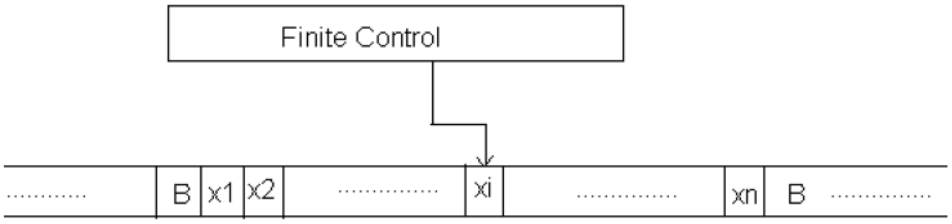
Turing Machine

Turing machine is an abstract machine developed by an English Mathematician Alan Turing in 1936. The model of computation provides a theoretical foundation for modern computers. A Turing machine will have;

- A finite set of alphabets
- A finite set of states
- A linear tape which is potentially infinite to both end.



The tape is marked off into squares, each of which can hold one symbol from the alphabet. If there is no symbol in the square then it contains blank. The reading and writing is done by a tape head. The tape serves as:

- Input device (input is simply the string assumed to this)
- The memory available for use during computations
- The output device (output is the string of symbols left on the tape at the end of computation).

A single move of Turing machine is function of the state of TM and the current tape symbol and it consists of three things;

- Replacing the symbol in the current squared by another, possibly different symbol.
- Moving the tape head one square right or left or leaving it where it is.
- Moving from current state to another, possibly different state.

Difference between TM and Other Automata (FSA and PDA)

The most significant difference between the TM and the simpler machine (FSA or PDA) is that; in a Turing Machine, 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.

Formal Description of Turing Machine

A Turing Machine T M is defined by the seven-tuples, M = (Q, Σ, Γ, δ, q0, B, F) where,

Q = the finite set of states of the finite control
Σ = the finite set of input symbols

Γ = the complete set of tape symbols Σ is always subset of Γ.
q0 = the start state; q0 ε Q
B = the blank symbol; B ε Γ but B does not belong to Σ.
F = the set of final or accepting states; F is subset of Q
δ = the transition function defined by
Q × Γ→ Q × Γ × (R, L, S); where R, L, S is the direction of movement of head left, or right or stationary. i.e. δ(q, x) = δ(p, Y, D); which means T M in state q and current tape symbol x, moves to next state P, replacing tape symbol x with Y and move the head either direction or remains at same cell of input tape.

Instantaneous Description for T M

The configuration of a T M is described by Instantaneous description (ID) of T M as like PDA.
A string x_1x_2……..$x_{i-1}qx_ix_{i+1}$……..x_n represents the I.D. of T M in which;

- q is the state of T M .
- the tape head scanning the ith symbol from the left.
- x_1x_2…… x_n is the portion of tape between the leftmost and rightmost non-blank.(If the head is to the left of leftmost non blank or to the right of rightmost non-blank then some prefix or suffix of x_1x_2……x_n will be blank and I will be 1 or n respectively.)

Moves of T M

The moves of T M, M = (Q, Σ, Γ, δ, q_0, B, F) is described by the notation ├, "yield", for single move and by ├* for zero or more moves as in PDA.

a) For δ(q, x_i) = (P, Y, L) i.e. next move is leftward then, x_1x_2……$x_{i-1}qx_ix_{i+1}$……..x_n ├x_1x_2….x_{i-1} P $x_{i-1}Yx_{i+1}.x_n$ reflects the change of state from q to p and the replacement of symbol x_i with Y and then head is positioned at i-1 (next scan is x_{i-1}).

i. If i = 1, M moves to the left of x_1 i.e. qx_1x_2……..x_n ├ pBYx_2…….x_n

ii. If i=n, Y = B, then M moves to state p and system B written over x_n joins the infinite sequence of trailing blanks which does not appear in next ID as x_1x_2……..x_{n-1}qx_n ├ x_1x_2……x_{n-2}px_{n-1}

b) If δ(q, x_i)= (P, Y, R) i.e. next move is rightward then, x_1x_2…….$x_{i-1}qx_i$……..x_n ├ x_1x_2….x_{i-1} Y p x_{i+1}……..x_n, which reflects that the symbol xi is replaced with Y and head has moved to cell i+1 with change in state from p to q.

