

**DR. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY, LONERE –
RAIGAD -402 103**

Semester Examination – December - 2019

Branch: Computer Science and Engineering

Sem:- V

Subject with Subject Code: - Theory of Computations. (BTCOC502)

Marks: 60

Date:- /11/2019

Time:- 3 Hr.

=====

Instructions to the Students

1. Each question carries 12 marks.
 2. Attempt **any five** questions of the following.
 3. Illustrate your answers with neat sketches, diagram etc., wherever necessary.
 4. If some part or parameter is noticed to be missing, you may appropriately assume it and should mention it clearly
-

(Marks)

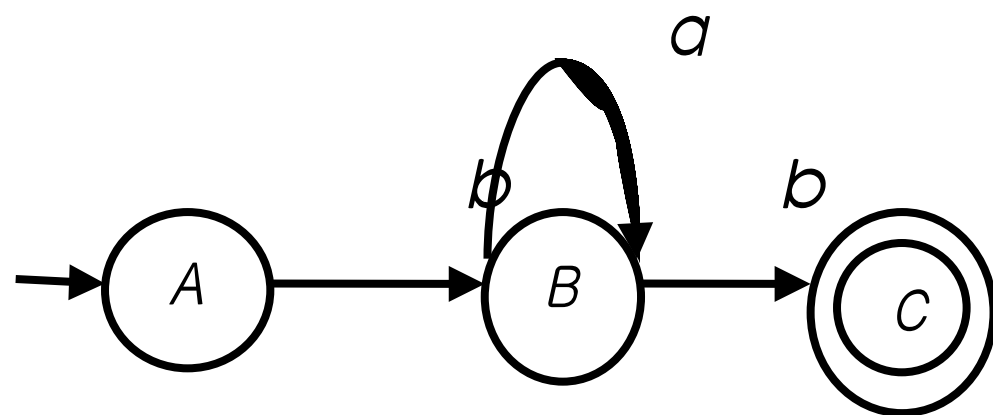
**Q1. a) Convert following regular expression to their equivalent FA
(10)**

i) ba^*b

ii) $(a+b) c$

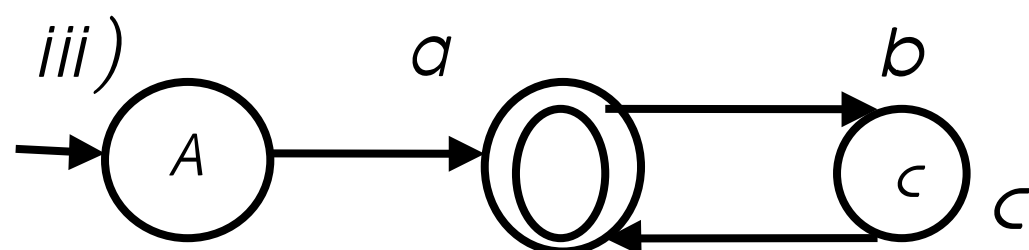
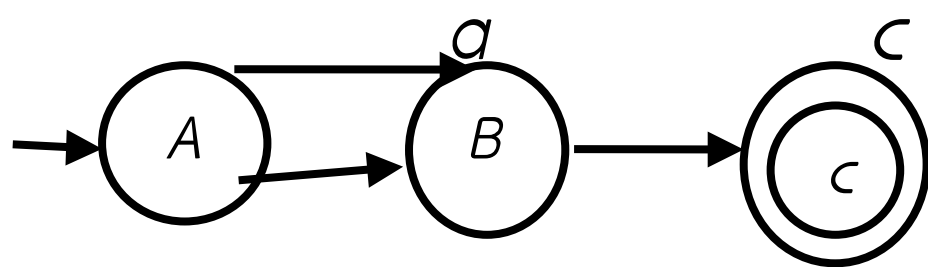
iii) $a (bc)^*$

i) $L = \{bb, bab, baab, baaab, \dots\}$



ii) $(a+b) c$

$L = \{ac, bc\}$



$L = \{a, abc, abcbc, abcbcbc, \dots\}$

b) Write note on Chomsky hierarchy in detail.
(07)

=> *Chomsky Hierarchy*

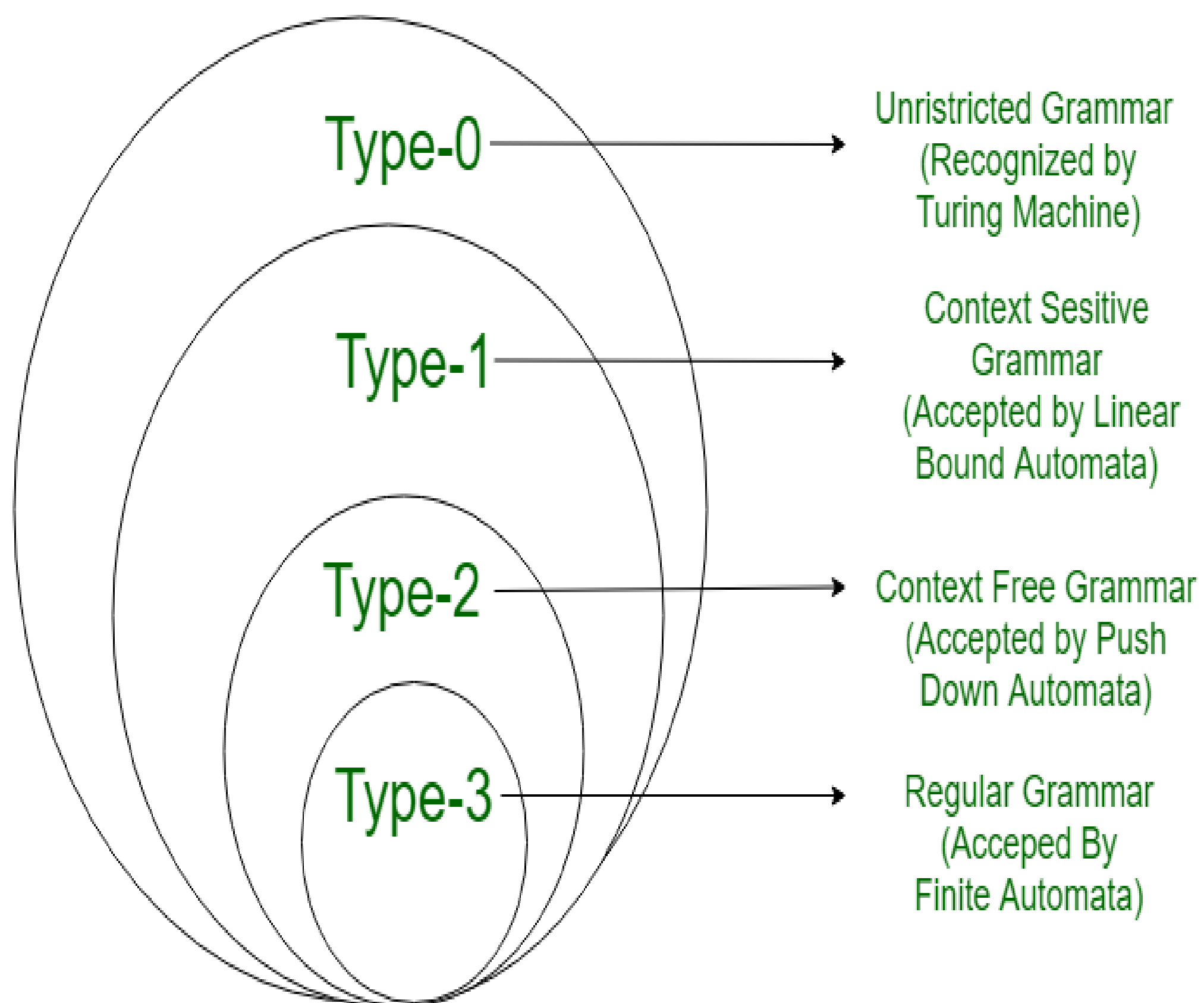
According to Chomsky hierarchy, grammars are divided of 4 types:

Type 0 known as unrestricted grammar.

