Turing Machine

Compiled by: Asst. Prof. Daya Ram Budhathoki
Nepal Engineering College.

Turing Machine as a Physical Computing Device:

Turing machines, first described by Alan Turing in (Turing 1937), are simple abstract computational devices intended to help investigate the extent and limitations of what can be computed.

Neither DFA nor PDA can be regarded as a truly general models for computers, since they are not capable of recognizing even such simple language $L=\{a^n b^n c^n \mid n \geq 0\}$. we can visualize Turing machine as a physical computing device that can be represented as a diagram below.

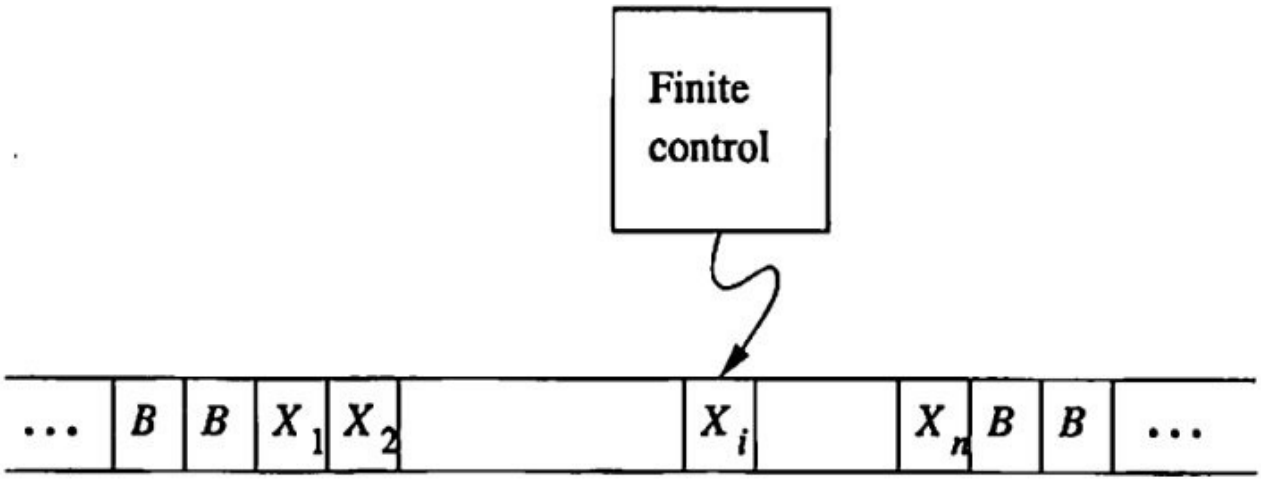


Fig: Turing Machine

The machine consists of a finite control which can be in any of a finite set of states. There is a tape divided into squares or cell; each cell can hold any one of a finite number of symbols. Initially, the inputs which is a finite-length string of symbols chosen from the input alphabet, is placed on the tape. All other tape cells, extending infinitely to the left and right, initially hold a special symbol called the blank. The blank is a tape symbol but not an input symbol, and there may be other tape symbols besides the input symbols and the blank, as well. There is a tape head that is always positioned at one of the tape cells. The Turing machine is said to be scanning that cell. Initially, the tape head is at the leftmost cell that holds the input.

A move of the Turing machine is a function of the state of the finite control and the tape symbol scanned. In one move, the Turing machine will:

1. Change state. The next state optionally may be the same as the current state.
2. Write a tape symbol in the cell scanned. This tape symbol replaces whatever symbol was in that cell. Optionally, the symbol written may be the same as the symbol currently there.
3. Move the tape head left or right. In our formalism we require a move, and do not allow the head to remain stationary. This restriction does not constrain what a Turing machine can compute,

since any sequence of moves with a stationary head could be condensed, along with the next tape-head move, into a single state change, a new tape symbol, and a move left or right.

In Summary:

• Unrestricted memory: an infinite tape
  - A finite state machine that reads/writes symbols on the tape
  - Can read/write anywhere on the tape
  - Tape is infinite in one direction only (other variants possible)

• Initially, tape has input and the machine is reading (i.e., tape head is on) the leftmost input symbol.

• Transition (based on current state and symbol under head):
  - Change control state
  - Overwrite a new symbol on the tape cell under the head
  - Move the head left, or right.

Turing Machine VS Finite Automata:

1. A TM can both write on the tape and read from it;
a FA can only read its input

2. The read/write head of a TM can move both to the left and to the right;
a FA can move in one direction only

3. The tape of a TM is infinite;
the input of a FA is finite

4. The special states of a TM for rejecting and accepting the input take immediate effect;
FA terminates when input is entirely

Formal Definition of Turing Machine:

We describe a TM by the 7-tuple

M $=(Q, \Sigma, \tau, \delta, q0, B, F)$

whose components have the following meanings:

Q: The finite set of states of the finite control.

