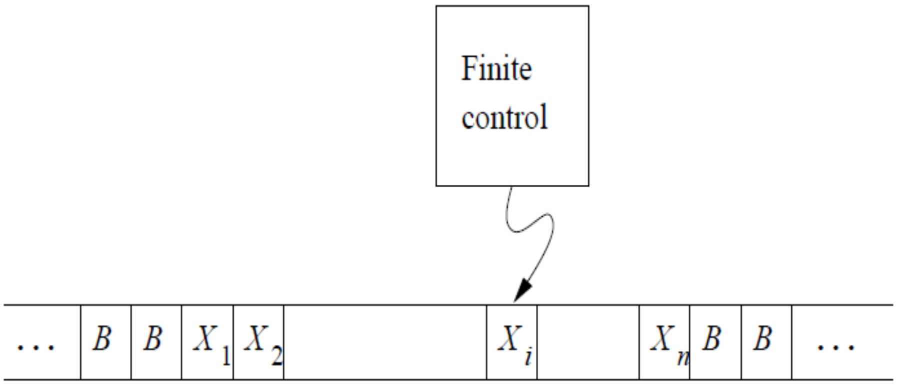
UNIT-V:TURING MACHINES

A Turing machine is an abstract computing machine with the power of real computers. A Turing machine can be visualized as shown in the following figure.



The Turing machine consists of

1. a **finite state control** which can be in any of a finite set of states and

2. an **Infinite tape** divided into cells where each cell holds one of a finite number of tape symbols (including blank symbol)

3. **Read/Write tape head** which can examine one cell at a time.

The Turing machine makes move based on

a) Its current state

b) The tape symbol at the cell scanned by the tape head.

In one move, the Turing machine can

1. Change the state

2. Overwrite the scanned cell with some tape symbol

3. Move the tape head one cell left or right.

Mathematically, a Turing Machine is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where:

- Q is the finite set of *states* of the finite control.
- Σ is the finite set of *input symbols.*
- Γ is the finite set of *tape symbols*; $\Sigma \subset \Gamma$.
- $q_0 \in Q$ is the *start state.*
- $B \in \Gamma$ is the *blank symbol*; $B \notin \Sigma$.
- $F \subseteq Q$ is the set of *final* or *accepting* states.
- δ is a transition function defined as follows.

The arguments of $\delta(q, X)$ are a state q and a tape symbol X. The value of $\delta(q, X)$, if it is defined, is a triple (p, Y, D), where:

1. p is the next state, in Q.
2. Y is the symbol, in Γ, written in the cell being scanned, replacing whatever symbol was there.
3. D is a *direction*, either L or R, standing for "left" or "right," respectively, and telling us the direction in which the head moves.

Representation of a Turing Machine:

A Turing machine can be described by means of a

a) Transition diagram

b) Transition table

c) Instantaneous descriptions using move relations

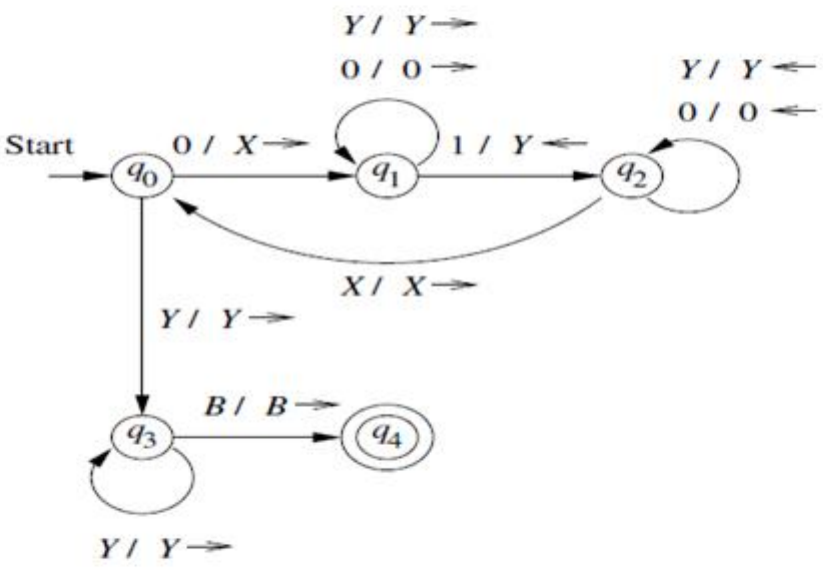
a) Transition diagram:

A transition diagram consists of a set of nodes corresponding to the states of the Turing machine.

An arc from state q to state p is labeled by one or more items of the form X / YD, where X and Y are tape symbols and D is the direction, either L or R. (we can use left arrow $\longleftarrow$ for L and right arrow $\rightarrow$ for R)

So whenever $\delta(q,X) = (p,Y,D)$, there is an arc from state q to state p with the label X / YD.

Transition diagram for the Turing machine that accepts the Language L={0^n1^n / n≥1}



b) Transition Table:

The transition function δ of a Turing Machine can also be represented by a table known as transition table.

ex: The transition table of a Turing machine that accepts the Language L={0^n1^n / n≥1}

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1,X,Y,B\}, \delta, q_0, B, \{q_4\})$$

State	Symbol				
	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

C) Instantaneous Descriptions:

A Turing machine changes its configuration upon each move.

We use instantaneous descriptions (IDs) for describing such configurations.

An *instantaneous description* is a string of the form

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$$

where

1. q is the state of the Turing machine.
2. The tape head is scanning the ith symbol from the left.
3. $X_1X_2\cdots X_n$ is the portion of the tape between the leftmost and rightmost nonblanks.

We use $\underset{M}{\vdash}$ to designate a move of a Turing machine M from one ID to another.

If $\delta(q, X_i) = (p, Y, L)$ then:

$$X_1X_2\ldots X_{i-1}\ q\ X_iX_{i+1}\ldots X_n \vdash_M X_1X_2\ldots X_{i-2}\ p\ X_{i-1}\ Y\ X_{i+1}\ldots X_n$$

Now, suppose $\delta(q, X_i) = (p, Y, R)$; i.e., the next move is rightward. Then

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \underset{M}{\vdash} X_1X_2\cdots X_{i-1}YpX_{i+1}\cdots X_n$$

Ex:

For the string 0011, initial ID is $q_0$0011

The entire sequence of moves of M is:

$$q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash q_2 X0Y1 \vdash$$
$$Xq_0 0Y1 \vdash XXq_1 Y1 \vdash XXYq_1 1 \vdash XXq_2 YY \vdash Xq_2 XYY \vdash$$
$$XXq_0 YY \vdash XXYq_3 Y \vdash XXYYq_3 B \vdash XXYYBq_4 B$$

Ex2: for the string 0010, the sequence of moves of M is given as follows.

$$q_0 0010 \vdash Xq_1 010 \vdash X0q_1 10 \vdash Xq_2 0Y0 \vdash q_2 X0Y0 \vdash$$
$$Xq_0 0Y0 \vdash XXq_1 Y0 \vdash XXYq_1 0 \vdash XXY0q_1 B$$

The language of a TM:

Let M = (Q, Σ, Γ, δ, q0, B, F) be a TM. Then L(M) is the set of strings w ∈ Σ^* such that

$$q_0\, w \overset{*}{\vdash} \alpha\, p\, \beta$$

for some state p ∈ F and any tape string α and β

Note:

1. We always assume that a Turing Machine halts if it accepts
2. The set of languages we can accept using a TM is often called the recursively enumerable languages or RE languages.
3. There is another notion of acceptance that is commonly used for TM: **acceptance by halting**

Acceptance by Halting:

We say a TM halts if it enters a state q scanning a tape symbol X, and there is no move in this situation, i.e., δ(q, X) is undefined.

