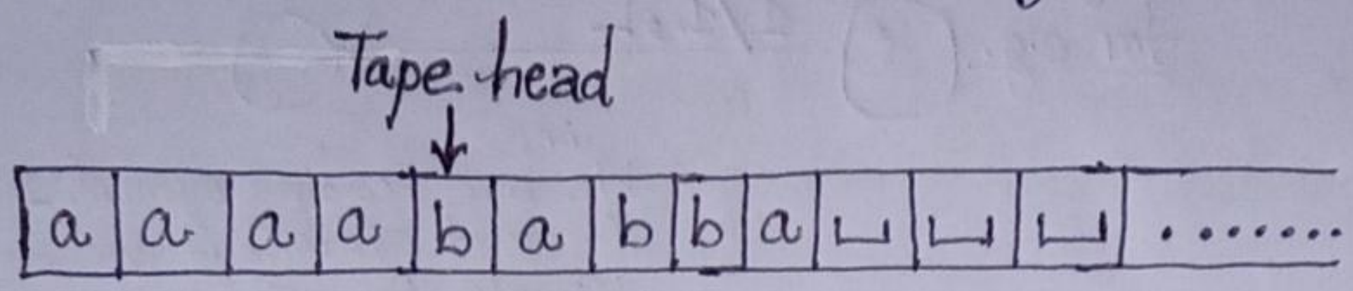
Unit-6

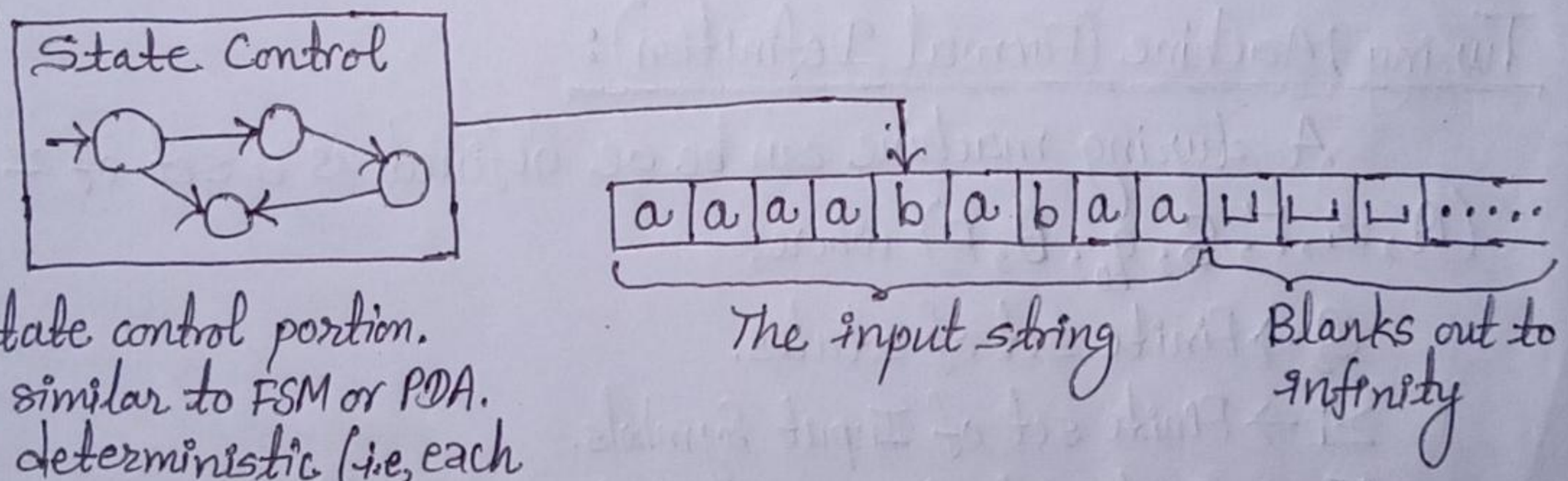
Turing Machines

Basics: In turing machine we have some sort of data structure known as tape. Tape looks like as the diagram shown below:-



As we see tape is a sequence of infinite symbols over which there is an arrow known as tape head, which is positioned over the symbol on which current control is present. Tape head can move either one step left or one step right at a single time of computation. Tape can contain any kind of alphabets like 0,1,a,b,x etc, and it also contains a special symbol i.e, the blank ⊔ symbol. It is a special symbol used to fill the infinite tape. Read, Write, Move the tape head one step LEFT, Move the tape head one step RIGHT are the operations that can be performed on tape.

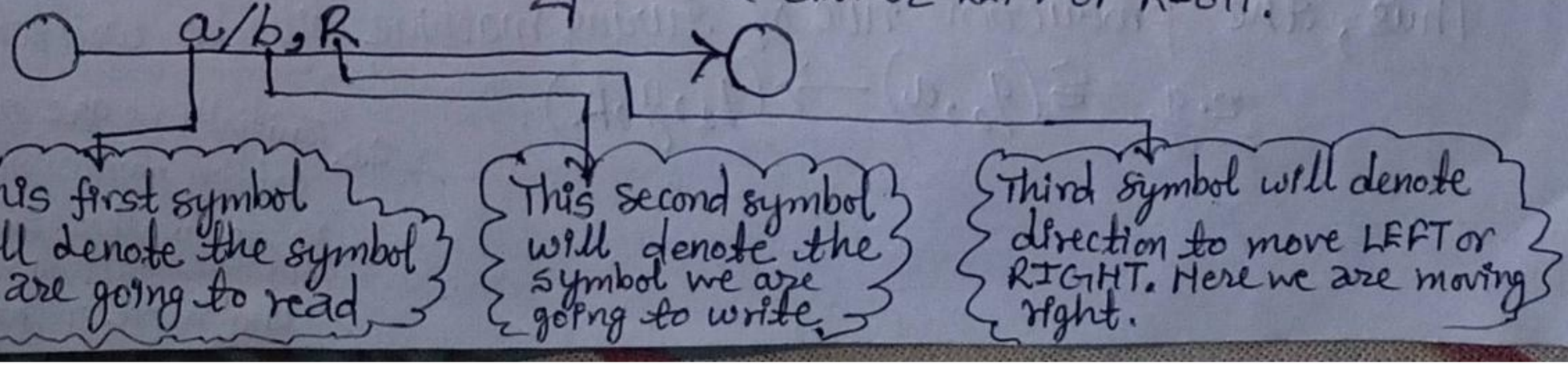
Turing Machine:



The state control portion.
It is similar to FSM or PDA.
It is deterministic (i.e, each state must be defined for all input symbols).

