammar which is used to generate all possible strings in a given formal language.

CFG can be defined by four tuples as :-

$G = (V, T, P, S)$

where,

G : describes grammar
V : Finite set of non-terminal symbols
P : Production Rules set
S : Start Symbol
T : ~~set~~ finite set of terminal symbols

In CFG, the start symbol is used to derive the string. String can be derived by repeatedly replacing a non-terminal by the

Sign :- Shreera

right hand side of the production until all non-terminal have been replaced by terminal symbols.

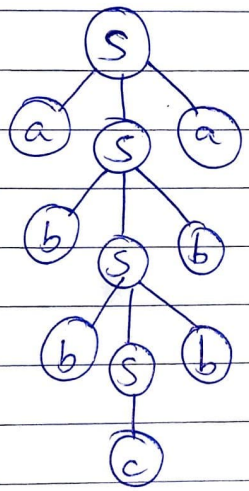
Eg:- $L = \{ wcw^R \mid w \in (a,b)^* \}$

P:-

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow c$



String generated through this tree is :-
abbcbba

There are various capabilities of CFG:-

→ Useful to describe most of the programming languages.

→ Capable of describing nested structures like balanced parathesis, matching begin-end, etc.

Closure properties of CFG :-

Sign:- Ekveera

Context-free languages are closed under :-

(i) Union

If L_1 & L_2 are two CFLs then $L_1 \cup L_2$ is also a context free language.

Eg :-

G_1 :- $S_1 \rightarrow aSb / \lambda$ L_1 = 'a' followed by equal no. of 'b'

G_2 :- $S_2 \rightarrow bSc / \lambda$ L_2 = 'b' " " " " 'c'

Union :- $S \rightarrow S_1 / S_2$

$L = L_1 \cup L_2 = \{a^n b^n\} \cup \{c^m d^m\}$ is also a CFL.

(ii) Concatenation

If L_1 & L_2 are two CFLs then $L_1 . L_2$ is also a CFL.

Eg :- $L = L_1 . L_2 = \{a^n b^n c^m d^m\}$ is also a CFL. ; G_1 :- $S \rightarrow S1\,S2$

(iii) Kleen closure

If L is a CFL then L^* is also CFL.

Eg :- $L = \{a^n b^n, n \geq 0\}$, G_1 :- $P : S \rightarrow aAb / \varepsilon$

$L_1 = \{a^n b^n\}^*$

Corresponding G_1 will have productions as :-

$S1 \rightarrow SS_1 / \varepsilon$

Context free languages are not closed under :- intersection and complementation.

Sign :- Shreera

(b) A turing machine is defined as :-

$M = \{Q, X, \Sigma, \delta, q_0, B, F\}$ where :-

Q = set of finite states

X = tape alphabet

Σ = input alphabet

δ = Transition function

$\delta : Q \times X \longrightarrow Q \times X \times \{\text{Left\_shift}, \text{right\_shift}\}$

q_0 = initial state

B = Blank symbol

F = set of final states

Time and complexity of a TM :-

Time complexity : refers to the no. of times the tape moves when the machine is initialized for some input symbols.

Space complexity : No. of cells of the tape written

T.C $\Rightarrow$ $T(n) = O(n \log n)$

S.C $\Rightarrow$ $S(n) = O(n)$

Representation of Turing machine :-

(i) Instantaneous description

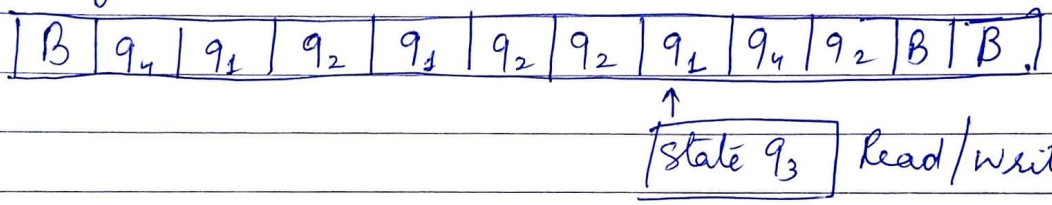
(ii) Transition table

(iii) Transition diagram

Sign :- Shreera

(i) Instantaneous description :-

An ID of a turing machine is defined in terms of the entire input string and the current state. An ID of a turing machine M is a string $\alpha\beta\gamma$ where β is the present state of M. The entire input string is split as $\alpha\gamma$. The first symbol of γ is the current symbol, say 'a'.



(ii) Transition Table

The definition of δ can be given in the form of a table called transition table.

If $\delta(q, a) = (\gamma, \alpha, \beta)$,
α, β, γ under q - column & q - row.

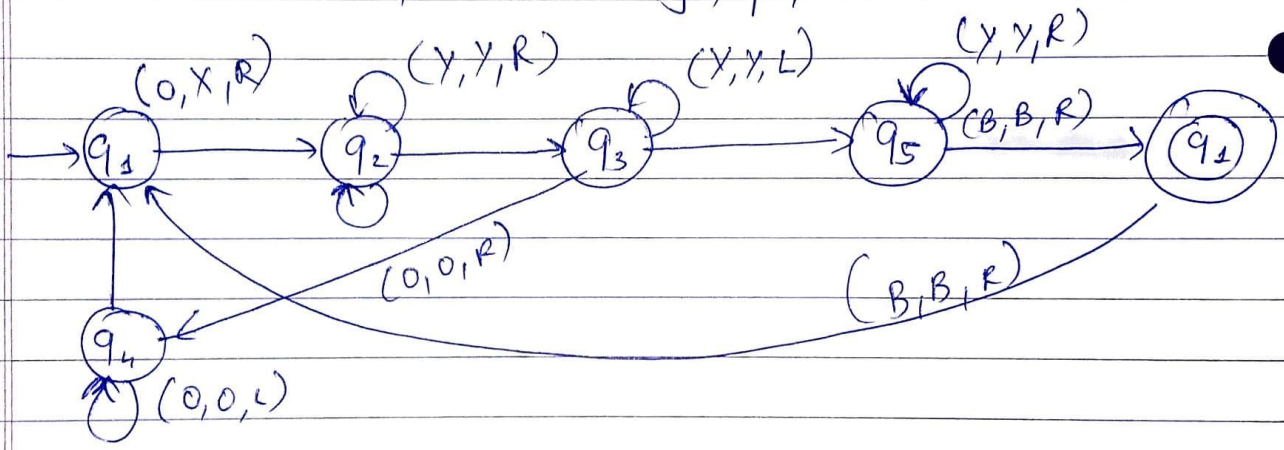
So, if we get $\alpha\beta\gamma$ in the table
$\gamma \rightarrow$ current cell
$\beta \rightarrow$ given movement of head
$\gamma \rightarrow$ new state into which the turing machine state is

(iii) Transition diagram

Sign : Shreeya

The states are represented by vertices and directed edges are used to represent transition of states.
The labels are tuples of the form (α, β, γ) when there is a directed edge from q_i to q_j, with label (α, β, γ)

$$\delta(q_i, a) = (q_j, \beta, \gamma)$$



Sign:- Shreeya