We can always assume that a TM halts if it accepts. Unfortunately, it is not always possible to require that a TM halts even if it does not accept

Note:

1. Recursive Languages:

Languages for which TM do halt, regardless of whether or not they accept are called as recursive Languages

2. If an algorithm to solve a given problem exists, then we say the problem is decidable.

Problems:

Design a Turing machine that recognizes the language

$$L= \{a^n b^n c^n / n \geq 1\}$$

Sol:

Logic:

First replace ‘a’ from front by X, then keep moving right till you find a ‘b’ and replace this ‘b’ by Y. Again, keep moving right till you find ‘c’, replace it by Z and move left. Now keep moving left till you find a X. When you find it, move a right, then follow the same procedure as above.

A condition comes when you find a X immediately followed by a Y. At this point we keep moving right and keep on checking that all b’s and c’s have been converted to Y and Z. If not then string is not accepted. If we reach Blank ‘B’ then string is accepted.

The turing machine is given as $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

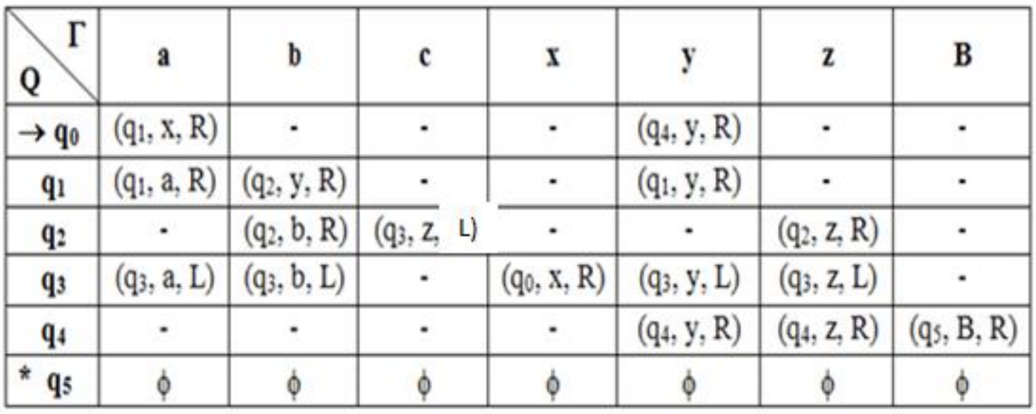
Q={q0,q1,q2,q3,q4,q5}

Σ ={a,b,c}

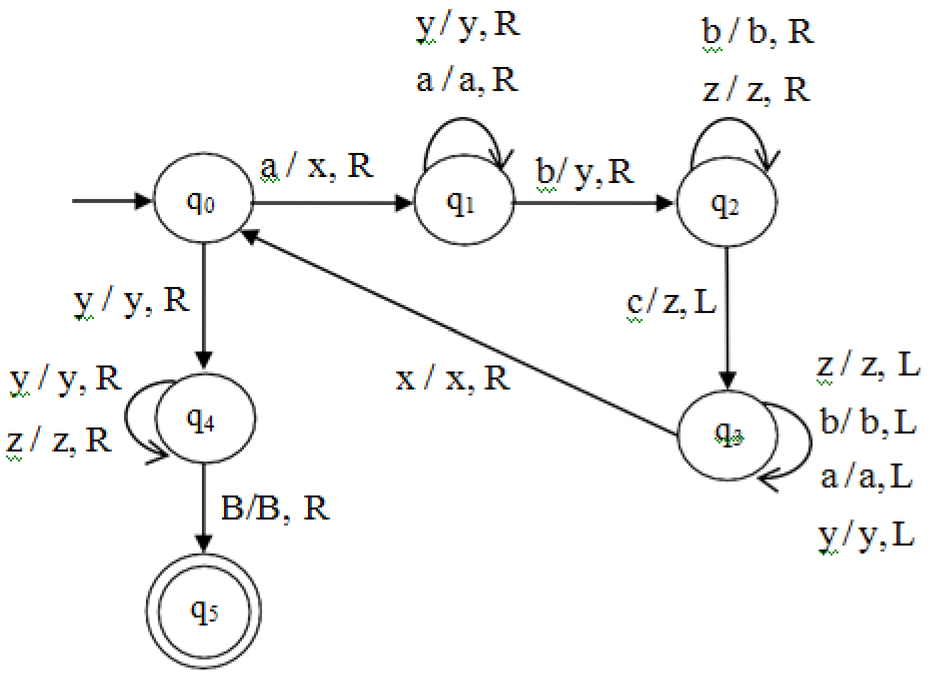
Γ={a,b,c,x,y,z,B}

F={q_5}

and δ is given as follows.



Q \ Γ	a	b	c	x	y	z	B
→ q_0	(q_1, x, R)	-	-	-	(q_4, y, R)	-	-
q_1	(q_1, a, R)	(q_2, y, R)	-	-	(q_1, y, R)	-	-
q_2	-	(q_2, b, R)	(q_3, z, L)	-	-	(q_2, z, R)	-
q_3	(q_3, a, L)	(q_3, b, L)	-	(q_0, x, R)	(q_3, y, L)	(q_3, z, L)	-
q_4	-	-	-	-	(q_4, y, R)	(q_4, z, R)	(q_5, B, R)
* q_5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ



2. Design a Turing machine that accepts the set of all palindromes over {0,1}

Sol:

The turing machine is given as M = (Q, Σ, Γ, δ, q_0, B, F) where

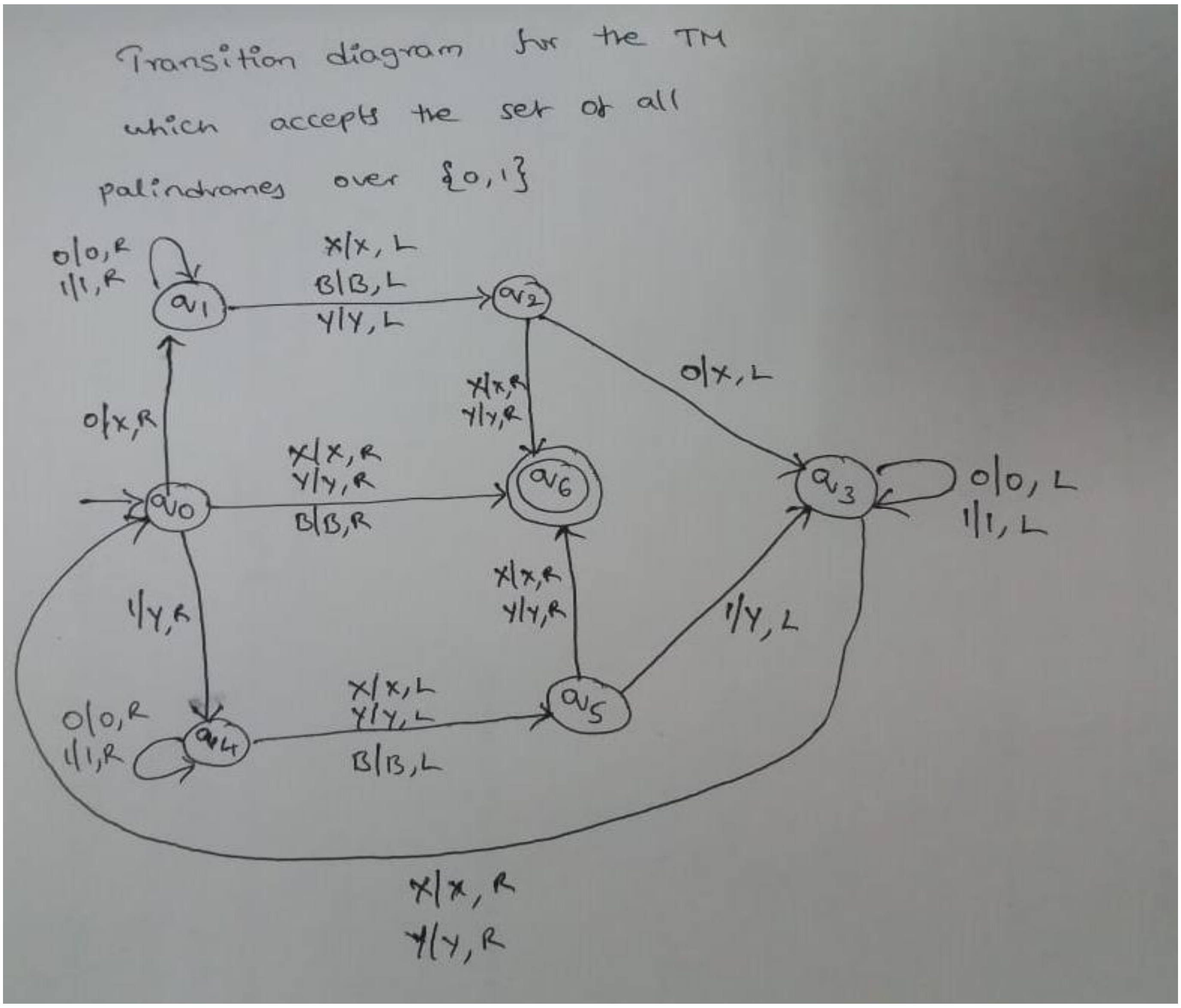
Q={q_0,q_1,q_2,q_3,q_4,q_5,q_6}

Σ ={a,b,c}

Γ={a,b,c,x,y,z,B}

F={q_6}

and δ is given as follows.



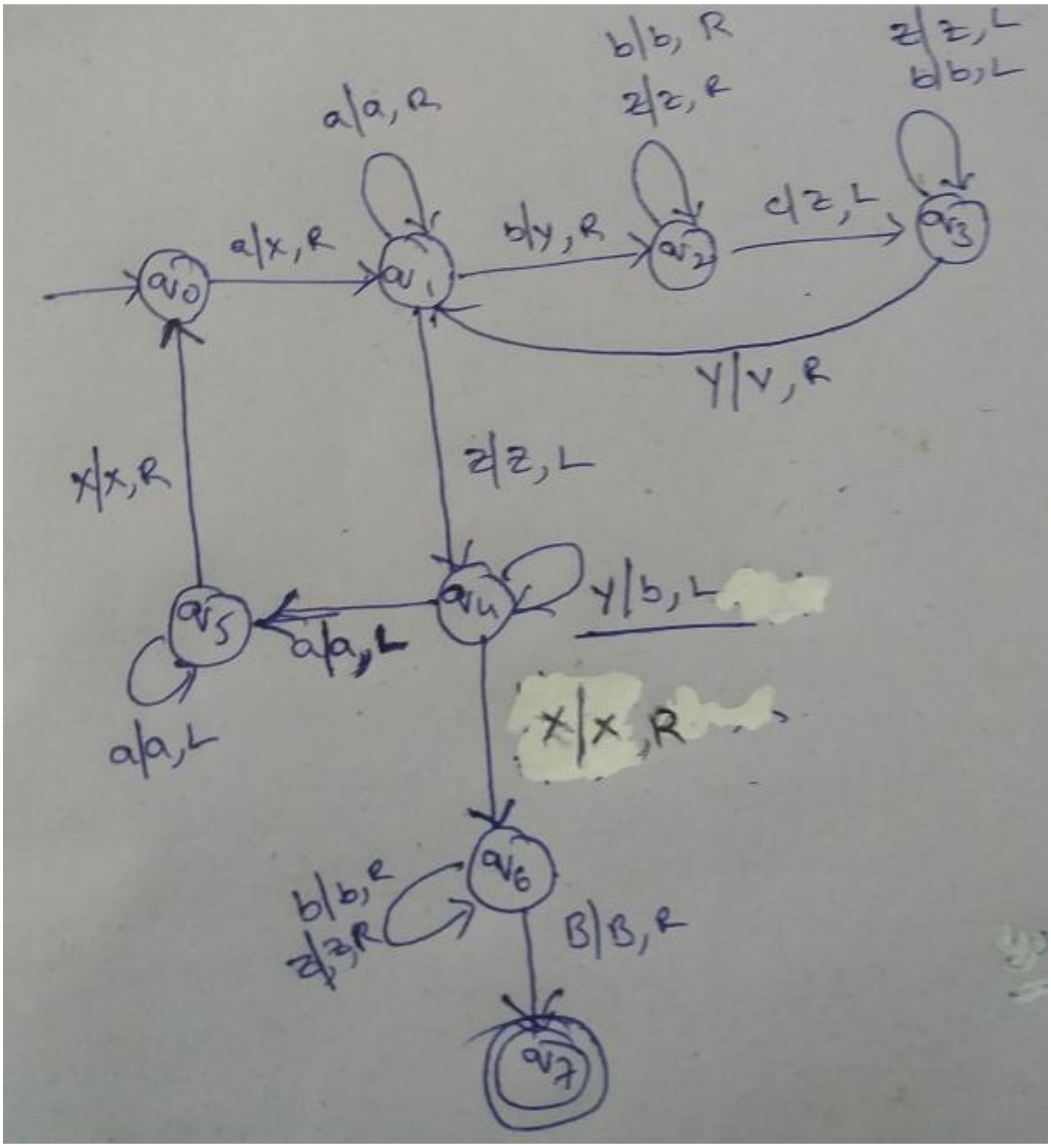
In state q0, when it encounters X, Y Or B then the palindrome is a even length palindrome. In state q2 or q5, when it encounters X or Y then the palindrome is a odd length palindrome.

In state q2, when it encounters 1 it is not a palindrome. Similarly in state q5, when it encounters 0 it is not a palindrome.

Design a turing machine that accepts L={$a^ib^jc^k$ / i*j=k, i,j,k>=1}

Sol:

The transition diagram for the turing machine is given below.