Rules of Operation:

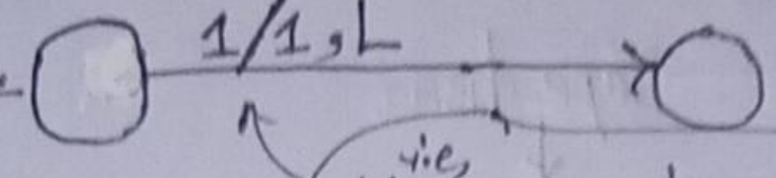
Rule 1: At each step of the computation:
- → Read the current symbol
- → Write the same cell.
- → Move exactly one cell either LEFT or RIGHT.



Note:

i) If we are at the left end of the tape and trying to move LEFT, then do not move, just stay at left end.

ii) If we don't want to write/update the cell, then just write the same symbol.

for e.g. 

i.e, We are reading symbol 1 on which we go to next state. but we do not want to update the cell, so we write same symbol 1 to cell again. L denotes move tape head to left.

Rule 2:

→ Initial State

→ Final states: (there are two final states)

i) The ACCEPT state.

ii) The REJECT state.

→ Computation can either;

i) HALT and ACCEPT

ii) HALT and REJECT

iii) LOOP (the machine fails to HALT).

Turing Machine (Formal Definition):

A turing machine can be de defined as a set of 7 tuples $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where;

$Q \rightarrow$ Finite set of states.

$\Sigma \rightarrow$ Finite set of Input Symbols.

$\Gamma \rightarrow$ Finite set of Tape Symbols.

$\delta \rightarrow$ Transition function defined as;

$$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$$

i.e, when we are on a particular state, on getting a particular input symbol, we write something on the tape sequence and we move right or left on the tape and then we go to the next state.

$q_0 \rightarrow$ Initial state

$B \rightarrow$ Blank Symbol

$F \rightarrow$ Set of Final States (Accept State & Reject state).

Thus, the production rule of turing machine will be written as;

e.g. $\delta(q_0, a) \rightarrow (q_1, y, R)$.

i.e, Initially we are on q_0 state, on getting particular input a, we go to another state q_1 writing the symbol y on tape sequence and move right on tape head

⊛. Instantaneous Description for TM:

A configuration of TM is described by Instantaneous description (ID) of TM as like PDA. A string $x_1 x_2 \ldots\ldots x_{i-1} q x_i x_{i+1} \ldots\ldots x_n$ represents the I.D. of TM in which;

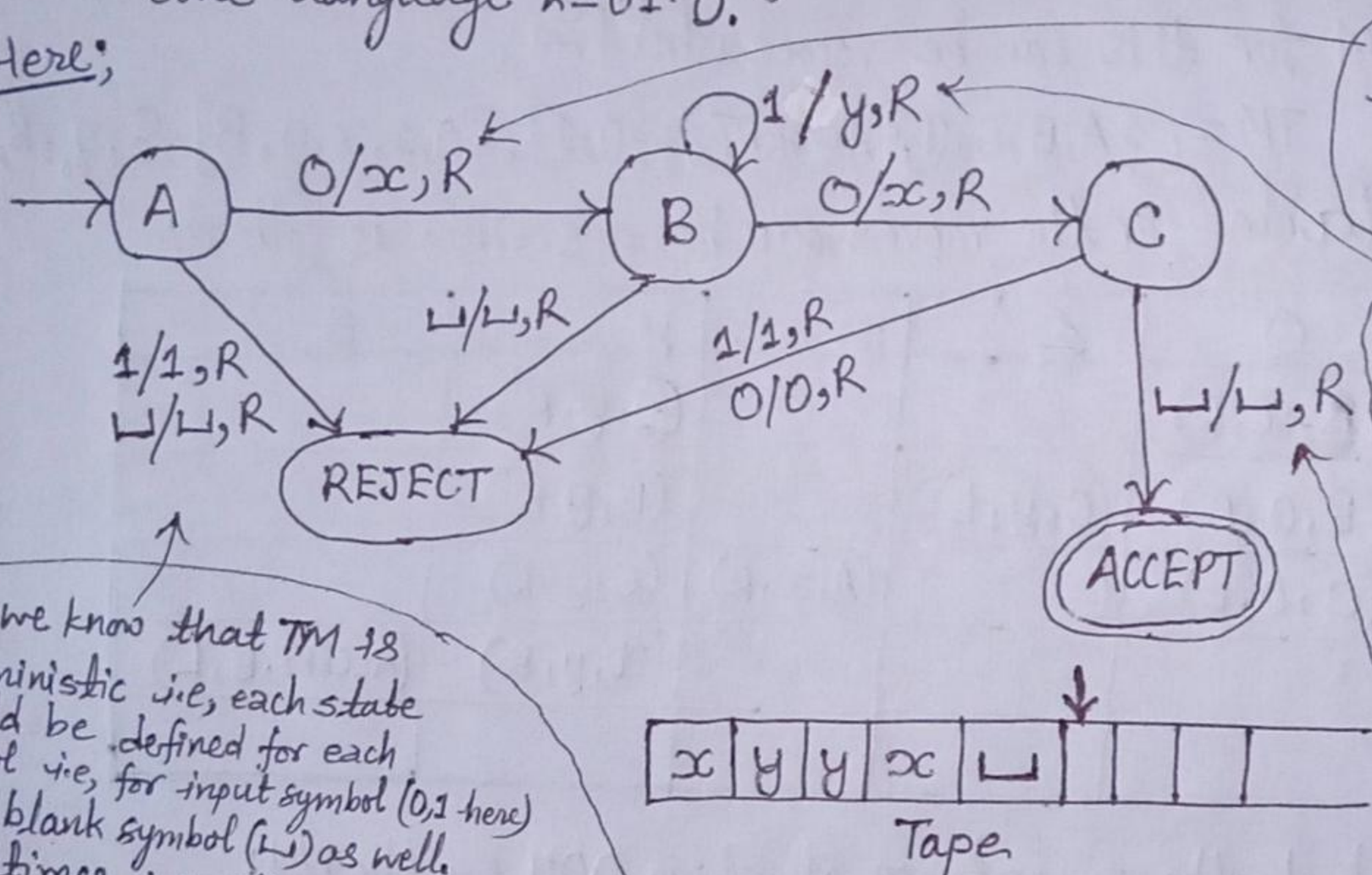
→ q is the state of TM.

→ the tape head scanning the i^{th} symbol from the left.

→ $x_1 x_2 \ldots\ldots x_n$ is the portion of tape between the leftmost and rightmost non-blank.

Example 1: Let we design a Turing Machine (TM) which recognizes the language $L = 01^*0$.

Here;



i.e, State A on getting input 0 goes to state B writing x on tape and moving arrow right similarly for others.

i.e, B state on getting any number of 1's should remain in state B writing y in ~~each~~ tape each time and moving tape arrow right each time.

It is similar to NFA with additional tape symbol and move and seperate accept state

i.e, C state on getting blank symbol ␣ goes to accept State writing same blank symbol (becoz we do not need it to update) and moving tape arrow right.

as we know that TM is deterministic i.e, each state should be defined for each symbol i.e, for input symbol (0,1 here) and blank symbol (␣) as well. Sometimes we will see Reject State and transitions to it may be missing, then do not think it as wrong. It is right because, in any TM if any transition is missing then it goes to Reject state by default.