If i= n, then i+1 cell holds blank which is not part of previous ID; i.e. x_1x_2……x_{n-1} ├ x_1x_2…… x_{n-1} YpB.

If i=1, Y=B, then the symbol B written over xi joins the infinite sequence of leading blanks and does not appear in next ID; i.e. x_1x_2…….x_n ├px_2x_3……x_n.

Note: Write equivalent definition from Adesh Kumar.

Consider an example; A TM accepting {0^n1^n / n ≥ 1}

- Given finite sequence of 0's and 1's on its tape preceded and followed by blanks.
- The TM will change 0 to an X and then a 1 to Y until all 0's and 1's are matches.

- Starting at left end of the input, it repeatedly changes a 0 to an X and moves to the right over whatever 0's and Y's it sees until comes to a 1.
- It changes 1 to a Y, and moves left, over Y's and 0's until it finds X. At that point, it looks for a 0 immediate to the right. If finds one 0 then changes it to X and repeats the process changing a matching 1 and Y.

Now, TM will have;

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

The transition rule for the move of M is described by following transition table:

	0	1	X	Y	B
q0	(q1,X,R)			(q3, Y, R)	
q1	(q1,0,R)	(q2,Y,L)		(q1, Y, R)	
q2	(q2,0,L)		(q0, X, R)	(q2, Y, L)	
q3				(q_3, Y, R)	
q4					(q4, B, R)

Now, the acceptance of 0011 by the TM, M1 is described by following sequence of moves;

q_0 0011 ⊢ X $q_1$011
⊢X0$q_1$11
⊢X$q_2$0Y1
⊢X$q_0$0Y1
⊢XXq_1Y1
⊢XXY$q_1$1
⊢XXq_2YY
⊢Xq_2XYY
⊢XXq_0YY
⊢XXYq_3Y
⊢XXYYq_3B
⊢XXYYBq_4B Halt and accept.

For string 0110 try yourself.

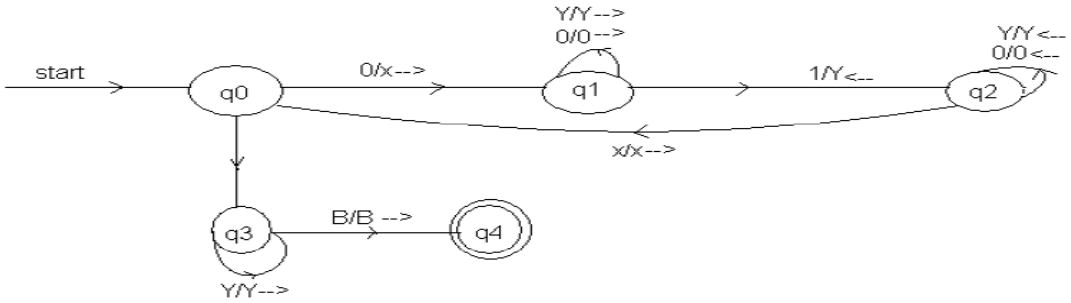
Transition diagram for a TM:

A transition diagram of TM consists of,

- A set of nodes representing states of TM.
- An arc from any state, q to p is labeled by the items of the form X / YD, where X and Y are tape symbols, and D is a direction, either L or R. that is, whenever $\delta(q, x) = (P, Y, D)$, we find the label X / YD on the arc from q to p.