Type 1 known as context sensitive grammar.

Type 2 known as context free grammar.

Type 3 known as Regular Grammar.



Type 0: Unrestricted Grammar:

Type-0 grammars include all formal grammars. Type 0 grammar language are recognized by turing machine. These languages are also known as the Recursively Enumerable languages.

Grammar Production in the form of $a \rightarrow \beta$

where

a is $(V + T)^ V (V + T)^*$*

V : Variables

T : Terminals.

β is $(V + T)^$.*

In type 0 there must be at least one variable on Left side of production.

For example,

$Sab \rightarrow ba$
 $A \rightarrow S.$

Here, Variables are S , A and Terminals a , b .

Type I: Context Sensitive Grammar

Type-I grammars generate the context-sensitive languages. The language generated by the grammar are recognized by the Linear Bound Automata

In Type I

- I. First of all Type I grammar should be Type 0.
- II. Grammar Production in the form of $a \rightarrow \beta$

Where $|a| \leq |\beta|$

i.e count of symbol in a is less than or equal to β

For Example,

$S \rightarrow AB$

$AB \rightarrow abc$

$B \rightarrow b$

Type 2: Context Free Grammar:

Type-2 grammars generate the context-free languages. The language generated by the grammar is recognized by a Pushdown automata. Type-2 grammars generate the context-free languages.

In Type 2,

- I. First of all it should be Type I.
2. Left hand side of production can have only one variable.
 $|a| = 1.$

There is no restriction on β

For example,

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

Type 3: Regular Grammar:

Type-3 grammars generate regular languages. These languages are exactly all languages that can be accepted by a finite state automaton.

Type 3 is most restricted form of grammar.
Type 3 should be in the given form only:

$$V \rightarrow VT^* / T^*$$

(or)

$$V \rightarrow T^*V / T^*$$

for example :

$$S \rightarrow ab.$$

Q2. a) Convert the given Grammar into Chomsky Normal Form (CNF)
(05)

$$S \rightarrow ASB$$

$$A \rightarrow aAS / a / \epsilon$$

$$B \rightarrow SbS / A / bb$$

=> **Step 1.** As start symbol S appears on the RHS, we will create a new production rule $SO \rightarrow S$. Therefore, the grammar will become:
 $SO \rightarrow S$

$$S \rightarrow ASB$$

$$A \rightarrow aAS / a / \epsilon$$

$$B \rightarrow SbS / A / bb$$

Step 2. As grammar contains null production $A \rightarrow \epsilon$, its removal from the grammar yields:

$$SO \rightarrow S$$

$$\underline{S \rightarrow ASB / SB}$$

$$\underline{A \rightarrow aAS / aS / a}$$

$$\underline{B \rightarrow SbS / A / \epsilon / bb}$$

Now, it creates null production $B \rightarrow \epsilon$, its removal from the grammar yields:

$SO \rightarrow S$

$S \rightarrow AS/ASB/ SB/ S$

$A \rightarrow aAS/aS/a$

$B \rightarrow SbS/ A/bb$

Now, it creates unit production $B \rightarrow A$, its removal from the grammar yields:

$SO \rightarrow S$

$S \rightarrow AS/ASB/ SB/ S$

$A \rightarrow aAS/aS/a$

$B \rightarrow SbS/bb/aAS/aS/a$

Also, removal of unit production $SO \rightarrow S$ from grammar yields:

$SO \rightarrow AS/ASB/ SB/ S$

$S \rightarrow AS/ASB/ SB/ S$

$A \rightarrow aAS/aS/a$

$B \rightarrow SbS/bb/aAS/aS/a$

Also, removal of unit production $S \rightarrow S$ and $SO \rightarrow S$ from grammar yields:

$SO \rightarrow AS/ASB/ SB$

$S \rightarrow AS/ASB/ SB$

$A \rightarrow aAS/aS/a$

$B \rightarrow SbS/bb/aAS/aS/a$

Step 3. In production rule $A \rightarrow aAS/aS$ and $B \rightarrow SbS/aAS/aS$, terminals a and b exist on RHS with non-terminates. Removing them from RHS:

$SO \rightarrow AS/ASB/ SB$

$S \rightarrow AS/ASB/ SB$

$A \rightarrow XAS/XS/a$

$B \rightarrow SYS/bb/XAS/XS/a$

$X \rightarrow a$

$Y \rightarrow b$

Also, $B \rightarrow bb$ can't be part of CNF, removing it from grammar yields:

$SO \rightarrow AS/ASB/ SB$

$S \rightarrow AS/ASB/ SB$

$A \rightarrow XAS/XS/a$

$B \rightarrow SYS/YY/XAS/XS/a$

$X \rightarrow a$

$Y \rightarrow b$

Step 4: In production rule $SO \rightarrow ASB$, RHS has more than two symbols, removing it from grammar yields:

$SO \rightarrow AS/PB/ SB$

$S \rightarrow AS/ASB/ SB$

$A \rightarrow XAS/XS/a$

$B \rightarrow SYS/YY/XAS/XS/a$

$X \rightarrow a$

$Y \rightarrow b$

$P \rightarrow AS$

Similarly, $S \rightarrow ASB$ has more than two symbols, removing it from grammar yields:

$SO \rightarrow AS|PB|SB$

$S \rightarrow AS|QB|SB$

$A \rightarrow XAS|XS|a$

$B \rightarrow SYS|YY|XAS|XS|a$

$X \rightarrow a$

$Y \rightarrow b$

Similarly, $A \rightarrow XAS$ has more than two symbols, removing it from grammar yields:

$SO \rightarrow AS|PB|SB$

$S \rightarrow AS|QB|SB$

$A \rightarrow RS|XS|a$

$B \rightarrow SYS|YY|XAS|XS|a$

$X \rightarrow a$

$Y \rightarrow b$

$P \rightarrow AS$

$Q \rightarrow AS$

$R \rightarrow XA$

Similarly, $B \rightarrow SYS$ has more than two symbols, removing it from grammar yields:

$SO \rightarrow AS|PB|SB$

$S \rightarrow AS|QB|SB$

$A \rightarrow RS|XS|a$

$$B \rightarrow TS/YY/XAS/XS/a$$
$$X \rightarrow a$$
$$Y \rightarrow b$$
$$P \rightarrow AS$$
$$Q \rightarrow AS$$
$$R \rightarrow XA$$
$$T \rightarrow SY$$

Similarly, $B \rightarrow XAS$ has more than two symbols, removing it from grammar yields:

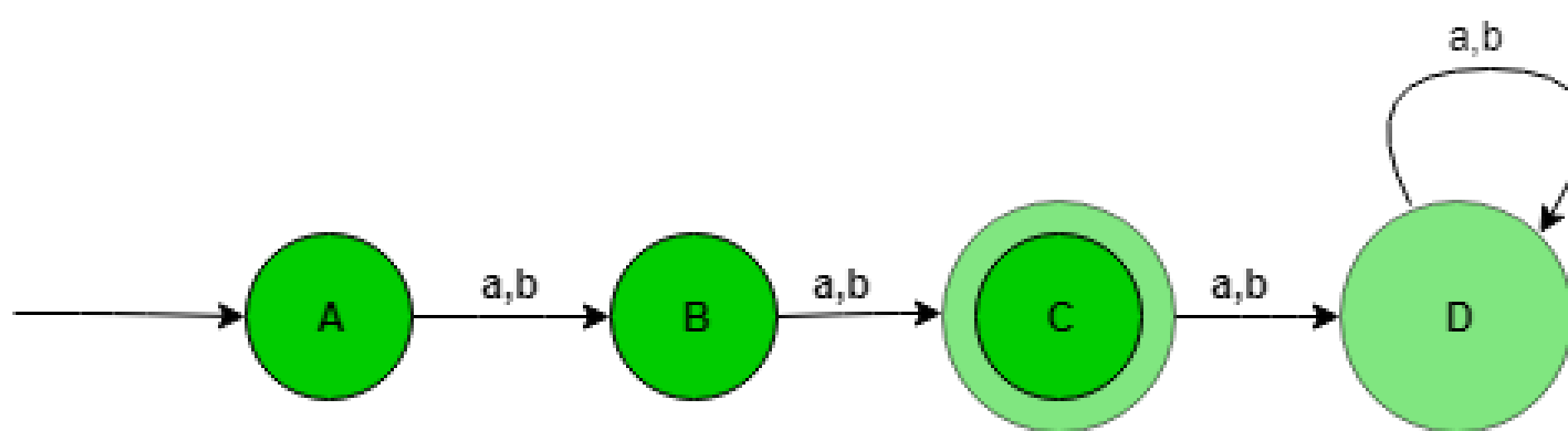
$$SO \rightarrow AS/PB/ SB$$
$$S \rightarrow AS/QB/ SB$$
$$A \rightarrow RS/XS/a$$
$$B \rightarrow TS/YY/US/XS/a$$
$$X \rightarrow a$$
$$Y \rightarrow b$$
$$P \rightarrow AS$$
$$Q \rightarrow AS$$
$$R \rightarrow XA$$
$$T \rightarrow SY$$
$$U \rightarrow XA$$

So this is the required CNF for given grammar.

b) Construction of a DFA for the set of string over $\{a, b\}$ such that length of the string $|w|=2$ i.e, length of the string is exactly 2.
(07)

=> Explanation – The desired language will be like:
 $L = \{aa, ab, ba, bb\}$

The state transition diagram of the language will be like:



Here,

State A represent set of all string of length zero (0), state B represent set of all string of length one (1), state C represent set of all string of length two (2). State C is the final state and D is the dead state it is so because after getting any alphabet as input it will not go into final state ever.

Number of states: $n+2$

Where n is $|w|=n$

The above automata will accept all the strings having the length of the string exactly 2. When the length of the string is 1, then it will go from state A to B. When the length of the string is 2, then it will go from state B to C and when the length of the string is greater than 2, then it will go from state C to D (Dead state) and after it from state D TO D itself.

Q3. a) What is pumping lemma? Explain with example. (05)

=> Theorem

Let L be a regular language. Then there exists a constant ' c ' such that for every string w in L -

$$|w| \geq c$$

We can break w into three strings, $w = xyz$, such that -

- $|y| > 0$
- $|xy| \leq c$
- For all $k \geq 0$, the string xy^kz is also in L .

Example:

Problem

Prove that $L = \{a^i b^i \mid i \geq 0\}$ is not regular.

Solution -

- At first, we assume that L is regular and n is the number of states.
- Let $w = a^n b^n$. Thus $|w| = 2n \geq n$.
- By pumping lemma, let $w = xyz$, where $|xy| \leq n$.
- Let $x = a^p$, $y = a^q$, and $z = a^r b^n$, where $p + q + r = n$, $p \neq 0$, $q \neq 0$, $r \neq 0$. Thus $|y| \neq 0$.
- Let $k = 2$. Then $xy^2z = a^p a^{2q} a^r b^n$.
- Number of a s = $(p + 2q + r) = (p + q + r) + q = n + q$
- Hence, $xy^2z = a^{n+q} b^n$. Since $q \neq 0$, xy^2z is not of the form $a^n b^n$.
- Thus, xy^2z is not in L . Hence L is not regular.

b) Construct a PDA for language $L = \{w c w^R \mid w = \{0, 1\}^*\}$
(07)

where w^R is the reverse of w .

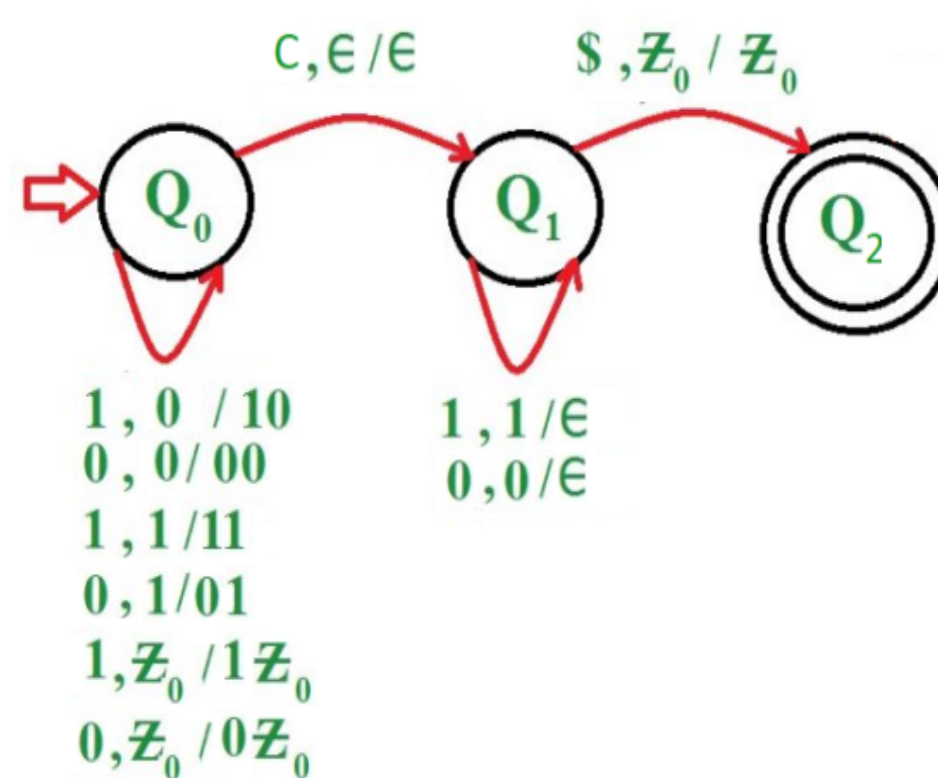
=> **Approach used in this PDA** –
 Keep on pushing 0's and 1's no matter whatever is on the top of stack until reach the middle element. When middle element 'c' is scanned then process it without making any changes in stack. Now if scanned symbol is '1' and top of stack also contain '1' then pop the element from top of stack or if scanned symbol is '0' and top of stack also contain '0' then pop the element from top of stack. If string becomes empty or scanned symbol is '\$' and stack becomes empty, then reach to final state else move to dead state.