Example 2: Design a TM which recognizes the language $L = 0^N 1^N$.
(i.e, the no. of 0's must be equal to no. of 1's & first 0's should occur then 1's).

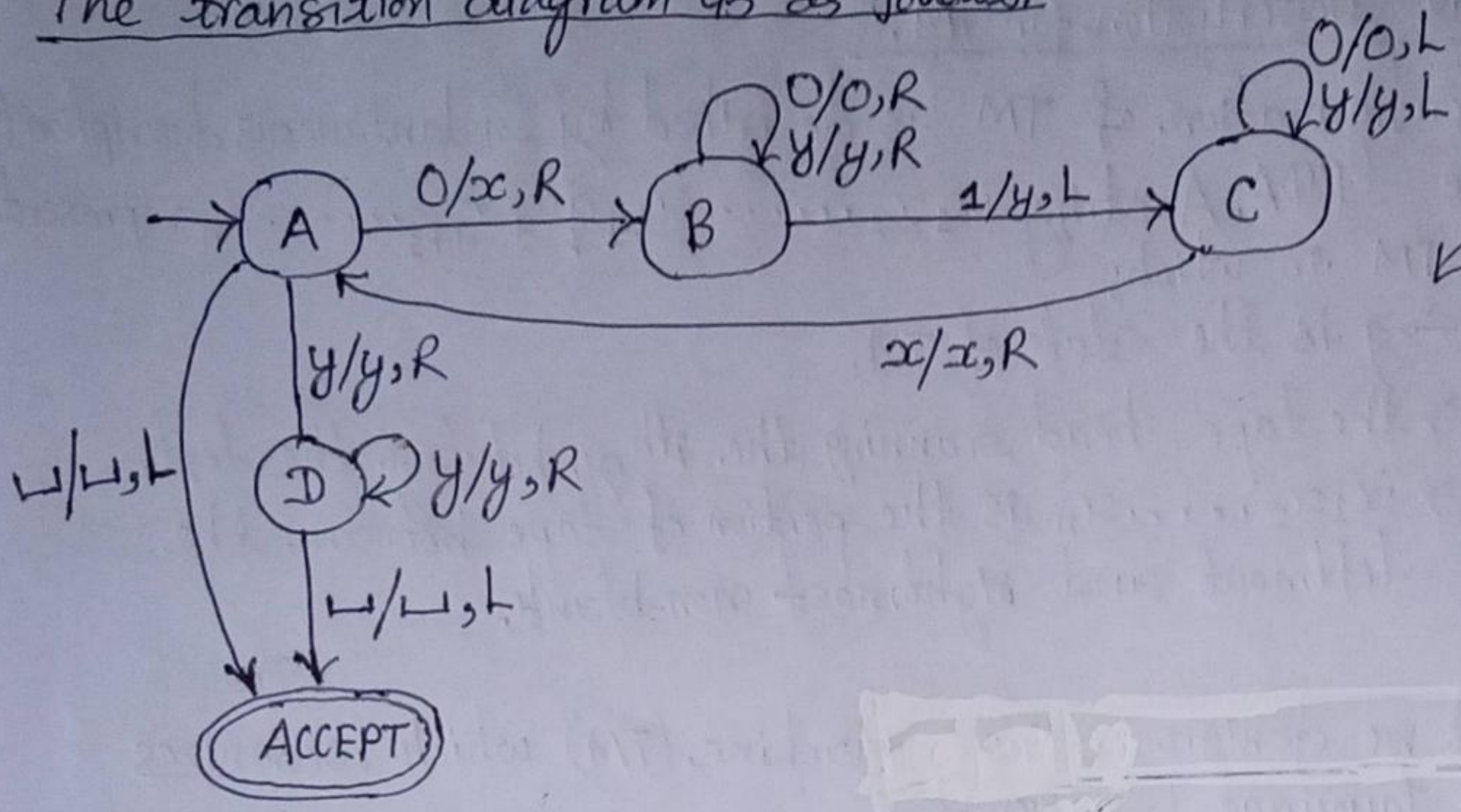
Solution:

It can be designed with the concept of algorithm as below:

→ Change "0" to "x"
→ Move RIGHT until we get first "1"
If None: REJECT
→ Change "1" to "y"
→ Repeat the above all steps until no more "0"s.
→ Make sure no more "1"s remain.

this algorithm will help to understand how the TM can be constructed. If difficult to understand refer video by neso academy TM example-2

The transition diagram is as follows:-



Here we see no reject state and some transitions missing. This means missing some transitions by default goes to Reject state so no need to draw

Now the TM for this can be represented as;

$TM = (\{A, B, C, D, ACCEPT\}, \{0, 1\}, \{0, 1, x, y, B\}, \delta, q_0, B, \{ACCEPT\})$.

Transition table for the moves can be described as follows:-

	0	1	x	y	B
A	(B, x, R)			(D, y, R)	
B	(B, 0, R)	(C, y, L)		(B, y, R)	
C	(C, 0, L)		(A, x, R)	(C, y, L)	
D				(D, y, R)	(ACCEPT, B, L)
ACCEPT					

Now, let we check the acceptance of string 0011 by the TM:

start state
given string

A 0011 ⊢ x B 011
⊢ x 0 B 11
⊢ x C 0 y 1
⊢ x A 0 y 1
⊢ x x B y 1
⊢ x x y B 1
⊢ x x C y y
⊢ x C x y y
⊢ x x A y y
⊢ x x y D y
⊢ x x y y D B
⊢ x x y y B ACCEPT B

Hence, Halt and accept.

e.g. मा accepting state को नाम ACCEPT राखेका छैन त्यसैले यो ACCEPT भनेको A,B,C जस्ते रहेछ state को नाम हो। If accept state हामिले E राखेको भए माथि यो लाइन x x y y B E B हुन्थ्यो।

Let similarly we check the acceptance for string 0110:

A 0110 ⊢ xB110
⊢ Cxy10
⊢ xAy10
⊢ xyD10

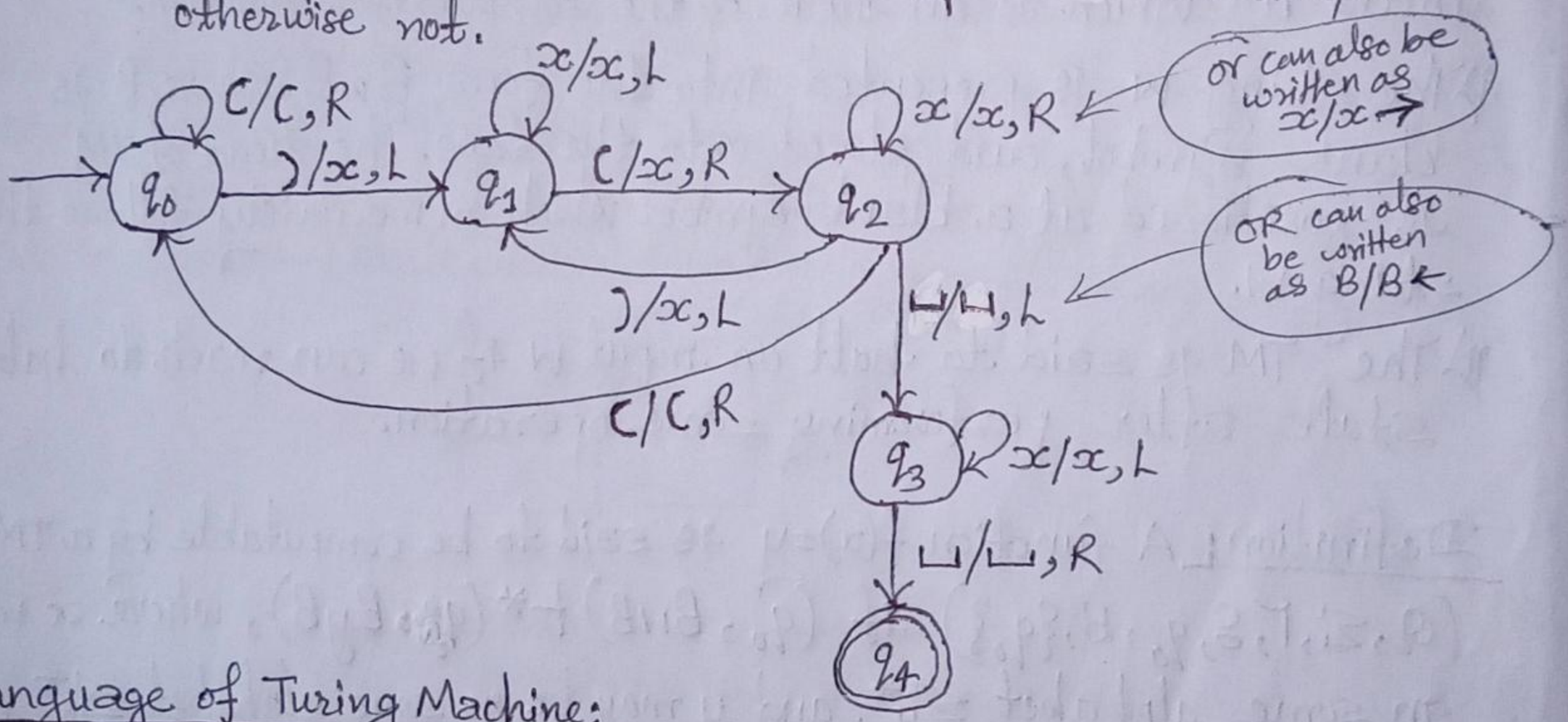
Halt and reject, since D has no move on symbol 1.

Example 3: Design a TM that accepts well formed string of parenthesis.
i.e, L = {(0), 0(0), (0)0, ...}.

Solution:

Idea:

→ Find the first (, then replace it with x and find it's corresponding) and replace it with x.
→ Perform the above step until all the symbols are scanned.
→ Finally if the tape consists of only x or ␣ then accept otherwise not.



⊛ Language of Turing Machine:

If T = (Q, Σ, Γ, δ, q0, B, F) is a turing machine and w ∈ Σ*, then language accepted by T, L(T) = {w | w ∈ Σ* and q0w ⊢* ∝ p β} for some p ∈ F and any type string ∝ and β.

The set of languages that can be accepted using TM are called recursively enumerable languages.

Roles of TM:

→ As a language recognizer
→ As a computer of function.
→ As a enumerator of string of a language.

⊛. Turing Machine as a language Recognizer:

A Turing Machine can be used as a languge recognizer to accept strings of certain language.

For Example:- A TM accepting $\{0^n1^n \mid n \geq 1\}$ as we discussed earlier.

A Turing Machine T recognizes a string x (over Σ) if and only if when T starts in the initial position and x is written on the tape, T halts in a final state. A Turing Machine T does not recognize a string x, if T does not halt in a state that is not final.

⊛Turing Machine as a Computing a Function:-

A Turing Machine can be used to compute functions. For such TM, we adopt the following policy to input any string to the TM which is an input of the computation function.

i) The string W is presented into the form BwB, where B is blank symbol, and placed onto the tape. The head of TM is positioned at a blank symbol which immediately follows the string W.

ii) The TM is said to halt on input W if we can reach to halting state after performing some operation.

Definition: A function $f(x)=y$ is said to be computable by a TM $(Q, \Sigma, \Gamma, \delta, q_0, B, \{q_f\})$ if $(q_0, BwB) \vdash^* (q_f, ByB')$, where x may be in some alphabet Σ_1^*. and y may be in some alphabet Σ_2^* and $\Sigma_1, \Sigma_2 \subseteq \Sigma$.

Example: Design a TM which compute the function $f(x)=x+1$ for each x that belongs to the set of natural numbers.

Solution:

Given function $f(x) = x+1$.

Here, we represent input x on the tape by a number of 1's on the tape. For example, for $x=1$, input will be B1B.

for $x=2$, input will be B11B.

for $x=3$, input will be B111B and so on.

Similarly output can be seen by the number of 1s on the tape when machine halts.

Let TM = $(Q, \Sigma, \Gamma, \delta, q_0, B, \{q_f\})$; where $Q = \{q_0, q_f\}$, $\Gamma = \{1, B\}$ and halt state = $\{q_0\}$. Then, δ can be simulated as;

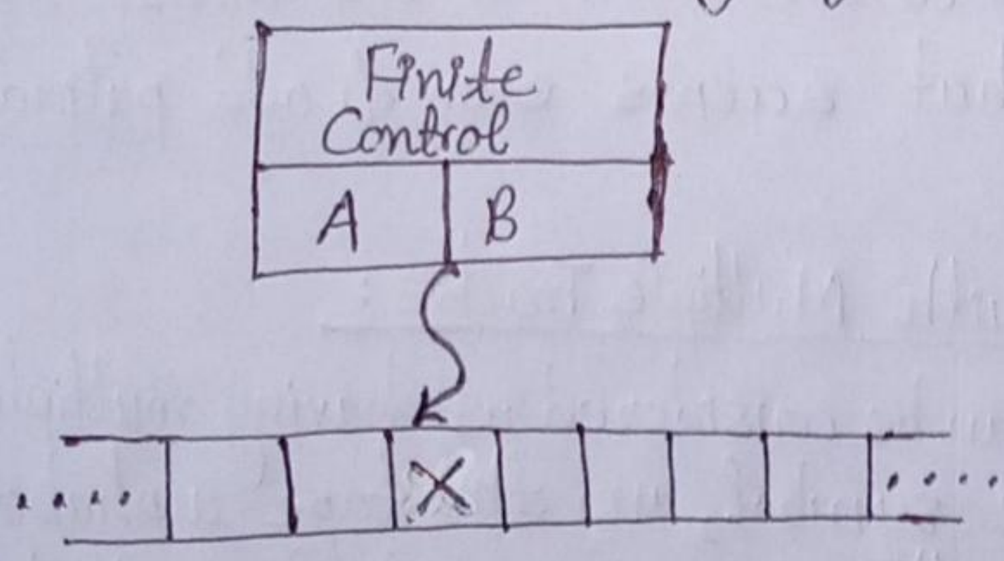
Q	B	1
q_0	$(q_0, 1, S)$	$(q_f, 1, R)$
q_f	–	–

So, let the input be x=4. So, input tape at initial step consists of B1111B.