However, in diagram, the direction D is represented by ← for left (L) and → for right (R)

Thus, transition diagram for the TM for L = $\{0^n1^n / n \geq 1\}$ as;



The language of Turing Machine:

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

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

The Turing Machine is designed to perform at least the following three roles;

1) As a language recognizer: TM can be used for accepting a language like Finite Automaton and Pushdown Automata.

2) As a Computer of function: A TM represents a particular function. Initial input is treated as representing an argument of the function. And the final string on the tape when the TM enters the halt state; treated as representative of the value obtained by an application of the function to the argument represented by the initial string.

3) As an enumerator of string of a language: It outputs the strings of a language, one at a time in some systematic order that is as a list.

Turing Machine for 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.

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

2) We can show by underlining that symbol to the current position of machine head in the tape as (q, BwB) or, we can represent it by ID of TM as BwqB.

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

i.e. If TM = (Q, Σ, Γ, δ, q0, B, {qa}) is a turing machine . Then this TM is said to be halt on input w if and only if BwqB yields to BαqaB, for some αεΓ* i.e.(q,BwB) ⊢(qa, B α B)

Formal Definition:
A function f(x) = y is said to be computable by a TM and defined as; (Q, Σ, Γ, δ, q0, B, {q0})
If (q0, B x B) ⊢* (qa, ByB); where x may be in some Σ*, and y may be in some Σ2* and Σ1Σ2……
It means that if we give input x to the Turing machine; it gives output as a string if it computes the function f(x) = y.
Example: Design a TM which computes the function f(x) = x+1 for each x belonging to set of natural numbers.
Given the function, f(x) = x+1. Here, we represent the input x on the tape by a number of 1's on the tape.
i.e. for x=1, input tape will have B1B,
for x=2 input tape will have B11B,
and so on……..

Similarly, output can be seen by the number of 1'son the tape when machine halts.
Let us configure a TM as;
TM = (Q, Σ, Γ, δ, q0, B, {qf});
Where Q = {q0, qf}
Γ = {1, B}
Halt state = {q0}
Then δ can be simulated as;

	B	1
q0	$(q_0, 1, S)$	$(q_f, 1, R)$
q1		

So, let the input be x = 4. So, input tape at initial step consists of B1111B.
Then (q0, B1111B) ⊢ (q0, B11111) ⊢ (qf, B11111B).Which means output is 5 accepted.

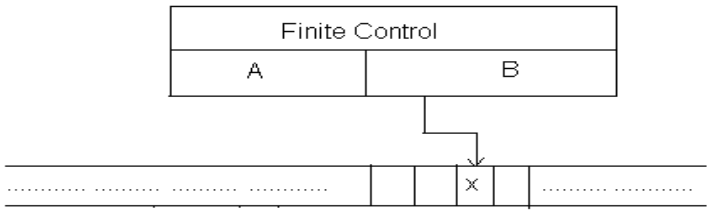
Turing Machines and Halting

There is another notion of acceptance that is commonly used for Turing machines: 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.

The Turing machine described above was not designed to accept any language rather we viewed it as computing an arithmetic function.
We always assume that a TM halts if it accepts. i.e. without changing language accepted, we can make δ(q,X) undefined whenever q is an accepting state.

Turing Machine with Storages in the state:

In Turing Machine, generally, any state represents the position in the computation. But the state can also be used to hold a finite amount of data. We can use the finite control not only to represent a position in the computation / program of the Turing Machine, but to hold a finite amount of data. In this case, a state is considered as a tuple; (state, data).
Following, it shows the model,



With this model of computation, δ is defined by:
δ([q, A], x) = ([q1, x], Y, R); means that q is the state and data portion associated with q is A. the symbol scanned on the tape is copied into the second component of the state and moves right entering state q1 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.

Thus, the TM can be designed as;
M = (Q, {0, 1}, {0, 1, B}, δ, [q0, B], {[q1, B]})
The set of states Q, is {q0, q1} × {0, 1, B}. That is the states may be thought of a pair with two components;

a) A control portion, q0 or q1 that remembers the TM is doing. Control state q0 indicates that M has not yet read its first symbol, while q1 indicates that it has read the symbol, and is checking that it does not appear elsewhere, by moving right and hoping to reach a blank cell.