COMPUTABLE LANGUAGES AND FUNCTIONS

Turing machine is capable of performing several operations/functions. It is capable of performing any sort of computations such as

- Addition
- Subtraction
- Multiplication
- Division
- 2's complementation
- 1's complementation
- Comparing two numbers
- Squaring a number
- GCD of numbers
- Finding binary equivalents.

Representation:

Computation of a function „f" is represented as

$$f : \sum_1^* \rightarrow \sum_2^*$$

The function f is Turing computable by the machine,

$$M = (Q, \sum, \Gamma, \delta, q_0, B, F)$$

With the transition function δ of the form,

$$(q_0, \omega) \underset{M}{\overset{*}{\vdash}} (q_f, f(\omega))$$

where,

$q_0 \rightarrow$ Initial state ($q_0 \ \varepsilon \ Q$)

$\omega \rightarrow$ Input string, where $\omega \ \varepsilon \ \sum_1^*$.

$q_f \rightarrow$ Final state ($q_f \ \varepsilon \ F$)

$f(\omega) \rightarrow$ Output string after computation of „ω" by f $\left(f(\omega) \ \varepsilon \ \sum_2^* \right)$.

Representation of a number

For simplicity, the unary numbers are represented by a single symbol. Let it be "0".

The decimal numbers are represented as,

No input → B

1 → 0

2 → 00

3 → 000

4 → 0000

5 → 00000

.

.

.

n → 'n' number of zeros [0^n]

1. Design a Turing Machine to perform 2's complement of a binary number.

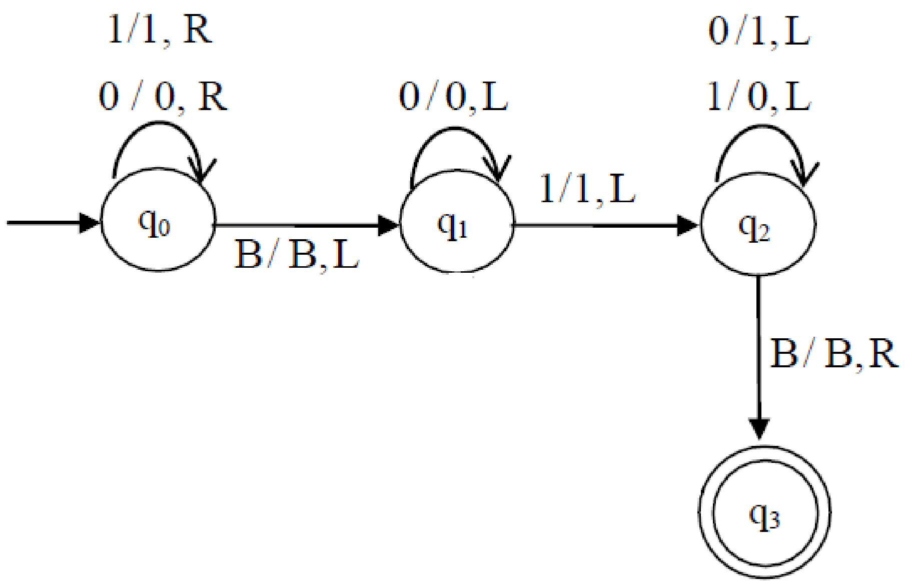
Sol:

<u>2's complement computation:</u>

1. Traverse right and locate the rightmost bit

2. Do not change the bits from the right towards left until the first '1' has been processed

3. Perform complementation to the rest of the bits from right to left (after first 1 is processed)



2. Design a Turing Machine to find 1's complement of a binary number

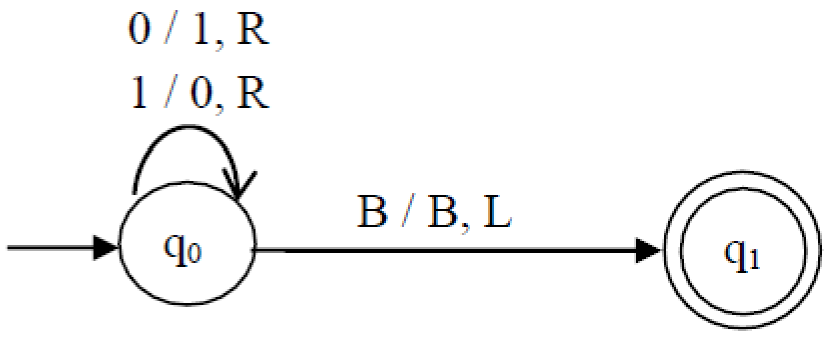
Sol:

<u>1's complement computation:</u>

1. On reading the input, if the symbol is '0', replace it by '1' and move right.

2. If the symbol read is '1' replace it by '0' and move right.

3. Halt the TM if all the string symbols are processed by step 1 and 2



3) Design a Turing Machine that computes addition of two unary numbers.

Sol:

Here the unary number ‘n’ is represented with a symbol ‘0’ n times.

Ex: 2 can be represented as 00

4 can be represented as 0000

6 can be represented as 000000

The separation symbol # is used between two inputs.

Ex: the inputs 5 and 2 can be represented as

00000#00

The input unary numbers m and n can be represented as $0^m\#0^n$

To construct the TM that will add two unary numbers m and n, the Turing Machine M will start with a tape consisting of $0^m\#0^n$ surrounded by blanks. M halts with 0^{m+n} on its tape surrounded with blanks. The required TM is given as

4) Design a Turing Machine that computes the following function

$$f(m,n) = \begin{cases} m-n & if\ m \geq n \\ 0 & otherwise \end{cases}$$

(the above function is called proper subtraction function $m \dot{-} n.$)

Sol: The Turing machine M will start with a tape consisting of $0^m\#0^n$ surrounded by blanks.

M halts with

$$0^{m \dot{-} n}$$

On its tape surrounded by blanks.

The required Turing Machine is given as

Techniques for Turing machine construction:

Describing a Turing machine transitions is not a simple task. To make this task easy, there are some high level conceptual techniques. These techniques can be used in the construction of simple and efficient Turing machines. Some of them are

1. Turing machine with stationary head

2. storage in the state

3. .multiple tracks

4. subroutines

1. Turing machine with stationary head:

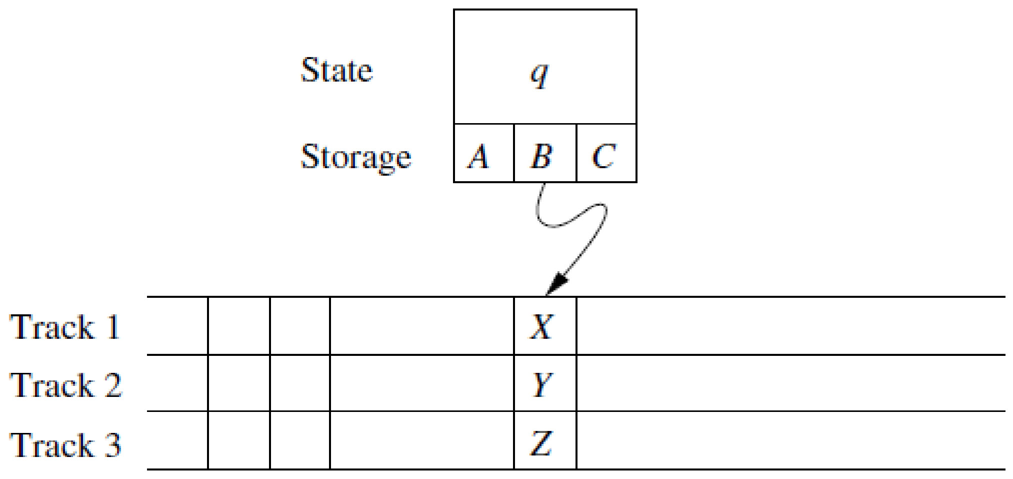
In the definition of TM, the value of δ(q,x), if it is defined, is given as (p,Y,D) where D= L or R. so the head moves to the left or righ after reading an input symbol.