Then $(q_0, B1111B) \vdash (q_0, B11111) \vdash (q_f, B11111B)$. Which means output 5 is accepted.

⊛ Turing Machine with Storage in the state:

In turing machine, any state represents the position in the computation. But a state can also be used to hold a finite amount of data. The finite control of machine consists of a state q and some data portion. In this case a state is considered as a tuple – (state, data). Following figure illustrates the model.



δ is defined by:

$\delta([q, A], X) = ([q_1, X], Y, R)$ means q is the state and data portion of q is A. The symbol scanned on the tape is copied into second component of the state and moves right entering state q_1 and replacing tape symbol by Y.

Example:- This model of TM can be used to recognize languages like $01^* + 10^*$, where first symbol (0 or 1) that it sees, and checks that it does not appear else where in the input. For this, it remembers the first symbol in finite control.

⊛ Turing Machine as Enumerator of Strings of Language:

Consider a multi-tape turing machine TM that uses a tape as an output tape, in which a symbol once written can never be changed, and those whose tape head never moves left. Suppose also that on the output tape, TM writes strings over some alphabet Σ separated by a marker symbol #.

We can define L(TM) to be set of w in Σ^* such that w is eventually printed between a pair of #'s on the output tape. Thus the language of this type of Turing machine is called Turing Enumerable languages.

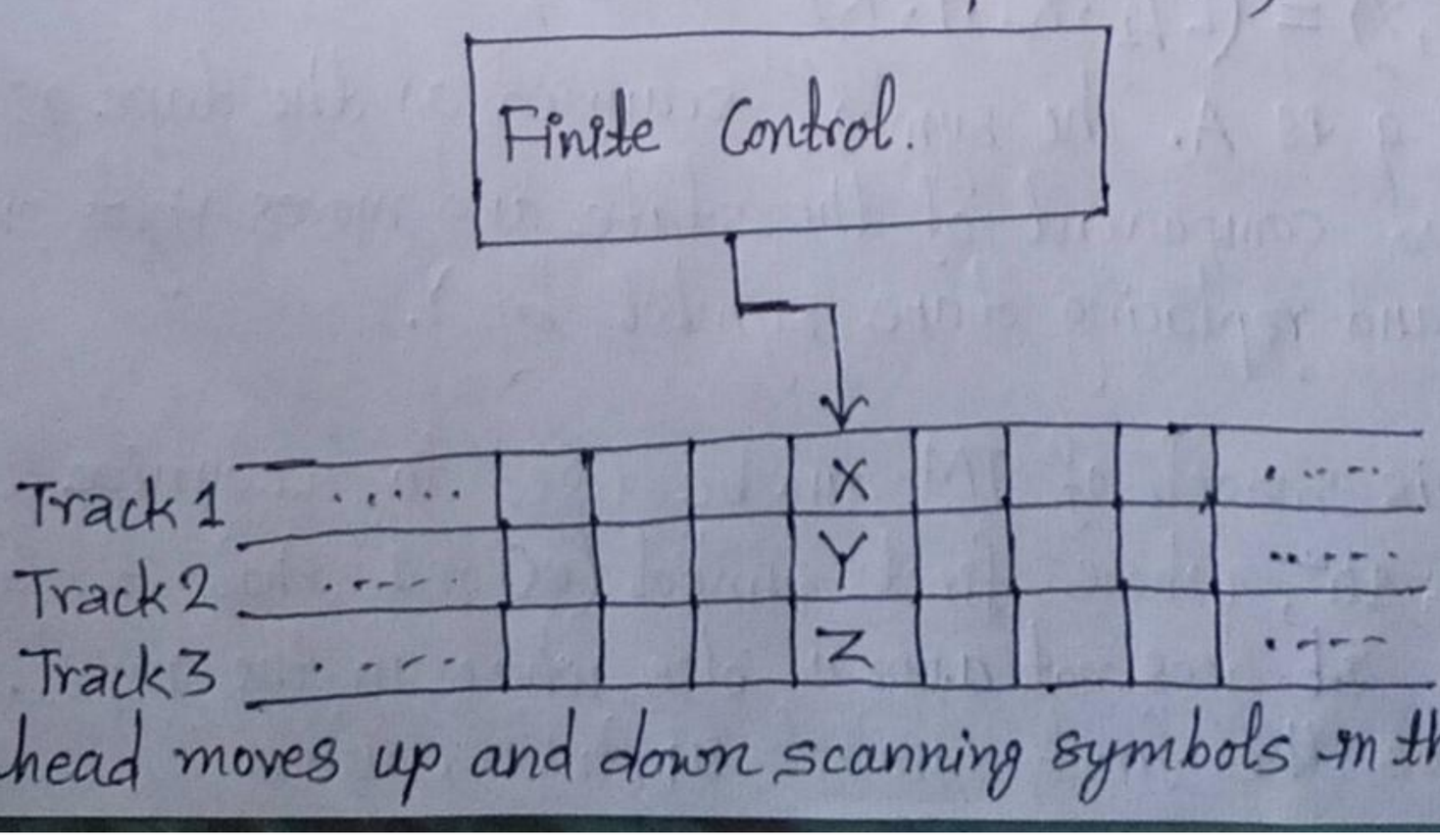
⊛. Turing Machine as Subroutine:

Subroutines are collection of interacting components. A TM subroutine is set of states that perform some useful process. This set of states includes a start state and other states that serves as the return to pass control to other set of states called subroutine. The "call" of a subroutine occurs whenever there is a transition to its initial state.

Example: A TM that accepts even & odd palindrome.