Σ: The finite set of input symbols.

τ: The complete set of tape symbols in τ, is always a subset of F.

δ: The transition function. The arguments of δ(q, X) are a state q and a tape symbol X. The value of δ(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 τ 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.

q0: The start state, a member of Q, in which the finite control is found initially.

B: The blank symbol. This symbol is in τ but not in Σ; i.e., it is not an input symbol. The blank appears initially in all but the finite number of initial cells that hold input symbols.

F: The set of final or accepting states, a subset of Q.

Instantaneous description:

A Turing machine M computes as follows. Initially M receives its input W == WI W2 . . . W n E L:* on the leftmost n squares of the tape, and the rest of the tape is blank (i.e., filled with blank symbols). The head starts on the leftmost square of the tape. Note that L: does not contain the blank symbol, so the first blank appearing on the tape marks the end of the input. Once M has started, the computation proceeds according to the rules described by the transition function. If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L. The computation continues until it enters either the accept or reject states at which point it halts. If neither occurs, M goes on forever.

An instantaneous description or configuration of a Turing machine requires:

1. The state the Turing machine is in.
2. The contents of the tape
3. The position of the tape head on the tape.

This is written in the form:

$X_i.....X_jq_mX_k.....X_l$

Where X's are the symbol on the tape, q_m is the current state and the tape head is on the square containing Xk(The symbol immediately following q_m).

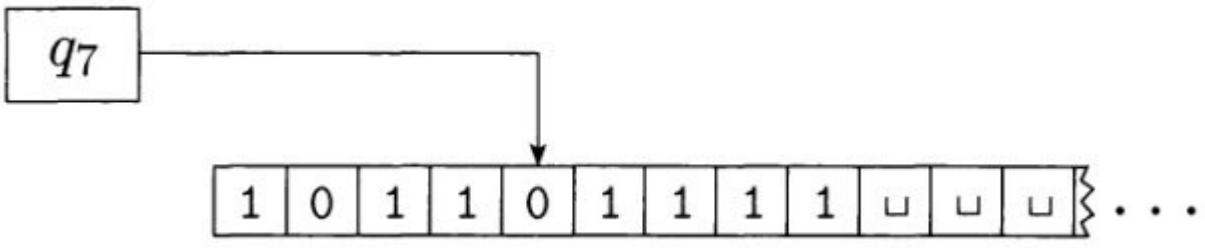
The move of a Turing machine can therefore be expressed as a pair of instantaneous descriptions, separated by " ⊢ " .

For example $\delta(q_5,b) = (q_8,c,R)$

Then possible move can be:

$abbabbq_5babb \vdash abbabcq_8abb$

As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location. A setting of these three items is called a configuration of the Turing machine. For example, 10 11q70 1111 represents the configuration when the tape is 101101111, the current state is Q7, and the head is currently on the second o. The following figure depicts a Turing machine with that configuration.



Fig; A Turing machine with configuration $1011q_701111$

Designing Turing Machine:

For Σ= {a, b} design a Turing Machine that accepts the language: L = { $a^n b^n$: n ≥ 1}
solution:
we solve the problem in the following fashion. Starting at the leftmost a, we check it off by replacing it with some symbol, say X. We then let the read-write head travel right to find the leftmost b, which in turn is checked off by replacing it with another symbol, say Y. After that, we go left again to the leftmost a, replace it with an X, then move to the leftmost b and replace it with Y and so on. Traveling back and forth this way, we match each a with a corresponding b. If after some time not any a's or b's remain, then the string must be in L.

Working out the details, we arrive at a complete solution for which
Q = {q0,q1,q2,q3,q4}
F = {q4},
Σ = {a,b},
τ = {a,b,X,Y,B}.
The transition can be broken into several parts. The set:

δ(q0,a) = (q1,X, R),
δ(q1,a) = (q1,a, R) ,
δ(q1,Y)= (q1,Y,R),
δ(q1,b) = (q2,Y, L)

replaces the leftmost a with an X, then causes the read-write head to travel right to the first b, replacing it with a Y. When the Y is written, the machine enters a state q2 indicating that an a has been successfully paired with a b.

The next set of transitions reverses the direction until an X is encountered, repositions the read-write head over the leftmost a, and returns control to the initial state.

δ(q2,Y) = (q2,Y,L),
δ(q2,a) = (q2,a,L),
δ(q2,X) = (q0,X,R),

We are now back at the initial state q0, ready to deal with the next input a and b. After one pass through this part of the computation, the machine will have carried out the partial computation.

q0aa.....abb.....b $\vdash^*$ Xq0a.....aYb.....b

After two passes we will have completed the partial computation:
q0aa.....abb.....b $\vdash^*$ XXq0.....aYY.....b
and so on, indicating that the matching process is being carried out properly.