Step 1: On receiving 0 or 1, keep on pushing it on top of stack without going to next state

Step 2: On receiving an element 'c', move to next state without making any change in stack.

Step 3: On receiving an element, check if symbol scanned is '1' and top of stack also contain '1' or if symbol scanned is '0' and top of stack also contain '0' then pop the element from top of stack else move to dead state. Keep on repeating step 3 until string becomes empty.

Step 4: Check if symbol scanned is '\$' and stack does not contain any element then move to final state else move to dead state.



Input : 1 0 1 0 1 0 1

Output :ACCEPTED

Input : 1 0 1 0 1 1 1 0

Output :NOT ACCEPTED

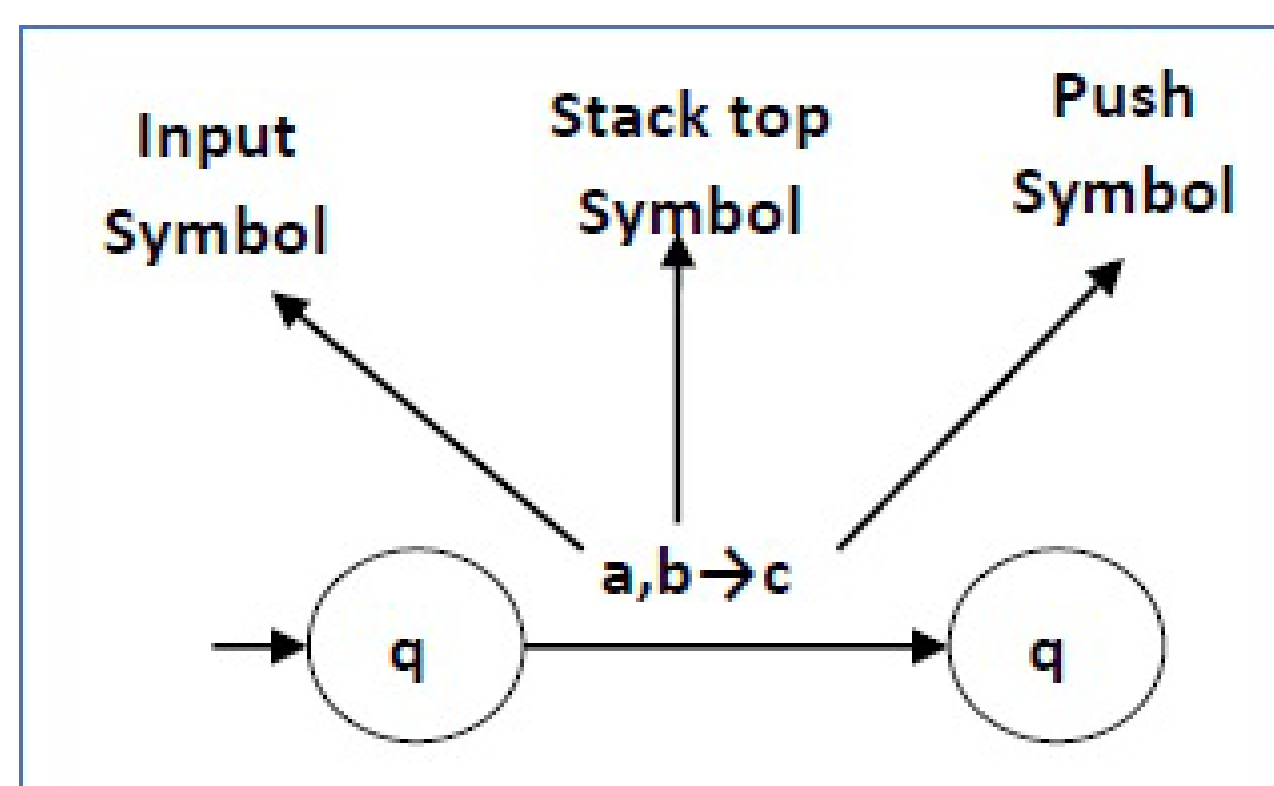
Q4. a) Define following Terms:
(05)

- i) Give formal definition of PDA with example.
- ii) Give formal definition of TM with example.

=>i) PDA can be formally described as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ -

- Q is the finite number of states
- Σ is input alphabet
- Γ is stack symbols or alphabet
- δ is the transition function: $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$
- q_0 is the initial state ($q_0 \in Q$)
- Z_0 is the initial stack top symbol ($Z_0 \in \Gamma$)
- F is a set of accepting states ($F \subseteq Q$)

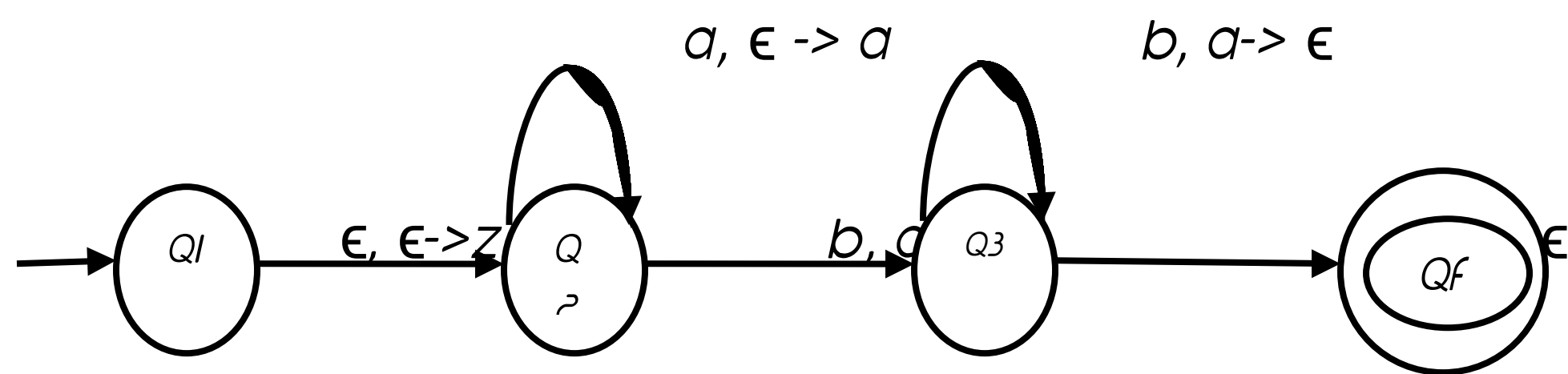
The following diagram shows a transition in a PDA from a state q_1 to state q_2 , labeled as $a, b \rightarrow c$ -



This means at state q_1 , if we encounter an input string ' a ' and top symbol of the stack is ' b ', then we pop ' b ', push ' c ' on top of the stack and move to state q_2 .