⊛ Turing Machine with Multiple Tracks:

The tape of TM can be considered as having multiple tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples, with one component for each track. Following figure illustrates the TM with multiple tracks;



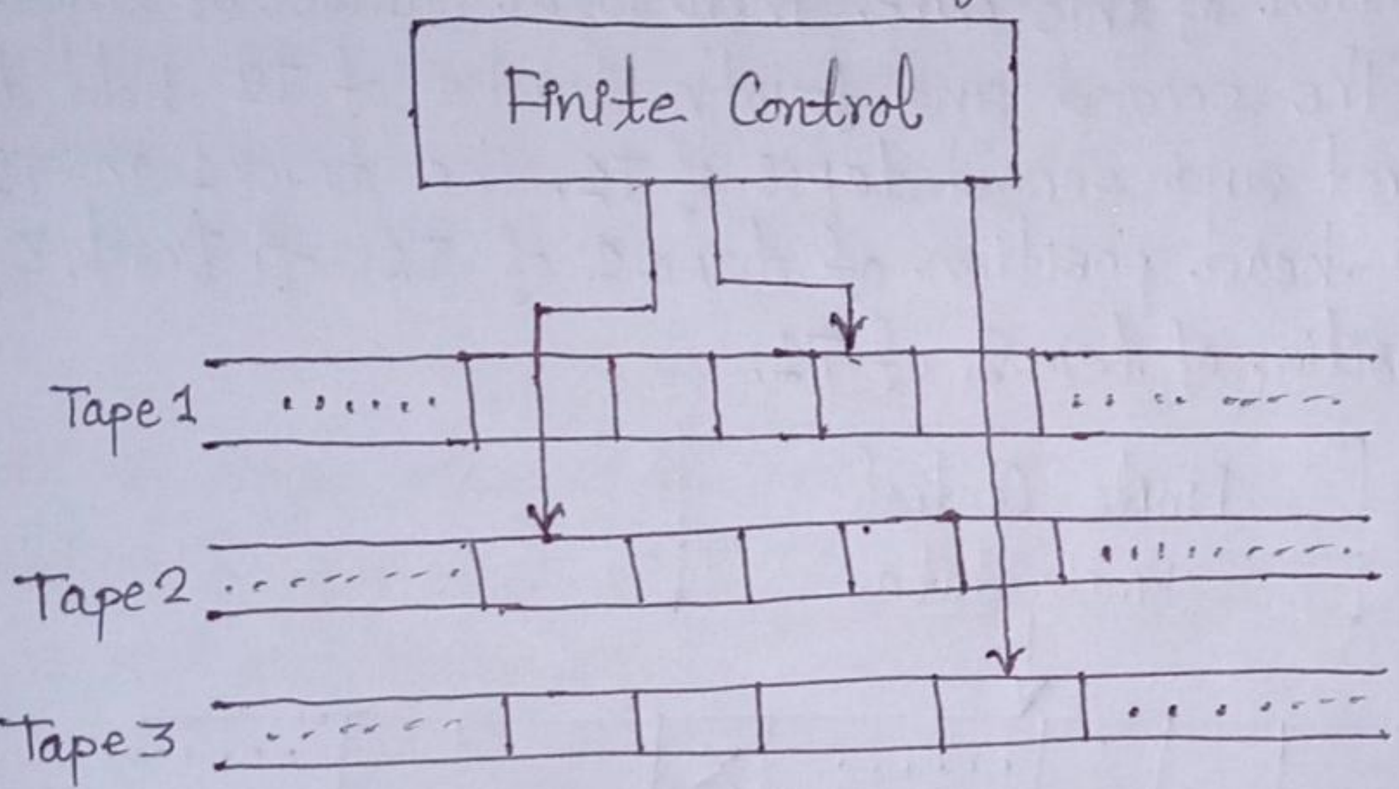
The tape head moves up and down scanning symbols in the tape at one position.

The tape alphabet Γ is a set consisting tuples like,

$$\Gamma = \{(X, Y, Z), \ldots\}.$$

⊛. Turing Machine with Multiple Tapes:

A TM can be multi-tape in which there is more ~~one~~ than one tape. Adding extra tape adds no power to the computational model, but only the ability to accept the language is increased. A multi-tape TM consists of finite control and finite number of tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabets. The set of tape symbols include a blank and the input symbols.



In the multi-tape TM, initially;

→ The Input (finite sequence of input symbols) w is placed on the first tape.
→ All other cells of the tapes hold blanks.
→ TM is in initial state q_0.
→ The head of the first tape is at the left end of the input.
→ All other tape heads are at some arbitrary cell.

A move of multi-tape TM depends on the state and the symbol scanned by each of the tape head. In one move, the multi-tape TM does following;

→ The control enters in a new state, which may also be same previous state.
→ On each step, a new symbol scanned is written on the cell, these symbols may be same as the symbols previously there.
→ Each of the tape head make a move either left or right or remains stationary. Different head may move different direction independently.

⊛ Equivalence of One-tape and Muti-tape TM's:

OR simulating one tape TM with multi-tape TM

Theorem: Every language accepted by a multitape TM is recursively enumerable.

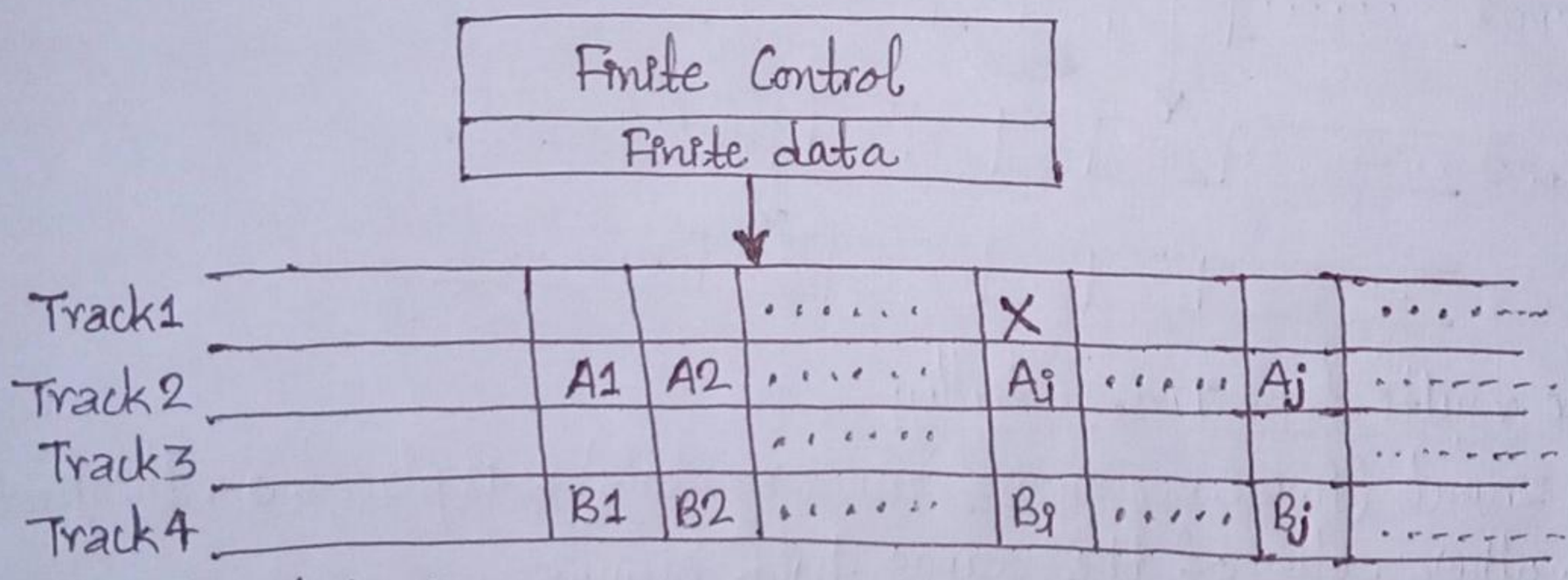
OR Any languages that are accepted by a multi-tape TM are also accepted by one tape Turing machine.