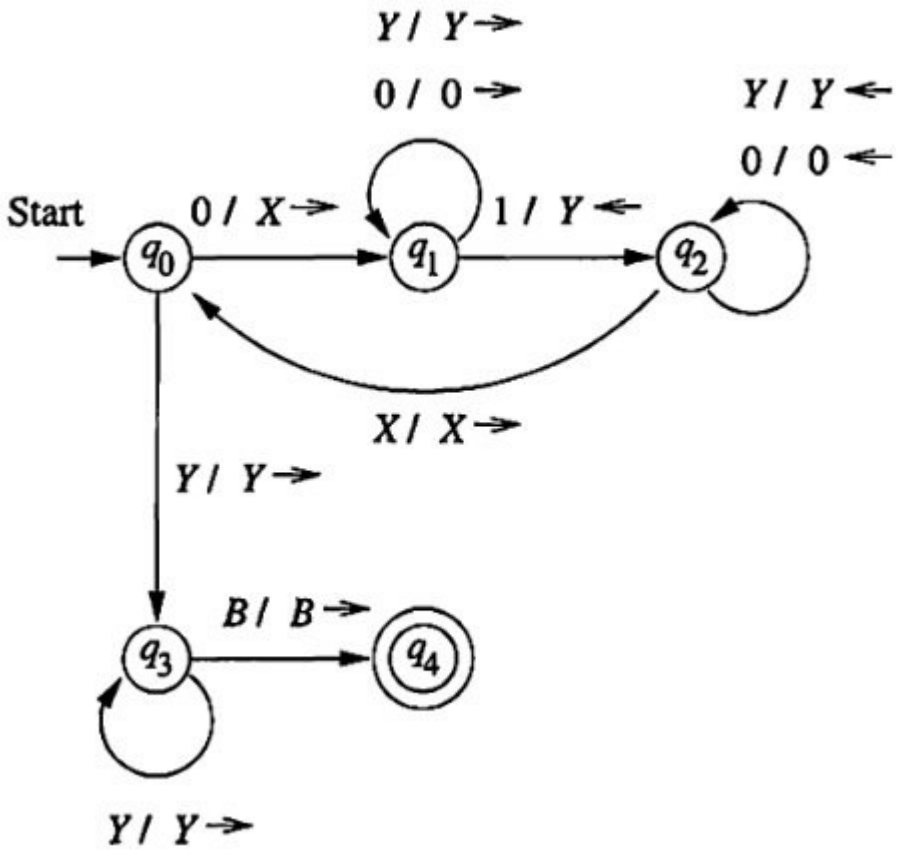
When the input is a string $a^n b^n$, the rewriting continues this way, stopping only when there are no more a's to be erased. When looking for the leftmost a, the read-write head travels left with the machine in state q2. When an x is encountered, the direction is reversed to get the a. But now, instead of finding an

a it will find a y. To terminate, a final check is made to see if all a's and b's have been replaced (to detect input where an a follows a b). This can be done by:

$\delta(q0,Y) = (q3,Y, R)$
$\delta(q3,Y) = (q3,Y,R)$
$\delta(q3,B) = (q4,B, R)$

The particular input aabb gives the following successive instantaneous descriptions.

q0aabb ├ Xq1abb ├ Xaq1bb ├ Xq2aYb ├ Xq0aYb ├ XXq1Yb ├ XXYq1b ├ XXq2YY ├ Xq2XYY ├ XXq0YY ├ XXYq3Y ├ XXYq3Y ├ XXYYq3B ├ XXYYBq4B



Transition diagram for a TM that accepts strings of the form 0^n1^n
Replace 0 with a and 1 with b for a^nb^n

Design a TM to accept the language L = {$a^n b^n c^n$ | n > 0}

On High level description

On input string w
    while there are unmarked a's, do
        Mark the left most a with X
        Scan right to reach the leftmost unmarked b;
            if there is no such b then crash
        Mark the leftmost b with Y
        Scan right to reach the leftmost unmarked c;
            if there is no such c then crash
        Mark the leftmost c with Z
    done
    Check to see that there are no unmarked a's or b's and c's.
        if there are then crash
    accept

Let TM be M =*(Q, Σ, τ, δ, q0, B, F)*
Where *δ,* is as follows:

$\delta(q_0,a)=(q_1,X,R)$
$\delta(q_1,a)=(q_1,a,R)$
$\delta(q_1,Y)=(q_1,Y,R)$
$\delta(q_1,b)=(q_2,Y,R)$
$\delta(q_2,b)=(q_2,b,R)$
$\delta(q_2,Z)=(q_2,Z,R)$
$\delta(q_2,c)=(q_3,Z,L)$

These above set of transition function replaces leftmost a by X, b by Y and c by Z.