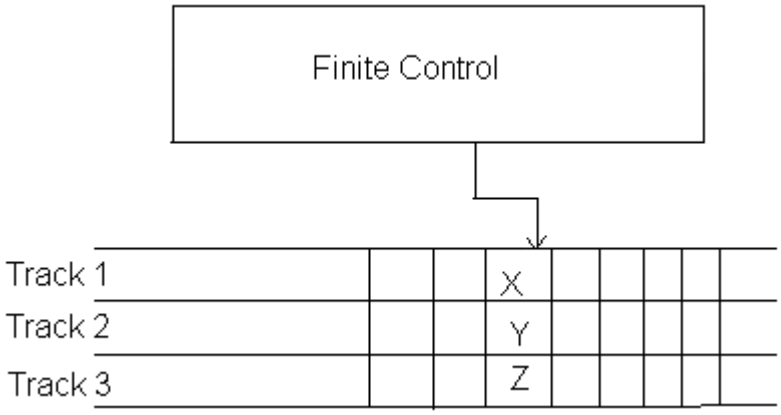
b) A data portion, which remembers the first symbol seen, which must be 0 or 1. The symbol B in this component means that no symbol has been read.

The transition function of M is defined as;
1) δ([q0, B], a) = ([q1, a], a, R); for a=0 or 1
2) δ([q1, a], a') = ([q1, a], a', R); where a' is the "complement" a, i.e. a'=0 if a=1 and a'=1 if a=0
3) δ([q1, a], B) = ([q1, B], B, R); for a=0 or 1. If M reaches the first blank, it enters the accepting state [q1, B].

Note: Since M has no definition for δ([q1, a], a); for a = 0 or 1, thus if M encounters a second occurrence of some symbol it stored initially in its control, it halts without having entered the accepting state. [eg: as 110 or 001 does not lie within the language so should be rejected.]

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 alphabet Γ is a set consisting of tuples like,
Γ = {(X, Y, Z), ……}
The tape head moves up and down scanning symbols in the tape at one position.

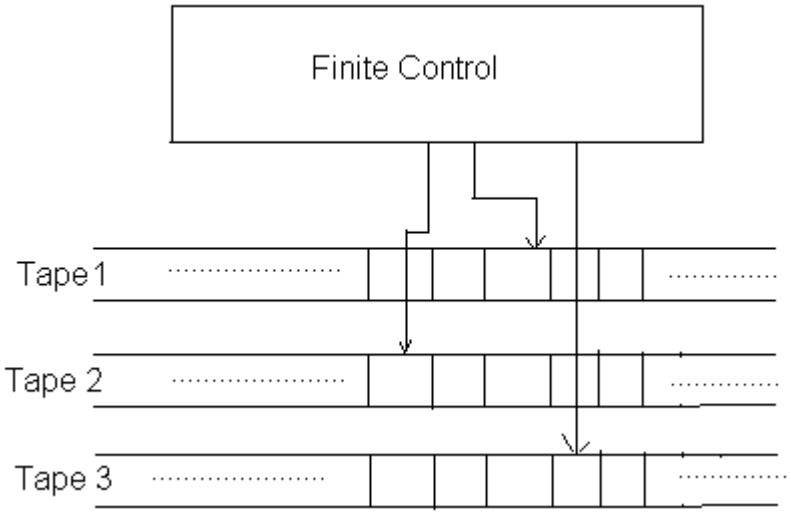
Like the technique of storage in the state, using multiple tracks does not extend what the TM can do. It is simply a way to view tape symbols and to imagine that they have a useful structure.

Sub-routines:
A complex TM can be thought as built from a collection of interacting components like general program. Such components of TM are called sub-routines. A TM subroutine is a set of states that performs some useful processes and can be called into another machine for the part of that computation.