Proof:

T1 denotes a name of TM with n-tape

Let L is a language accepted by a n-tape TM, T1. Now, we have to simulate T1 with a one-tape TM, T2 considering there are 2n tracks in the tape of T2.

For simplicity, let us assume $n=2$, then for $n>2$ is the generalization of this case. Then total number of tracks in T2 will be 4. The second and fourth tracks of T2 hold the contents of first and second tapes of T1. The track1 in T2 holds ~~the contents of~~ head position of tape 1 of T1 & track 3 in T2 holds head position of tape 2 of T1.



Now, to simulate the move of T1,

→ T2's head must visit the n-head markers so that it must remember how many head markers are to it's left at all times. That count is stored as a component of T2's finite control.

→ After visiting and storing scanned symbol, T2 knows what tape symbols are being scanned by each of T1's head.

→ T2 also knows the state of T1, which it stores in T2's own finite control. Thus T2 knows what move T1 will make.

→ T2 now revisits each of the head markers on its tape, changes the symbol in track representing corresponding tapes T1 and moves the head marker left or right if necessary.

Finally, T2 changes the state of T1 as recorded on its own finite control. Hence T2 has simulated one move of T1. We select T2's accepting states, all those states that record T1's state as one of the accepting state of T1. Hence, whatever T1 accepts T2 also accepts.

⊛. Non-Deterministic Turing Machines:-

A non-deterministic Turing Machine (NTM) is exactly the same as ordinarly TM, except the value of transition function δ. In TM any state on getting any particular input can go to only one particular next state writing a particular symbol onto tape. But in NTM any state on getting any particular input can go to any state among multiple next states writing corresponding symbol onto tape, based on path selected.

In NTM, the values of the δ are subsets, rather than a single element of set, $Q \times \Gamma \times \{R, L, S\}$. Here the transition function δ for each state q and tape symbol x, is $\delta(q, x)$ which is a set of triples as;

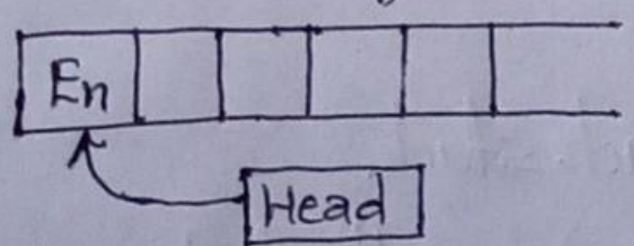
$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots\ldots, (q_k, Y_k, D_k)\}$$

where k is any finite integer.

The NTM can choose any of the triples to be the next move at each step.

⊛ Restricted Turing Machines:-

1. Semi-Infinite Tape: A Turing Machine with a semi-infinite tape has a left end but no right end. The left end is limited with an end marker.



It is a two track tape;

i) Upper track → It represents the cells to the right of the initial head position.

ii) lower track → It represents the cells to the left of the initial head position in ~~reser~~ reverse order.

The infinite length input string is initially written on the tape in contiguous tape cells.

The machine starts from the initial state q_0 and head scans from the left end. In each step, it reads the symbol on the tape under its head. It writes a new symbol on that tape cell and then it moves the head one tape cell left or right. A transition function determines the actions to be taken.

It has two special states called accept state and reject state. If at any point of time if it enters into the accept state then the input is accepted. and if it enters into the reject state then the input is rejected. In some cases it continues to run infinitely without being accepted or rejected for some certain input symbols.

2. Multi Stack Machines:

Multi Stack Machines are the generalizations of the PDA's. TM's can accept languages that are not accepted by any PDA with one stack, but PDA with two stacks can accept any language that a TM can accept.

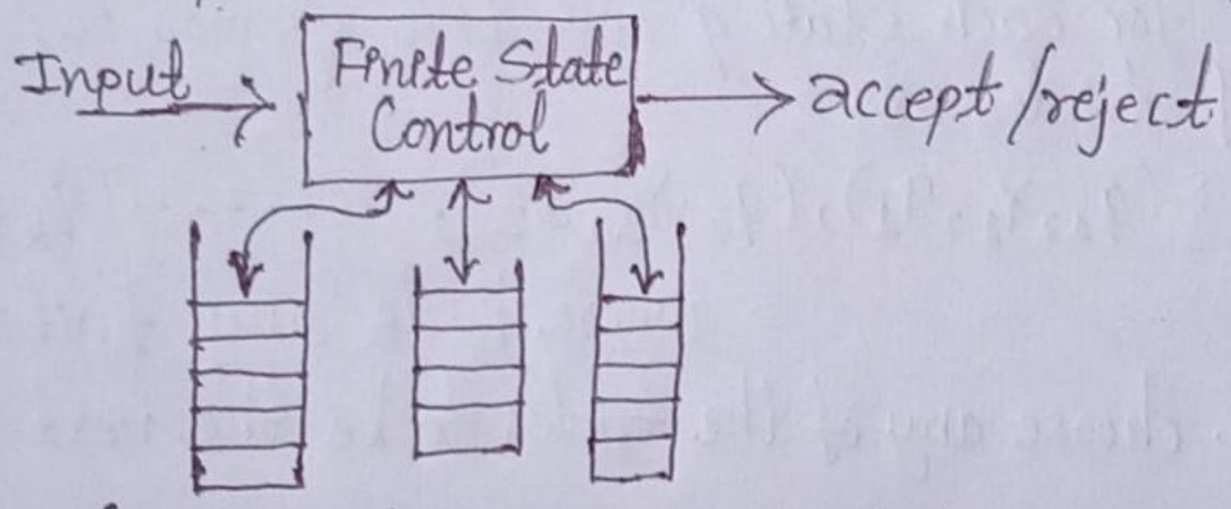


fig. A machine with three stacks.

It obtains it's input source rather than having the input placed in a tape. A move of a multistack machine is based on;

→ The state of finite control.
→ The input symbol read.
→ The top stack symbol on each stack.