Suppose we want to include the option that the **head can continue to be in the same cell for some input symbol.** Then we can define δ(q,x) as (p,Y,S). This means that the TM, on reading the input symbol x, changes to state p and writes Y in place of x and continues to remain in the same cell.

Thus in this model, the value of δ(q,x), if it is defined, is given as (p,Y,D) where D=L , R and S.

2. storage in the state:

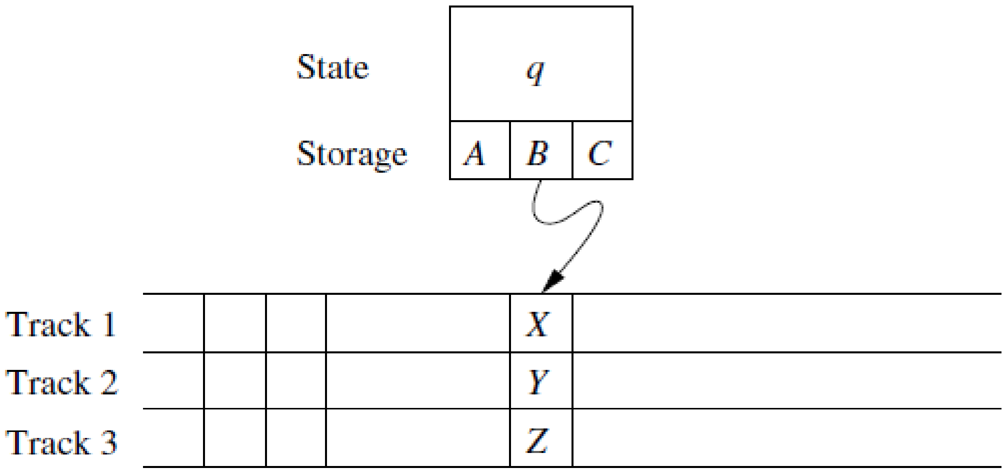
Sometimes it helps to design a TM for a particular language if we imagine that the **state two or more components**. One component is control component and the other components hold data that the TM need to remember. This is illustrated in the following figure.


State
q
Storage
A
B
C
Track 1
Track 2
Track 3
X
Y
Z

Here, the finite control not only consists of a control state q, but also holds three data elements A,B and C. This **technique requires to think state as [q,A,B,C]**. Regarding states this way allows us to describe transitions in a more systematic way.

3.Multiple Tracks:

In a multiple track TM, a single tape is assumed to be divided into several tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples, with one component for each track. For ex, consider the following TM which consists of three tracks.



Here the cell scanned by the tape head contains the symbol [X,Y,Z].

If the number of tracks is equal to k, then the tape alphabet is required to consist of k-tuples of tape symbols. Here the tape symbols are elements of Γ^k .

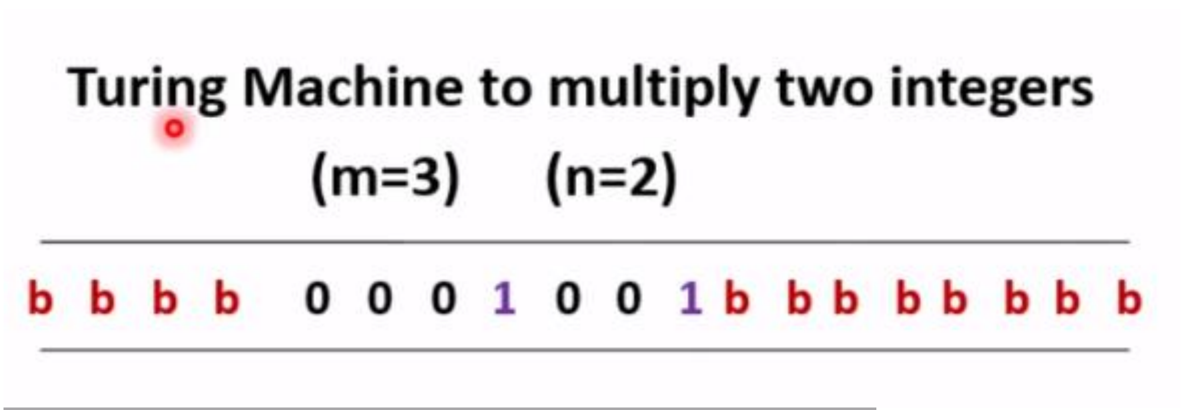
4. subroutines:

Just as a computer program has a main procedure and subroutines, the TM can also be programmed to have a main TM and TMs which serve as subroutines.

A subroutine of a Turing machine is a set of states in the TM such that performs a small useful computation. Usually, a subroutine has single entry state and a single exit state. Many very complicated tasks can be performed by TMs by breaking those tasks into smaller subroutines.

In order that a Turing machine M1 uses another Turing machine M2 as a subroutine, the states of M1 and M2 have to be disjoint. Also when M1 wants to call M2, from a state of M1, the control goes to the initial state of M2. When the subroutine finishes and returns to M1, from the halting state of M2, the machine goes to some state of M1.

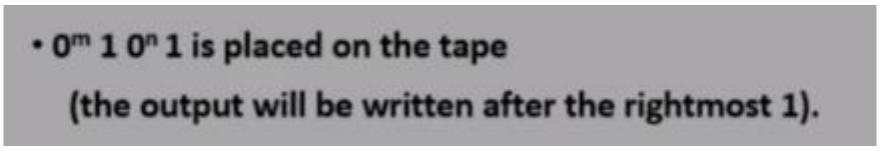
Ex: For multiplying two unary numbers m and n, n has to be copied on m times. We can write a sub TM for copying and main TM will call this m times.



Turing Machine to multiply two integers

(m=3) (n=2)

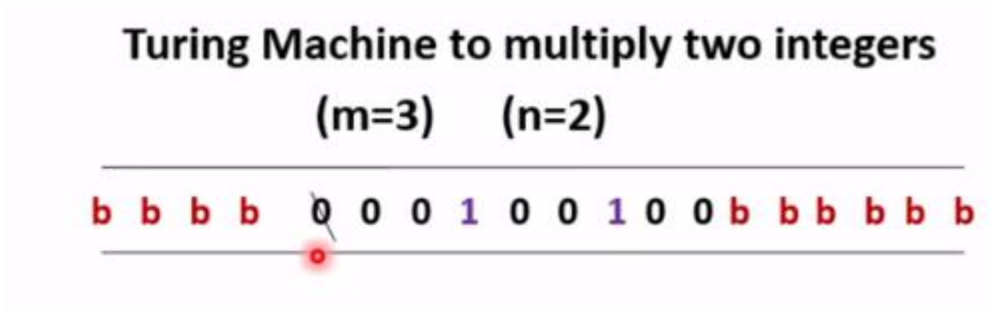
b b b b 0 0 0 1 0 0 1 b b b b b b b b b



• $0^m 1 0^n 1$ is placed on the tape
(the output will be written after the rightmost 1).