Example: PDA of $L = \{ a^n b^n \mid n \geq 1 \}$

$L = \{ab, aabb, aaabbb, aaaabbbb, \dots\}$



Stack transition functions –

$$\delta(q_1, \epsilon, Z_0) \rightarrow (q_1, Z_0)$$

$$\delta(q_1, a, Z_0) \rightarrow (q_2, aZ_0) \text{ // PUSH}$$

$$\delta(q_2, a, a) \rightarrow (q_2, aaZ_0) \text{ // PUSH}$$

$$\delta(q_2, b, a) \rightarrow (q_3, \epsilon) \text{ // POP}$$

$$\delta(q_3, \epsilon, Z_0) \rightarrow (q_f, \epsilon)$$

ii) Give formal definition of TM with example.

=> A Turing Machine is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars. It was invented in 1936 by Alan Turing.

Definition

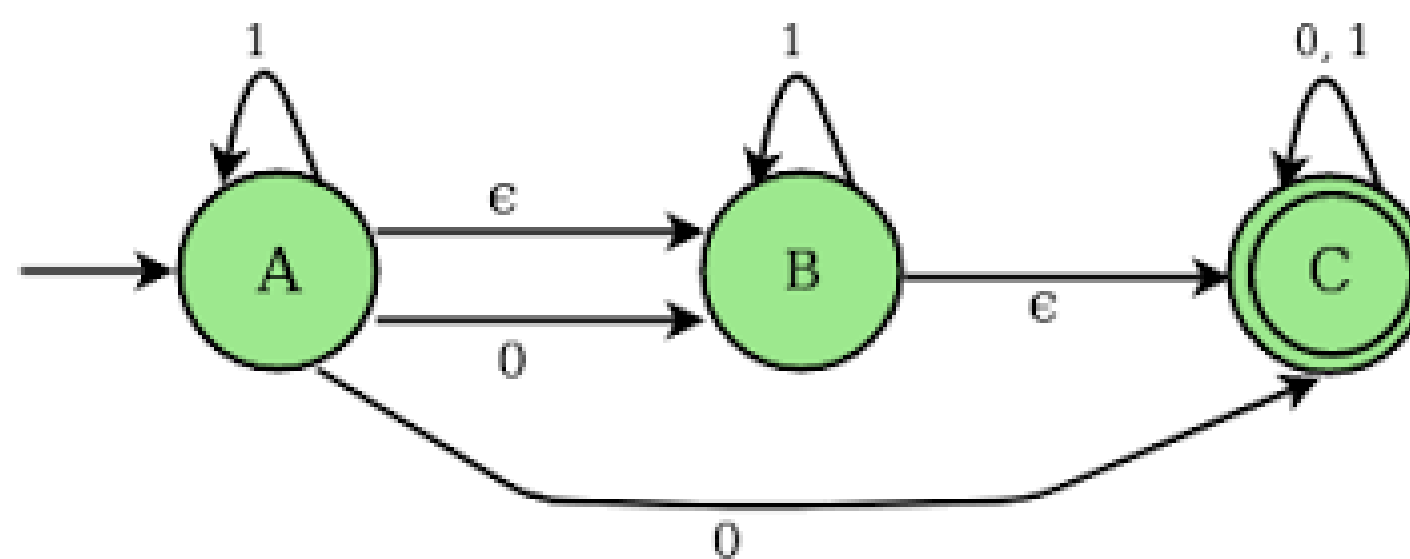
A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, 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 formally described as a 7-tuple $(Q, \Gamma, \Sigma, \delta, q_0, B, F)$ where –

- Q is a finite set of states
- Γ is the tape alphabet
- Σ is the input alphabet
- δ is a transition function; $\delta : Q \times \Gamma \rightarrow Q \times \Gamma^* \times \{\text{Left_shift}, \text{Right_shift}\}$.
- q_0 is the initial state
- B is the blank symbol ($B \in \Gamma$)
- F is the set of final states (ACCEPT OR REJECT)

Example of any TM language.

b) Convert following NFA- ϵ to DFA.
(07)



$$\in \text{closure}(A) = \{A, B, C\} \in \text{closure}(B) = \{B, C\} \in \text{closure}(C) = \{C\}$$

Step 1 : Take $\in \text{closure}$ for the beginning state of NFA as beginning state of DFA.

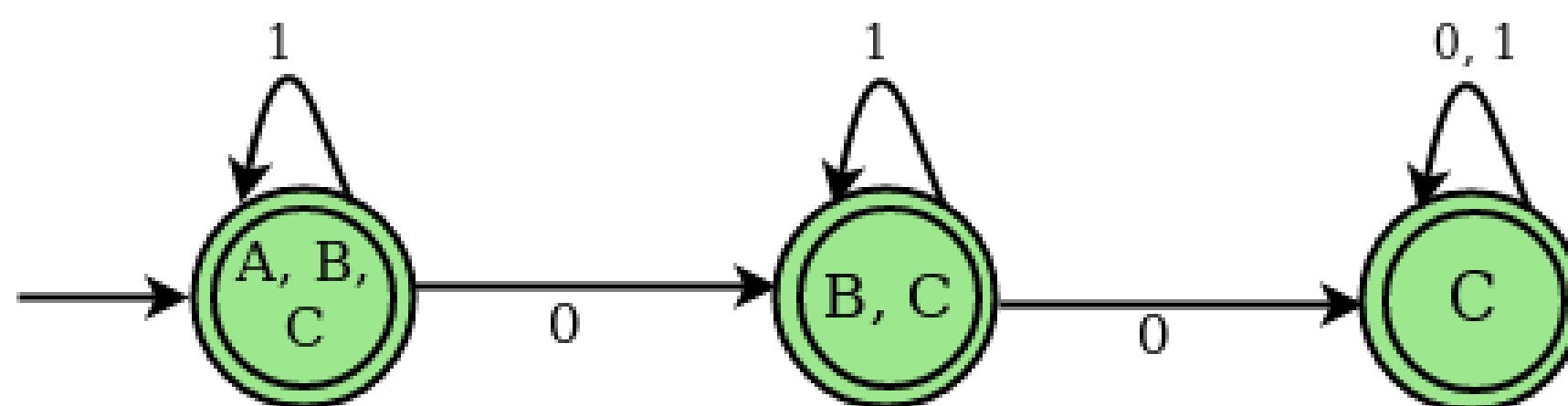
Step 2 : Find the states that can be traversed from the present for each input symbol
(union of transition value and their closures for each states of NFA present in current state of DFA).