In one move a multistack machine can change to a new state and can replace the top symbol of each stack with a string of zero or more stack symbols.

3. Counter Machines:

Counter Machine are offline TM's whose storage tapes are semi-infinite and whose tape alphabets contain only two symbols Z and B. The symbol Z serves as a bottom of stack marker appearing initially on the cell scanned by the tape head and may never appear on any other cell. B is the Blank Symbol.

A stored number can be incremented or decremented by moving the tape head left or right. We can test whether a number is zero, but cannot directly test whether two numbers are equal. Instantaneous description of a counter machine can be described by the state, the input tape contents, the position of input head, and the distance of the storage heads from the symbol Z.

⊛. Church. Turing Thesis:

It is a mathematically un-provable hypothesis about the computability. This hypothesis simply states that - "Any algorithmic procedure that can be carried out at all can be carried out by a TM." This statement was first formulated by Alonzo Church a mathematician and a logician in 1930s. According to Church, "No computational procedure will be considered an algorithm unless it can be represented by a Turing Machine".

After adopting Church-Turing Thesis, we are giving precise meaning of the term: An algorithm is a procedure that can be executed on a Turing Machine. Another use of Church-Thesis is that, when we want to describe a solution to a problem, we will often satisfy with a verbal description of the algorithm, translating it into detailed Turing Machine Implementations.

⊛. Universal Turing Machine:

A machine that can simulate the behaviour of an arbitrary TM is called a Universal Turing Machine. Thus, we can describe a Universal Turing Machine T_u as a TM, that on input $\langle M, w \rangle$; where M is a TM and w is a string of input alphabets, simulates the computation of M on input w. and satisfies two properties;

i) Tu accepts $\langle M, w \rangle$ iff M accepts w.
ii) Tu rejects $\langle M, w \rangle$ iff M rejects w.

⊛. Encoding of Turing Machine:-

For the representation of any arbitrary TM T_1, and an input string w over as arbitrary alphabet, as binary strings $e(T_1)$, $e(w)$ over some fixed alphabet, a notational system should be formulated. Encoding the TM T_1, and input w into $e(T_1)$ and $e(w)$, it must not destroy any information. For encoding of TM, we use alphabet $\{0,1\}$, although the TM may have a much larger alphabet.

To represent a TM $T_1 = (Q_1, \{0,1\}, \Gamma, \delta, q_1, B, F)$ as binary string, we first assign integers to the states, tape symbols and directions. We assume two fixed infinite sets $Q = \{q_1, q_2, q_3, \ldots\}$ and $S = \{a_1, a_2, a_3, \ldots\}$ so that Q_1 is subset of Q and Γ is subset of S. Now we have a subscript attached to every possible state and tape symbols, we can represent a state or a symbol by a string of 0's of the appropriate length. Here, 1's are used as seperators.

Once we have established an integer to represent each state, symbol and direction, we can encode the transition function δ. Let one transition rule is:

$$\delta(q_i, a_j) = (q_k, a_l, D_m)$$ for some integer i, j, k, l, m.

Then we shall code this rule by the string $s(q_i)1s(a_j)1s(q_k)1s(a_j)1s(D_m)$, say this as m1, where s is the encoding function defined below. A code for entire TM T1 consist of all the codes for the transitions, in some order, seperated by pairs of 1's:

$$m_1 11 m_2 11 \ldots m_n.$$

Now the code for TM and input string w will be formed by separating them by three consecutive ones i.e, 111.

The Encoding function s:-

First, associate a string of 0's, to each state, to each of the three directions, and to each tape symbol. Let the function s is defined as;

$S(B) = 0$

$S(a_i) = 0^{i+1}$ for each $a_i \in S$.

$S(q_i) = 0^{i+2}$ for each $q_i \in Q$.

$S(S) = 0$

$S(L) = 00$

$S(R) = 000$.

Example: Consider an example where, TM T is defined as;

$T = (\{q_1, q_2, q_3\}, \{a, b\}, \{a, b, B\}, \delta, q_1, B, F)$, where δ is defined as;

$\delta(q_1, b) = (q_3, a, R) \rightarrow m_1$

$\delta(q_3, a) = (q_1, b, R) \rightarrow m_2$

$\delta(q_3, b) = (q_2, a, R) \rightarrow m_3$

$\delta(q_3, B) = (q_3, b, L) \rightarrow m_4$.

Now, using the encoding function s defined above, as the rule, we have

$S(q_1) = 000$

$S(q_2) = 0000$

$S(q_3) = 00000$

$S(a_1) = 00$

$S(a_2) = 000$

$S(B) = 0$

$S(R) = 000$

$S(L) = 00$

$S(S) = 0$

Considering $a_1 = a$ & $a_2 = b$.

Now, encoding for rules;

$e(m_1) = S(q_1)1S(b)1S(q_3)1S(a)1S(R)$

$= 000100010000010010000$

$e(m_2) = S(q_3)1S(a)1S(q_1)1S(b)1S(R)$

$= 000001001000100010000$

$e(m_3) = S(q_3)1S(b)1S(q_2)1S(a)1S(R)$

$= 000001000100001001000$

$e(m_4) = S(q_3)1S(B)1S(q_3)1S(b)1S(L)$

$= 000001010000010000100$

Now, the code for TM T is $e(m_1)11\,e(m_2)11\,e(m_3)11\,e(m_4)$.

i.e, 000100010000010010001100000100100010000100011000

1000100001001000110000010100000100000100.

For this machine T, for any input w, where w=ab, the code will be e(T)111e(w), where e(w)= s(a)1s(b)
= 001000.

⊛. Turing Machines and Computers:

The Turing Machines and Computers both can accept the same, recursively enumerable languages. Since the notation of "a common computer" is not well defined mathematically, the arguments in this are necessarily informal. Turing Machines and Computers can be divided into two parts as follows:

i) A computer can simulate a Turing Machine.

ii) A Turing machine can simulate a computer, and can do so in an amount of time that is at most some polynomial in the number of steps taken by the computer.

⊛. Difference between TM and Other Automata (FSA and PDA):

The most significant difference between the TM and the simpler machine (FSA and PDA) is that; in a TM, processing a string is no longer restricted to a single left to right pass through input. The tape head can move in both directions and erase or modify any symbol it encounters. The machine can examine part of the input, modify it, take time execute some computation in a different area of the tape, return to re-examine the input, repeat any of these actions and perhaps stop the processing before it has locked at all input.

⊛ Enumerating the Binary Strings:-

We shall need to assign integers to all the binary strings so that each string corresponds to one integer, and each integer corresponds to one string. If w is a binary string, then treat 1w as a binary integer i. Then we shall call w the ith string. That is ε is the first string, 0 is the second, 1 is the third, 00 the fourth, 01 the fifth and so on. Equivalently, strings are ordered by length, and strings of equal length are ordered lexicographically. Hereafter, we shall refer to the ith string as w_i.