Multi-tape Turing Machine:

Modern computing devices are based on the foundation of TM computation models. To simulate the real computers, a TM can be viewed as multi-tape machine in which there is more than one tape. However, adding extra tape adds no power to the computational model, only the ability to accept the language is concerned.

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;

1. The Input (finite sequence of input symbols) w is placed on the first tape.
2. All other cells of the tapes hold blanks.
3. TM is in initial state q0.
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. Since all other tapes except first tape consists completely blank.

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 the following:

1. The control enters in a new state, which may be same previous state.
2. On each step, a new symbol is written on the cell scanned, these symbols may be same as the symbols previously there.
3. Each of the tape head make a move either left or right or remains stationary. Different head may move different direction independently i.e. if head of first tape moves leftward; at same time other head can move another direction or remains stationary.

The initial configuration (initial ID) of multi-tape TM with n-tapes is represented as;
(q0, ax, B, B, …… B); n+1 tuples. Where w= ax is an input string and head first tape is scanning first symbol of w.
So, in general, it can be rewritten as;
$(q, x_1a_1y_1, x_2a_2y_2, \ldots\ldots\ldots\ldots\ldots\ldots x_na_ny_n)$
Where each x_i are the portion of string on tapes before the current head position, each a; are the symbol currently scanning in each tapes and each y_i are the portion of string on tapes just rightward to the current head position.
q is the control state.

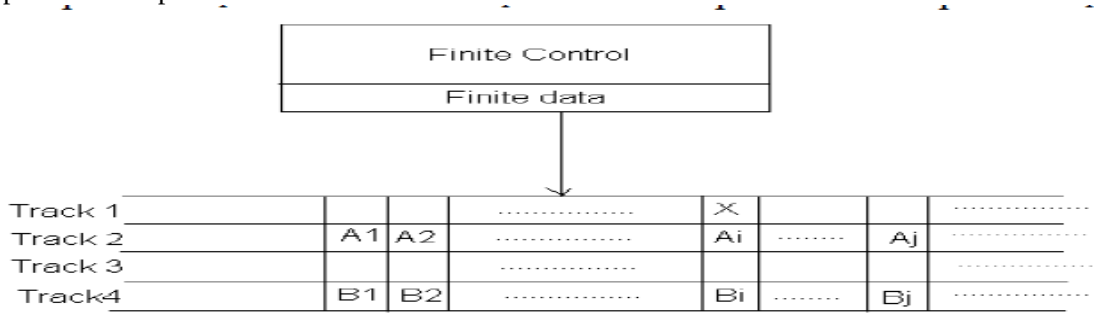
Equivalence of one-tape and Multi-tape TM

Any recursively enumerable languages that are accepted by one-tape TM are also accepted by multi-tape TM. i.e. Any n-tuples TM for $n \geq 2$, are at least as powerful as 1-tape TM's.

Simulating one tape TM with multi-tape TM:

Theorem: Every language accepted by a Multi-tape TM is recursively enumerable.
Or,
Any languages that are accepted by a multi-tape TM are also accepted by one tape Turing Machine.

Proof:
Let L is a language accepted by a n-tape TM, T1= (Q1, Σ, Γ1, δ1, q1, B, F1). Now, we have to simulate T1 with a one-tape TM, T2 = (Q2, Σ, Γ2, δ2, q2, B, F2) 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 head position of tape 1 of T1 and track3 in T2 holds head position of tape2 of T1.



Now, to simulate a move of T1,
 T2's head must visit the n-head markers so that it must remember how many head markers are to its left at all times. That count is stored as a component of T2's finite control.
 After visiting each head marker and storing the scanned symbol in a component of its finite control, 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 in 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 a state of T1. Hence, whatever T1 accepts T2 also accepts.