Step 3 : If any new state is found take it as current state and repeat step 2.

Step 4 : Do repeat Step 2 and Step 3 until no new state present in DFA transition table.

Step 5 : Mark the states of DFA which contains final state of NFA as final states of DFA.

STATES	0	1
A, B, C	B, C	A, B, C
B, C	C	B, C
C	C	C



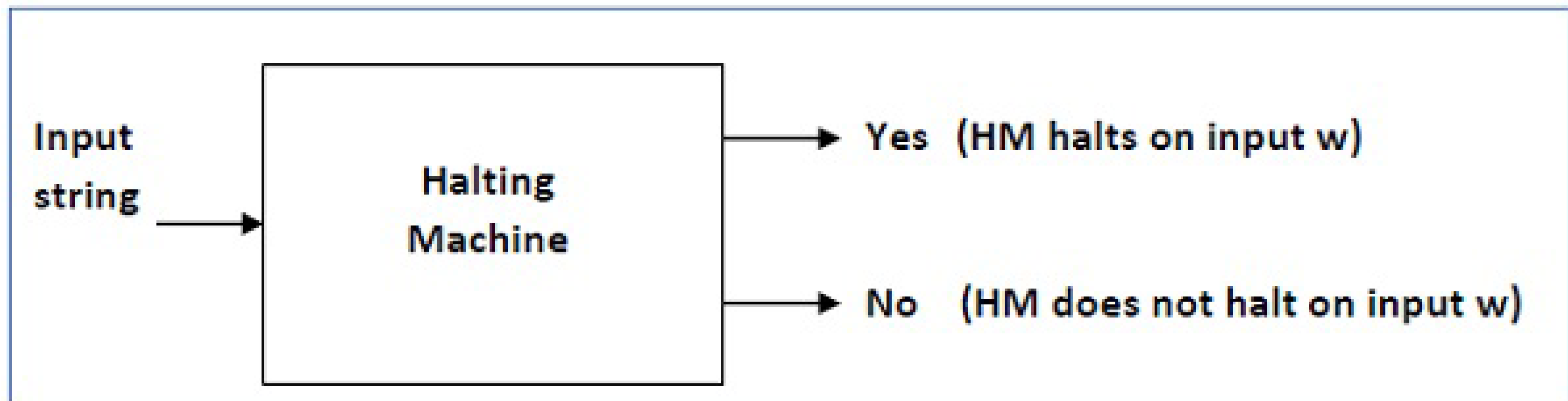
Q5. a) Explain in detail TM halting problem is undecidable.

(05)

=> **Input** – A Turing machine and an input string w .

Problem – Does the Turing machine finish computing of the string w in a finite number of steps? The answer must be either yes or no.

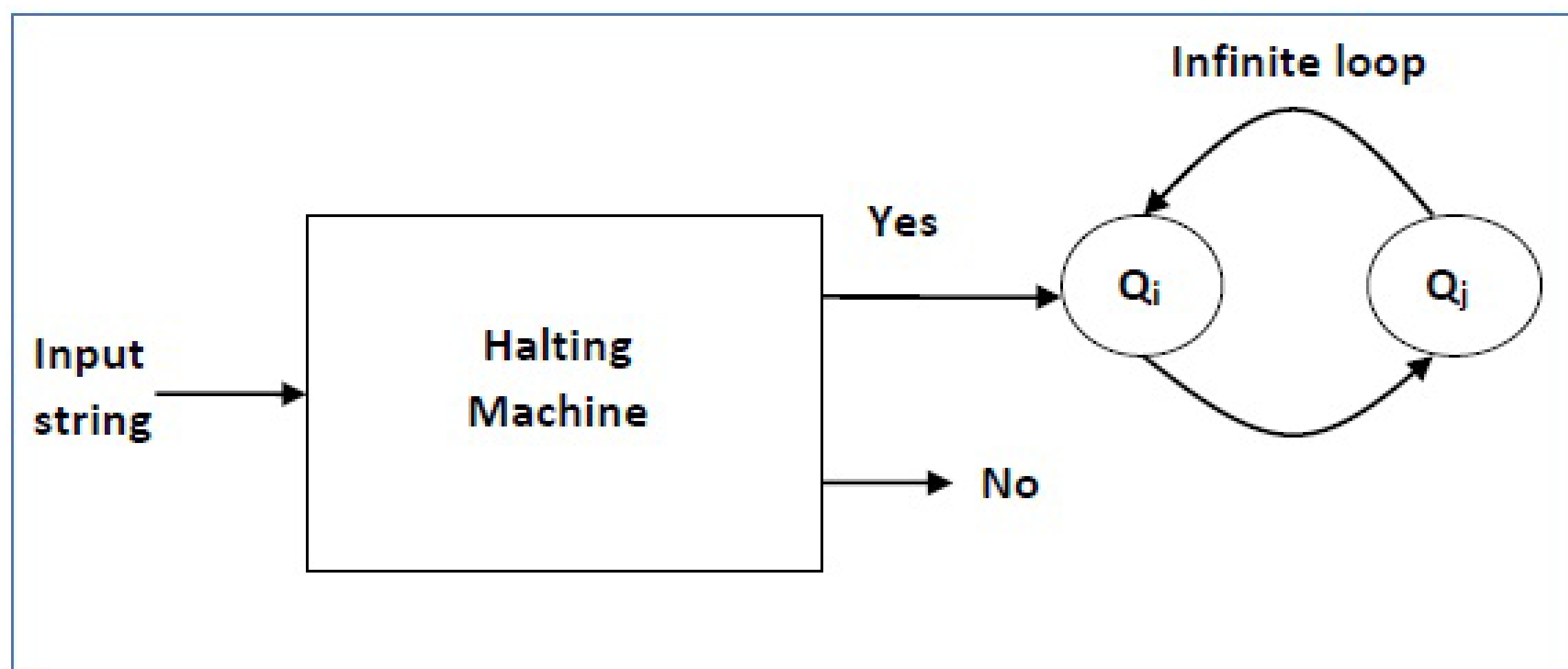
Proof – At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine –



Now we will design an *inverted halting machine (HM)'* as –

- If H returns YES, then loop forever.
- If H returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine' –



Further, a machine $(HM)_2$ which input itself is constructed as follows –

- If $(HM)_2$ halts on input, loop forever.
- Else, halt.

Here, we have got a contradiction. Hence, the halting problem is undecidable.

b) Define Mealy machine and Moore machine and

Convert following Mealy machine to Moore machine.

(07)

State	Input	
	a	b
Q_0	$Q_{2,1}$	$Q_{3,0}$
Q_1	$Q_{0,0}$	$Q_{1,1}$
Q_2	$Q_{1,1}$	$Q_{2,0}$
Q_3	$Q_{2,0}$	$Q_{0,1}$

=> Finite automata may have outputs corresponding to each transition. There are two types of finite state machines that generate output –

- Mealy Machine
- Moore machine

Mealy Machine

A Mealy Machine is an FSM whose output depends on the present state as well as the present input.