•The leftmost 0 is erased.

•A block of n 0's is copied onto the right end.

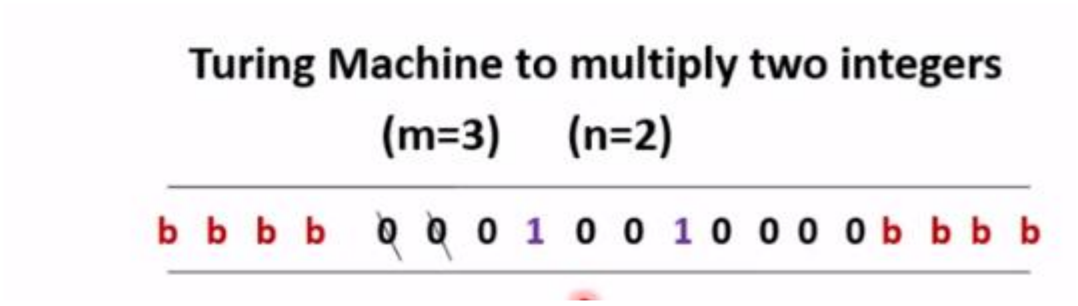


Turing Machine to multiply two integers

(m=3) (n=2)

b b b b $\cancel{0}$ 0 0 1 0 0 1 0 0 b b b b b b

The next 0 is erased and once again block of n 0's is copied onto the right end.

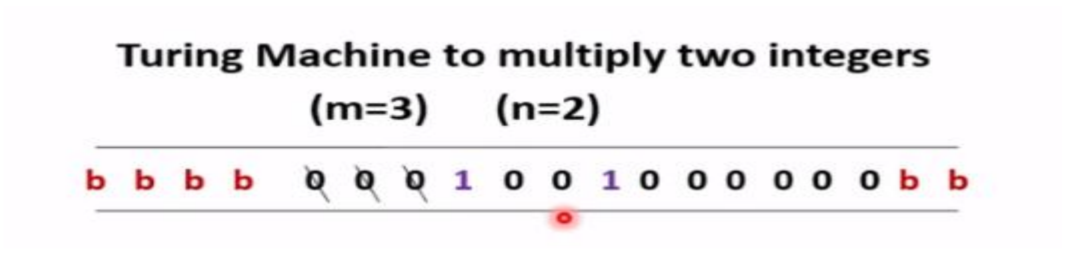


Turing Machine to multiply two integers

(m=3) (n=2)

b b b b $\cancel{0}$ $\cancel{0}$ 0 1 0 0 1 0 0 0 0 b b b b

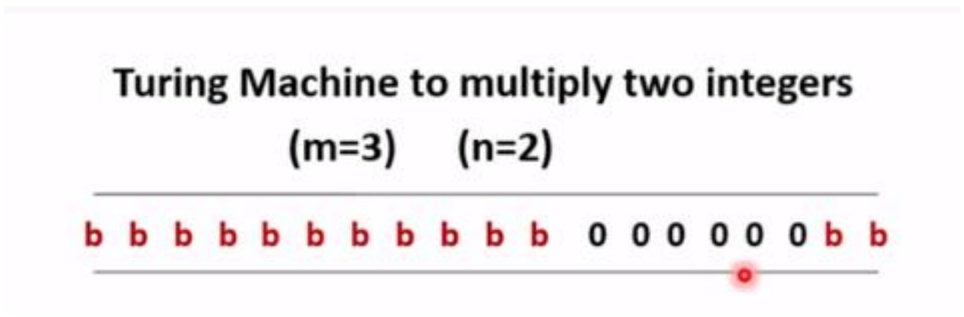
The next leftmost 0 is erased and once again block of n0's is copied on to the right end.


Turing Machine to multiply two integers

(m=3) (n=2)

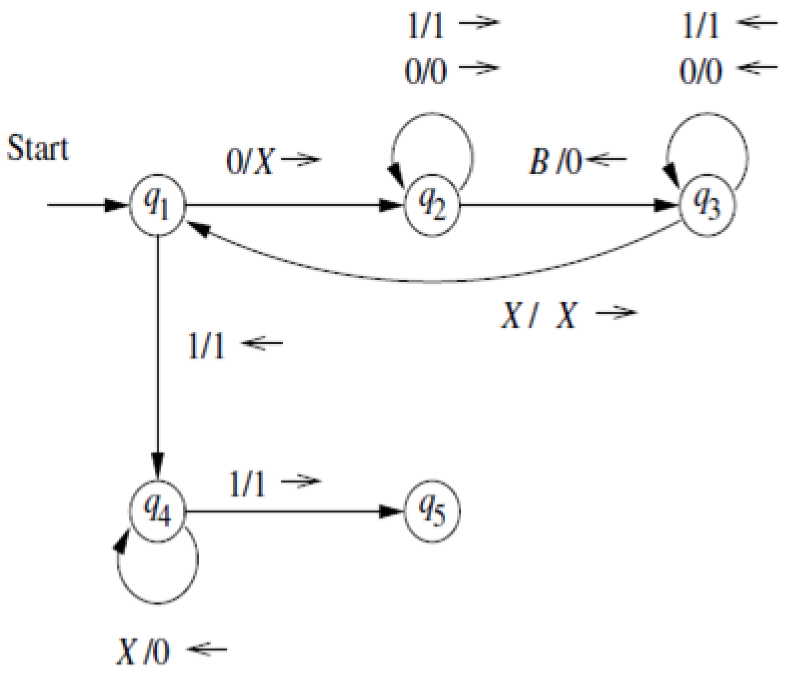
b b b b ~~0~~ ~~0~~ ~~0~~ 1 0 0 1 0 0 0 0 0 0 b b

The above process is repeated m times and 10^n10^{mn} is obtained on the tape. Change the leading 10^n1 to blanks, leaving the product 0^{mn} as output.