Non-Deterministic Turing Machine

A non-deterministic Turing Machine (NTM), T = (Q, Σ, Γ, δ, q0, B, F) is defined exactly the same as an ordinary TM, except the value of transition function δ. In NTM, the values of the transition function δ are subsets, rather than a single element of the set Q×Γ×{R,L,S}. Here, the transition function δ is such that for each state q and tape symbol x, δ(q, x) is a set of triples:

{(q1, Y1, D1), (q2, Y2, D2), ……(qk, Yk, Dk) where k is any finite integer.

The NTM can choose, at each step, any of the triples to be the next move. It cannot, however pick a state from one, a tape symbol from another, and the direction from yet another.

Church-Turing Thesis:

The Church-Turing Thesis states that in its most common form that "every computation or algorithm can be carried out by a Turing Machine."
This statement was first formulated by Alonzo Church. It is not a mathematically precise statement so un-provable. However, it is now generally assumed to be true.

The thesis might be replaced as saying that the notation of effective or mathematical method in logic and mathematics is captured by TM. It is generally assumed that such methods must satisfy the following requirements;
1. The method consists of a finite set of simple and precise instructions that are described with a finite number of symbols.
2. The method will always produce the result in a finite number of steps.
3. The method can in principle be carried out by a human being with only paper and pencil.
4. The execution of the method requires no intelligence of the human being except that which is needed to understand and execute the instructions.

The example of such a method is the "Euclidean algorithm" for determining the "Greatest Common Divisor" of two natural numbers.

The invention of TM has accumulated enough evidence to accept this hypothesis.
Following are the some of evidences:

1. In nature, a human normally works on 2D paper. The transfer of attention is not adjacent block like TM. However, transferring attention from one place to another during computation can be simulated by TM as one human step by multiple tapes.

2. Various extensions of TM model have been suggested to make computation efficient like doubly infinite tape, a tape with multiple tracks, multi-tape etc. In each case the computational power is reserved.

3. Other theoretical model have been suggested that are closer to modern computers in their operation (e.g. simple programming type languages, grammar and others)

4. Since the introduction of the TM, no one has suggested any type of computations that ought to be included in the category of "Algorithmic procedure."
Thus after adopting Church-Turing Thesis, we are giving a precise meaning of the term;
"An algorithm is a procedure that can be executed on a Turing Machine."

Universal Turing Machine

If a TM really is a sound model of computation, it should be possible to demonstrate that it can act as a stored program machine, where the program is regarded as an input, rather than hard-wired. We shall construct a Turing Machine Mu, that takes as input a description of a Turing Machine M and an input word x, and simulates the computation of M on input x. A machine such as Mu that can simulate the behavior of an arbitrary TM is called a Universal Turing Machine.

Thus, we can describe a Universal Turing Machine Tu as a TM, that on input <M, w>; where M is a TM and w is string of input alphabets, simulates the computation of M on input w.
specially,

- Tu accepts <M, w> iff M accepts w.
- Tu rejects <M, w> iff M rejects w.

Encoding of TM

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 T1, and input w into e (T1) 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 T1 =(Q1,{0,1}, Γ, δ, q1,B,F) as binary string ,we first assign integers to the states, tape symbols and directions. We assume two fixed infinite sets Q = {q1, q2, q3……} and S = {a1, a2, a3…….} so that Q1 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 separators.

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, separated by pairs of 1's:
$m_1 11 m_2 11 \ldots\ldots\ldots m_n$.

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
$S(ai) = 0^{i+1}$ for each ai ε S
$S(qi) = 0^{i+2}$ for each qi ε Q
S(S) = 0
S(L) = 00
S(R) = 000

Consider an example, where, TM T is defined as;
T = ({q1, q2, q3}, {a, b} {a, b, B}, δ, q1, B, F),

where δ is defined as
δ(q1, b) = (q3, a, R) → m1
δ(q3, a) = (q1, b, R) → m2
δ(q3, b) = (q2, a, R) → m3
δ(q3, B) = (q3, b, L) → m4