It can be described by a 6 tuple $(Q, \Sigma, O, \delta, \lambda, q_0)$ where –

- Q is a finite set of states.
- Σ is a finite set of symbols called the input alphabet.

- O is a finite set of symbols called the output alphabet.
- δ is the input transition function where $\delta: Q \times \Sigma \rightarrow Q$
- λ is the output transition function where $\lambda: Q \times \Sigma \rightarrow O$
- q_0 is the initial state from where any input is processed ($q_0 \in Q$).

Moore Machine

Moore machine is an FSM whose outputs depend on only the present state.

A Moore machine can be described by a 6 tuple $(Q, \Sigma, O, \delta, \lambda, q_0)$ where -

- Q is a finite set of states.
- Σ is a finite set of symbols called the input alphabet.
- O is a finite set of symbols called the output alphabet.
- δ is the input transition function where $\delta: Q \times \Sigma \rightarrow Q$
- λ is the output transition function where $\lambda: Q \rightarrow O$
- q_0 is the initial state from where any input is processed ($q_0 \in Q$).

Conversion of Mealy to Moore Machine

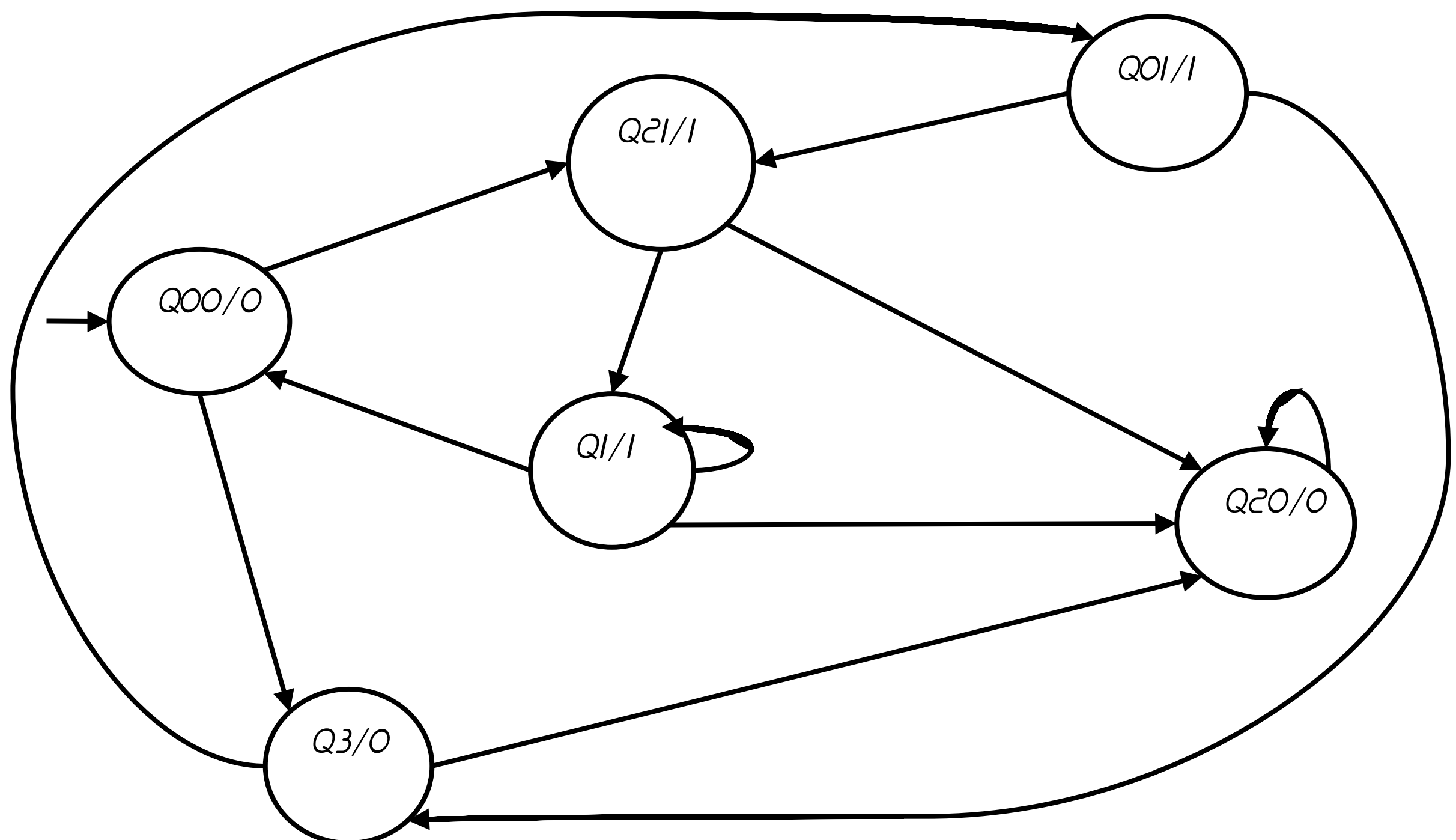
Step 1 is construct Mealy machine according to give transition table.

Step 2 is construct respective Moore machine transition table.

State	Input		Output
	a	b	
Q_{00}	Q_{21}	Q_3	0
Q_{01}	Q_{21}	Q_3	1
Q_1	Q_{00}	Q_1	1

Q_{20}	Q_1	Q_{20}	0
Q_{21}	Q_1	Q_{20}	1
Q_3	Q_{20}	Q_{01}	0

Step 3 is construct Moore machine diagram according to transition table of moore machine in step 2.



Q6. a) What is ambiguous CFG? Explain an example of ambiguous CFG.
(05)

=> for a context free grammar G has more than one derivation tree for some string

$w \in L(G)$, it is called an **ambiguous grammar**. There exist multiple right-most or left-most derivations for some string generated from that grammar.