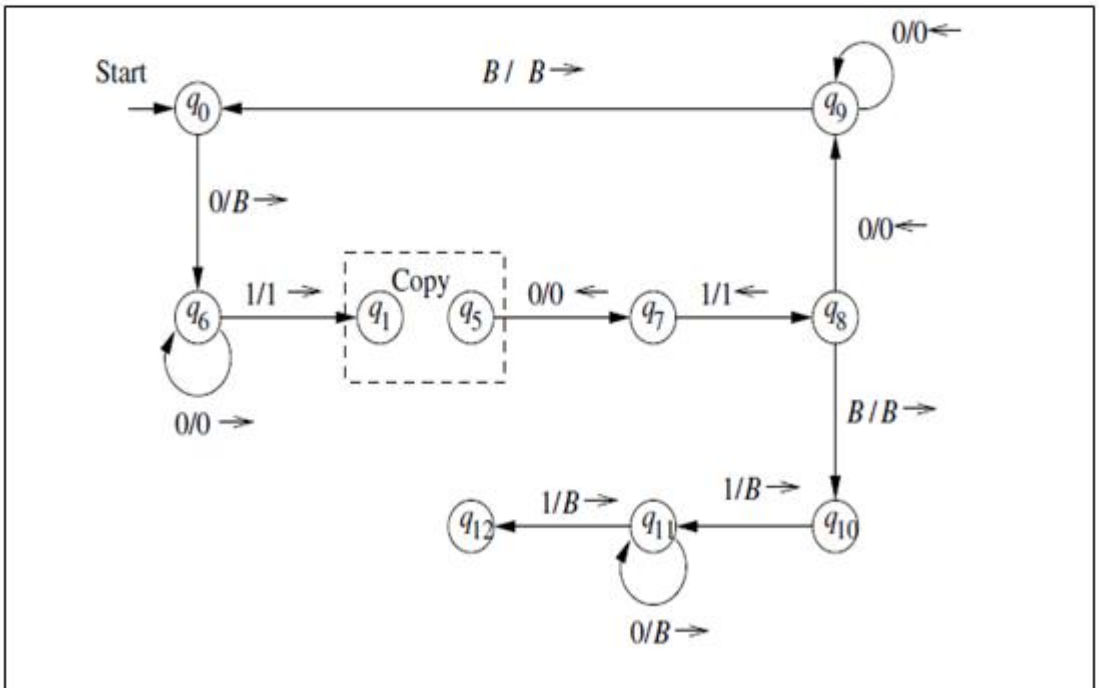

Turing Machine to multiply two integers

(m=3) (n=2)

b b b b b b b b b b b 0 0 0 0 0 0 b b



The subroutine Copy



The complete multiplication program uses the subroutine Copy

Variants of Turing machines:

The following are the variants of turing machines

1. multitape turing machines

2. Nondeterministic Turing Machines

3. Restricted Turing Machines

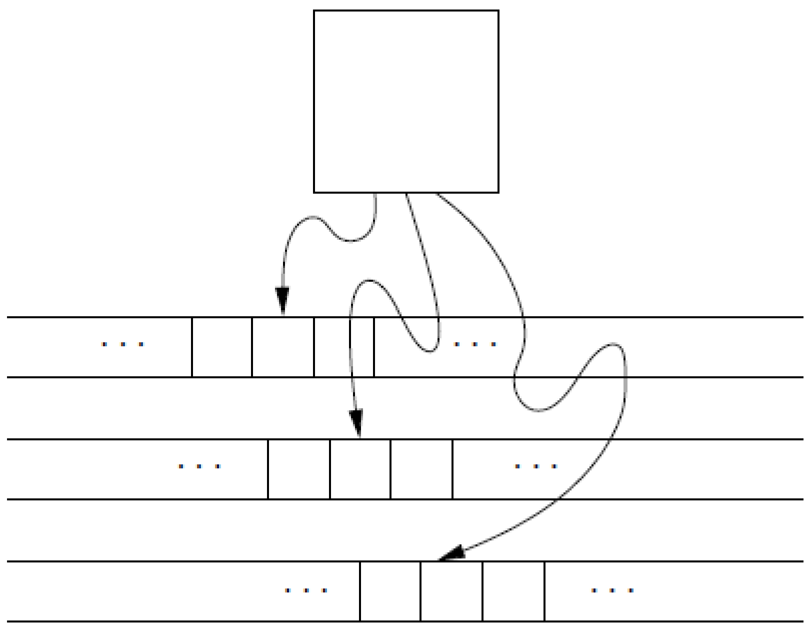
    i) semi-Infinite tape Turing Machines

    ii) multistack machines

    iii)Counter machines

1. Multitape Turing Machines:

A multitape TuringMachine is illustrated in the following figure.



It has a finite control and some finite number of tapes. Each tape is divided into cells, and each cell can hold any symbol of the finite tape alphabet. The set of states includes an initial state and some final states.Initially

1. the Input is placed on the first tape.

2. All other cells of all the tapes hold the blank

3. the finite control is in the initial state.

4. The head of the first tape is at the left end of the input.

5. All other tape heads are at some arbitrary cell. Some tapes other than the first tape are completely blank.

A move of the multitape TM depends on the state and the symbol scanned by each of the tape heads. In one move, the multitape TM does the following.

1. The control enters a new state, which could be the same as the previous state.

2. On each tape, a new tape symbol is written on the cell scanned. Any of these symbols may be the same as the symbol previously there.

3. Each of the tape heads makes a move, which can be either left, right or stationary. The heads move independently, so different heads may move in different directions, and some may not move at all.

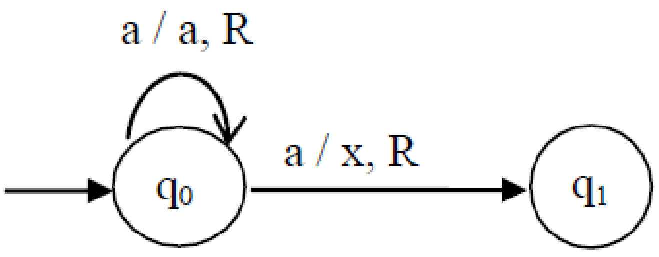
2. Nondeterministic Turing Machine:

For a non deterministic turing machine, the transition function δ is defined such that for each state q and tape symbol x, δ**(q,x) is a set of triplets**

$\{(q_1,y_1,D_1),(q_2,y_2,D_2),\ldots\ldots\ldots\ldots,(q_k,y_k,D_k)\}$ where k is any finite integer.

So a NTM has a finite number of choices of next move for each state and symbol scanned. The NTM M accepts an input w if there is any sequence of choices of move that leads from the initial ID with w as input, to an ID with an accepting state.

Ex:



The above transition takes on two paths for the same input 'a'. the transition of 'a' at q_0 is defined as

$$\delta\,(q_0, a) = \{(q_0, a, R), (q_1, x, R)\}$$

Note:

If M_N is a nondeterministic turing machine, then there exists a deterministic turing machine M_D such that L (M_N) =L (M_D)

3. Restricted Turing Machines:

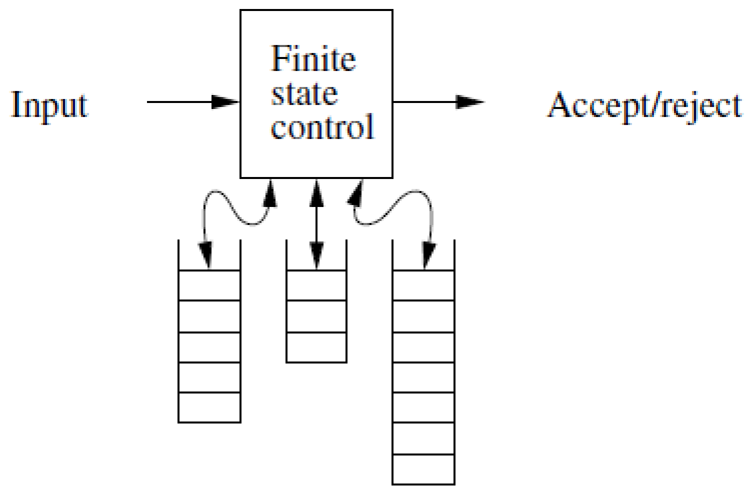
i) Semi Infinite tape Turing Machine:

Restrict a TM to have a tape that is infinite only to the right, with no cells to the left of the initial head position. Here tape head of a Turing machine either move left or right from its initial position but it never moves left of its initial position.

ii) multistack machines:

we can restrict the tapes of a multitape TM to behave like a stacks. A k-stack machine is a deterministic PDA with k stacks. The input is on a separate tape which read once from left-to-right mimicking the input mode for the PDA.