Now, using the encoding function s defined above, as the rule, we have
S(q1) = 000
S(q2) = 0000
S(q3) = 00000
S(a1) = 00 considering a1 = a & a2 = b
S(a2) = 000
S(B) = 0
S(R) = 000
S(L) = 00
S(S) = 0

Now, encoding for rules
e (m1) = S(q1) 1 S(b)1 S(q3)1 S(a)1 S(R)
= 00010001000001001000

e (m2) = S(q3) 1 S(a)1 S(q1)1 S(b)1 S(R)
= 00000100100010001000

e (m3) = S(q3) 1 S(b)1 S(q2)1 S(a)1 S(R)
= 00000100010000100100 0

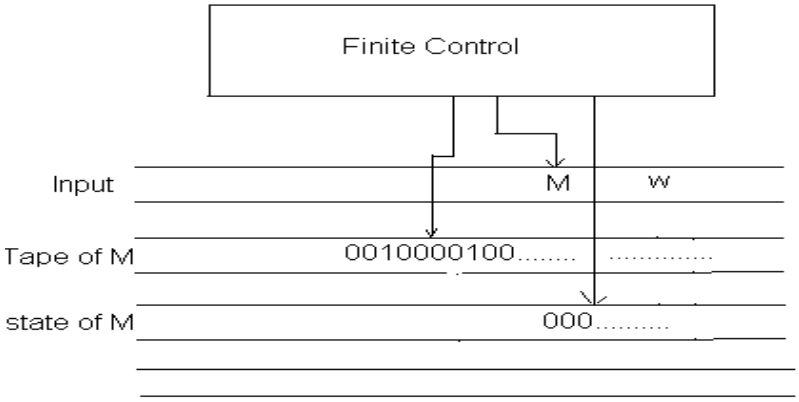
e (m4) = S(q3) 1 S(B)1 S(q3)1 S(b)1 S(L)
= 0000010100000100010 0

Now the code for TM T is
e(m1)11e(m2)11e(m3)11e(m4)=00010001000001001000
11000001001000100010001100000100010000100100 0
11000001010000010001 00

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

Operation of Universal Turing Machine

Now, the Universal Turing Machine Tu can be described as a multi-tape Turing Machine in which the transitions of any other Turing Machine M are stored initially on the first tape, along with string w. The second tape holds the simulated tape of M, using the same format as for the code of M. The third tape holds the state of M, with suitable encoding. The sketch for Tu can be shown as;



The operation of universal Turing Machine Tu can be described as;

1. Examine the input to make sure that the code for M is valid code for some TM. If not, Tu halts without accepting. Any invalid codes represent TM with no moves.
2. Initialize the second tape to contain the input w in encoded form (simulated tape of M).
3. Place code of q1 (the start state of M) on the third tape and move head of Tu's second tape to the first simulated sell.
4. To simulate a move of M, Tu searches on its first tape for a transition $0^i10^j10^k10^l10^m$ such that o^i is the state on tape 3, o^j is the tape symbol of M that begins at the position on tape 2 scanned by Tu. This transition is the one move of M. Tu should;
   a) Change the content of tape 3 to 0^k, i.e. simulate state change of M.
   b) Replace 0^j on tape 2 by 0^l i.e. change the tape symbol of M. If more or less space is needed ($j \neq l$) use the scratch tape and shifting over technique as;
      - Copy onto a scratch tape, the entire non-blank tape to the right of where new value goes.
      - Write the new value using correct amount of space for that value.
      - Recopy the scratch onto tyape2, immediately to right of new value
   c) Move head on tape 2 to the position of next 1 to the left or right or stationary. If m = 1 (stationary), if m = 2 (move left) and m = 3 (move right).

   Thus, Tu simulates one move of M.
5. If M has no transition that matches the simulated state and tape symbol, then in 4 , no transition will be found. Thus Tu halts in the simulated configuration and Tu must do likewise.
6. If M enters its accepting state then Tu accepts.