Problem

Check whether the grammar G with production rules -

$$X \rightarrow X+X \mid X*X \mid X \mid a$$

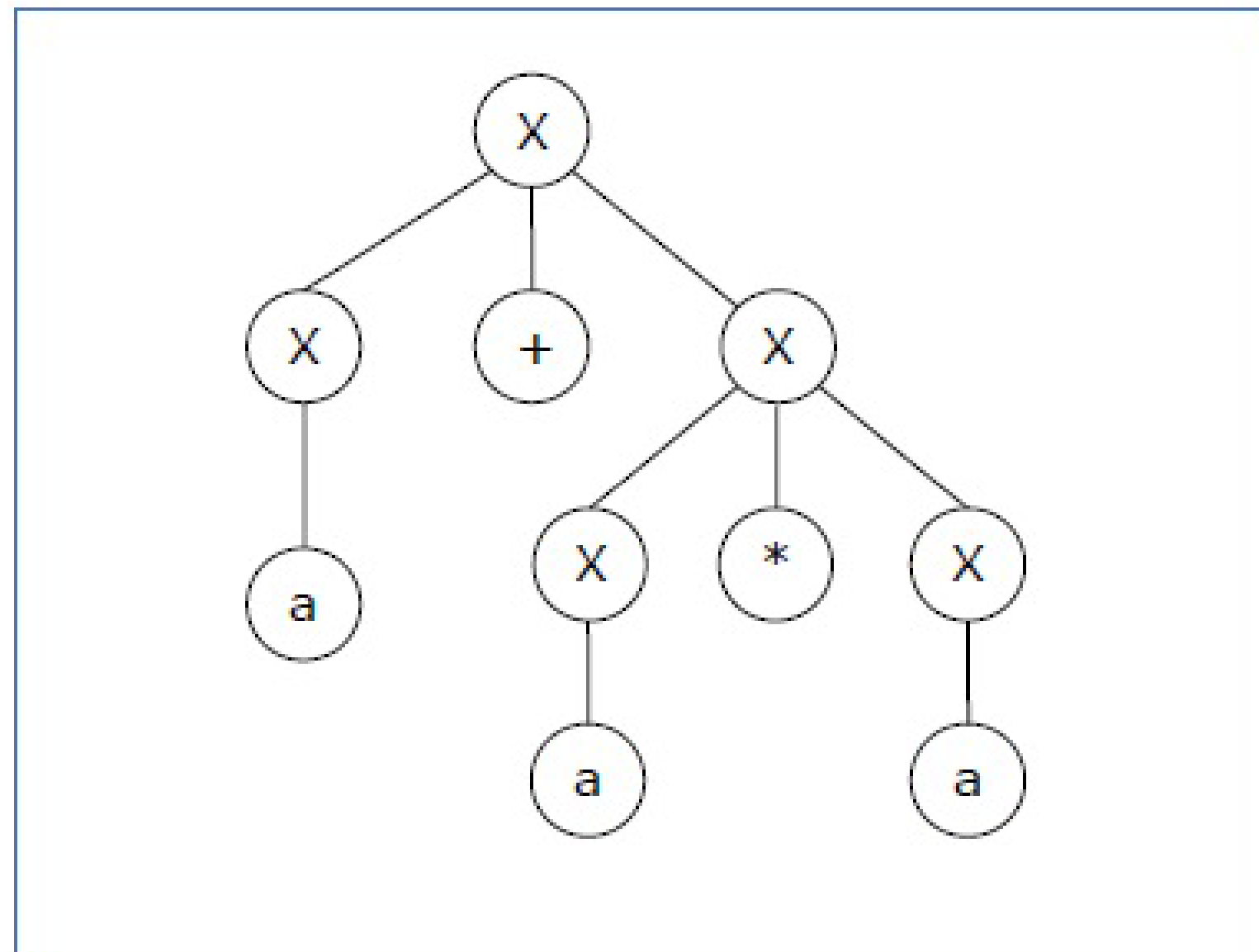
is ambiguous or not.

Solution

Let's find out the derivation tree for the string " $a+a*a$ ". It has two leftmost derivations.

Derivation 1 - $X \rightarrow X+X \rightarrow a+X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$

Parse tree 1 -



Since there are two parse trees for a single string " $a+a*a$ ", the grammar G is ambiguous.

DERIVATION-2

$X \rightarrow X*X \rightarrow X+X*X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$

b) Construct the TM which recognize the language $L=01^*0$.

(7)

=>

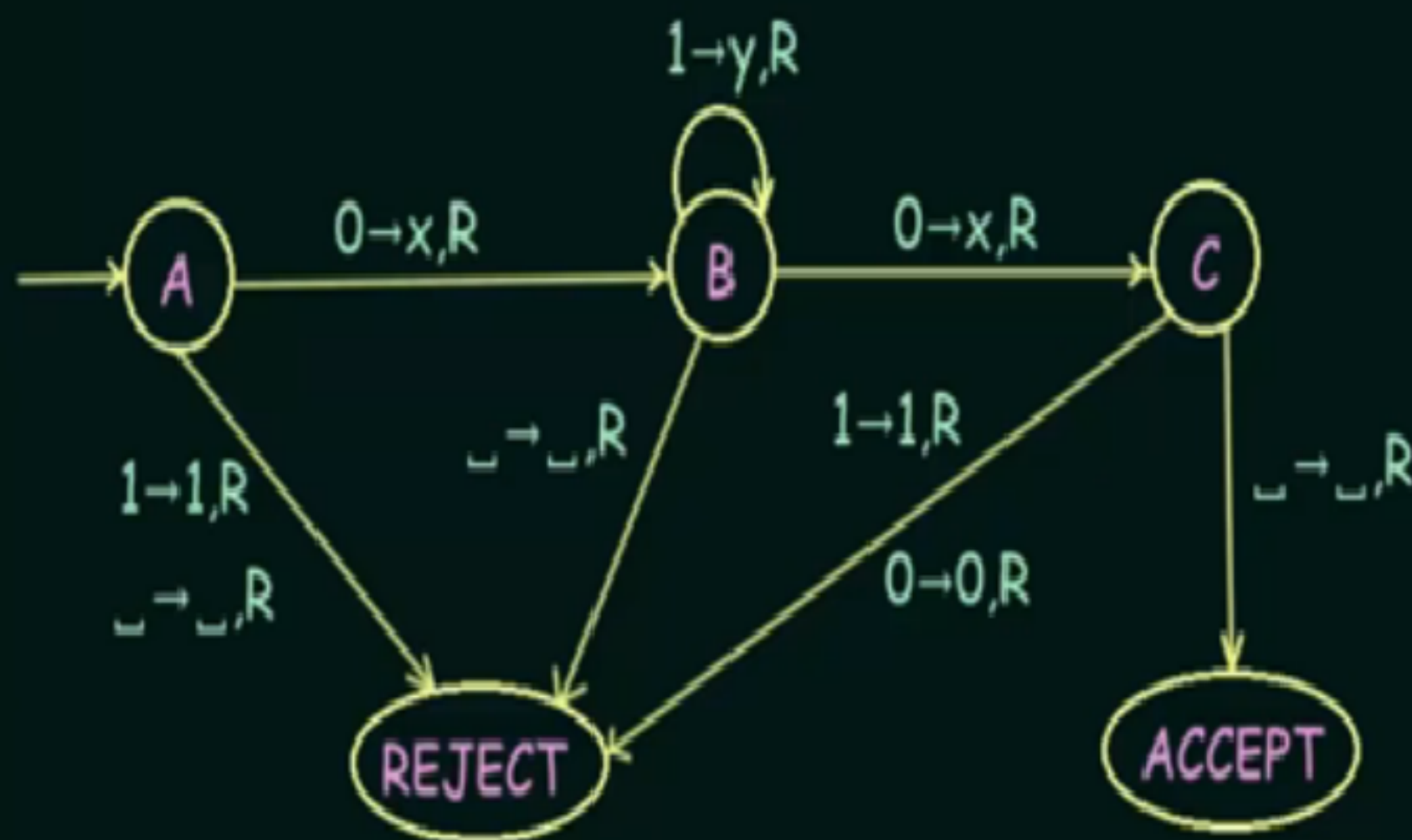
Design a Turing Machine which recognizes the language

$$L = 01^*0$$

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

$$b = \sqcup$$

0110 ✓



Explanation is required.