A machine with three stacks is illustrated in the following figure.



It has a finite control which is in one of a finite set of states. It has a finite stack alphabet which it uses for all its stacks. A move of a multistack machine is based on

1. the state of the finite control

2. The Input symbol read which is chosen from the finite input alphabet.

3. the top most stack symbol on each of its stacks/

In one move, the multistack machine can

a) change to a new state

b) replace the top symbol of each stack with a string of zero or more stack symbols. There can be a different replacement string for each stack.

Thus the transition rule for a k-stack machine will be like

$$\delta(q, a, X_1, X_2, \ldots\ldots, X_k) = (p, \alpha_1, \alpha_2, \ldots\ldots\ldots, \alpha_k)$$

The interpretation of this rule is that in state q, with X_i on top of i^{th} stack, for i=1,2,3,…..,k , the machine may consume a from its input (either an input symbol or ε), go to state p, and replace X_i on top of i^{th} stack by string α_i for i=1,2,3,….,k. The multistack machine accepts the input by entering into a final state.

iii) counter machines:

The counter machine has the same structure as that of multistack machine, but in place of each stack is a counter. Counters hold any nonnegative integer, but we can distinguish between zero and nonzero counters.

The move of a counter machine depends on its state, input symbol, and which, if any, of the counters are zero. In one move, the counter machine can

a) change state

b) Add or subtract 1 from any of its counters , independently. But a counter is not allowed to become negative, so it cannot subtract 1 from a counter that is currently zero.

Church-Turing Thesis:

By the 1930s the emphasis was on formalising algorithms. **Alan Turing**, at Cambridge, **devised** an abstract machine, now called a **Turing Machine** to define/represent algorithms. **Alonso Church**, at Princeton, **devised the Lambda Calculus** which formalises algorithms as functions. The demonstrated equivalence of their formalisms strengthened both their claims to validity, expressed as the Church-Turing Thesis that shows the equivalence between mathematical concepts of algorithm or computation and Turing machine. The Thesis can be given as follows.

“a problem can be solved by an algorithm iff it can be solved by a Turing Machine”

(or)

“all algorithmically solvable problems can be solved by a Turing Machine”

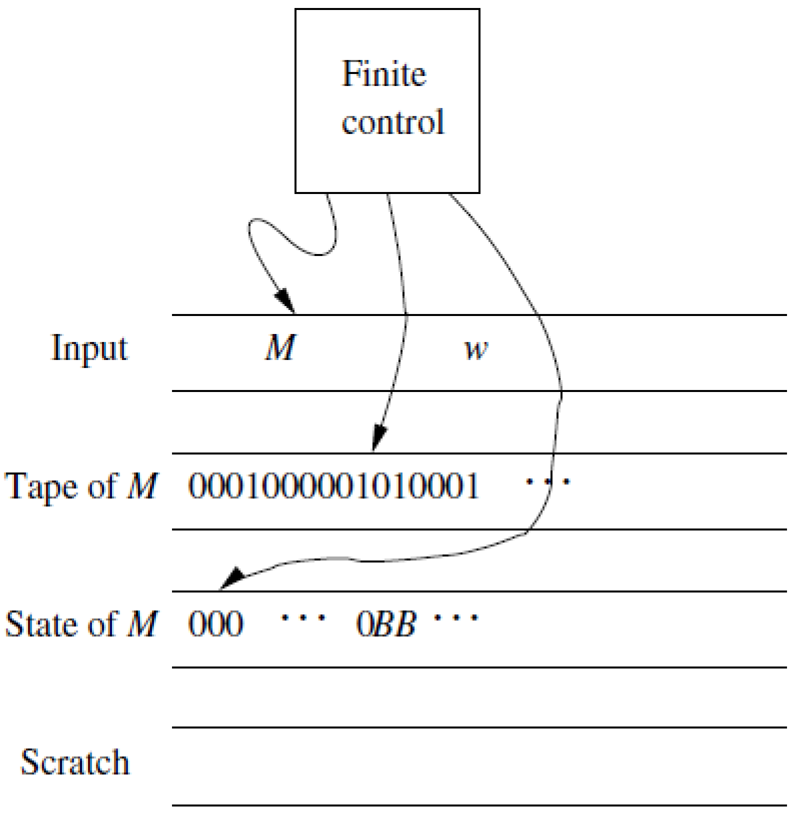
(or)

“a function is computable iff it can be solved by a Turing Machine”

So the church-Turing thesis asserts that if some calculation is effectively carried out by an algorithm, then there exists a Turing Machine which will compute that function.

Universal Turing Machine:

A Universal Turing Machine U is a single Turing Machine that is capable of simulating any other Turing machine. This machine will take coded pair <M ,w> as input, where M is a Turing Machine and string w is an input to M. It will then simulate M’s computation on w. It is a multitape Turing machine whose organization is shown in the following figure.



Initially the transitions of M along with the string w are stored on the first tape. A second tape will be used to hold the simulated tape of M where tape symbol x_i of M will be represented by o^i and tape symbols will be separated by single 1’s. The third tape of U hold the state of M with state q_i represented as i 0’s.

The operation of U can be summarized as follows.

1. Examine the input to make sure that the code for M is a legitimate code for some TM. If not, U halts without accepting.

2. Initialize the second tape to contain the input w in its encoded form. That is, for each 0 of w, place 10 on the second tape, and for each 1 of w , place 100 on the second tape.

3. Place 0, the start state of M, on the third tape, and move the head of U's second tape to the first simulated cell.

4. To simulate a move of M, U searches on its first tape for a transition $0^i10^j10^k10^l10^m$ such that 0^i is the state on tape 3 and 0^j is the tape symbol of M that begins at the position on tape 2 scanned by U. This transition is the one that m would next make. U should:

(a) Change the contents of tape 3 to 0^k; that is, simulate the state change of M. To do so, U first changes all the 0's on tape 3 to blanks, and then copies 0^k from tape 1 to tape 3.

(b) Replace 0^j on tape 2 by 0^l; that is, change the tape symbol of M.

(c) Move the head on tape 2 to the position of the next 1 to the left or right, respectively, depending on whether $m = 1$ (move left) or $m = 2$ (move right). Thus, U simulates the move of M to the left or to the right.

If M has no transition that matches the simulated state and tape symbol then no transition will be found. Thus M halts as well as U halts.

5. If M enters into its accepting state, then U accepts.

In this way, U simulates M on w. U accepts the coded pair <M,w> if and only if M accepts w.