$\delta(q_3,Z)=(q_3,Z,L)$
$\delta(q_3,Y)=(q_3,Y,L)$
$\delta(q_3,b)=(q_3,b,L)$
$\delta(q_3,c)=(q_3,c,L)$
$\delta(q_3,X)=(q_0,X,R)$

These above transition function reverses the direction until an X is encountered. Reposition the read-write head over the leftmost 'a' and returns the control to the initial state.

$\delta(q_0,Y)=(q_4,Y,R)$
$\delta(q_4,Y)=(q_4,Y,R)$
$\delta(q_4,Z)=(q_4,Z,R)$
$\delta(q_4,B)=(q_{acc},B,R)$

When looking for the leftmost 'a' the read-write head travels left with the machine in state q3, when an X is encountered, the direction is reversed to get the a. But now, instead of finding on a, it will find a 'Y'. to terminate, a final check is made to see it all a's , b's and c's have been replaced.

Here Q={q0,q1,q2,q3,q4,qcc}
Σ={a, b, c}
τ = {a, b, c, X, Y, Z, B}
F= { q_{acc}}

Transition Diagram:

Construct a Turing Machine accepting a language of palindrome over {a,b}* with each string of even length.

The TM scans the first symbol of input tape (a or b), erases , that is replaces it with Blank symbol B and changes state (q1, or q2). TM scans the remaining part without changing the tape symbol until it encounters blank symbol B. Then the read write head moves to the left. If the rightmost symbol tallies with the leftmost symbol (Which can be erased but remembered), the rightmost symbol is erased otherwise TM halts. The read write head moves to the left until blank B is encountered. The above steps are repeated after changing the states suitably.

The Transition functions are as shown below:

$\delta(q_0,a) = (q_1,B, R),$
$\delta(q_0,b) = (q_2,B, R),$

$\delta(q_1,a) = (q_1,a,R),$
$\delta(q_1,b) = (q_1,b,R),$
$\delta(q_1,B) = (q_3,B,L),$

$\delta(q_2,a) = (q_2,a,R),$
$\delta(q_2,b) = (q_2,b,R),$
$\delta(q_2,B) = (q_4,B,L),$

$\delta(q_3,a) = (q_5,B,L),$
$\delta(q_5,a) = (q_5,a,L),$
$\delta(q_5,b) = (q_5,b,L),$
$\delta(q_5,B) = (q_0,B,R),$

$\delta(q_4,b) = (q_6,B,L),$
$\delta(q_6,a) = (q_6,a,L),$
$\delta(q_6,b) = (q_6,b,L),$
$\delta(q_6,B) = (q_0,B,R),$

$\delta(q_0,B) = (q_7,B,R),$

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

Solution:

Given a function, f(x)=x+1. Here we represent input x on the tape by the number of I's on the tape. For example x=1, input will be BIB (note that B stands for blank in the input tape).

For x=2; input will be BIIB. Similarly, output can be seen by the number of I's on the tape. Let Turing machine TM be:

$M = (Q, \Sigma, \tau, \delta, q0, B, F)$

$Q = \{ q0, h\}$

$\tau = \{ I,B\}$

$\delta(q0,I) = (q0,I,R)$

$\delta(q0,B) = (q1,I,R)$

$\delta(q1,B) = (h,B,R)$

Now let x=3, then tape situation will be: BIIIB. Let us process the string, (Here we are processing the string from left to right).

$q_0IIIB \vdash^* IIIq_0B \vdash IIII \vdash IIIq_1B \vdash IIIIBhB$

Since the number of I in the input tape is 4, hence the output.

Example: *Prove that the following function is computable.*

f(n)=n+2.

Solution:

We know that if any function is computable, then there exists a Turing Machine for it. So, it will be sufficient to construct a TM to prove any function is computable.

Let Turing machine for the given function be:

$M = (Q, \Sigma, \tau, \delta, q0, B, F)$

$Q= \{q0,q1,q2,h\}$

$\tau = \{ B, I \}$

q0 is the initial state and halting state is represented by h.

Initially TM is in state q0, when it reads I it moves towards Right until it finds blank Symbol B. when q0 finds Blank B, it changes to state q1 and blank is replaced by I and moves right. At state q2, B is replaced by I and machine halts after moving right.

It is clarified by the following transition functions:

$\delta(q0,I) = (q0,I, R)$

$\delta(q0,B) = (q1,I, R)$

$\delta(q1,B) = (q2,I,R)$

$\delta(q2,B) = (h,B,R)$

Let n=3, that is BIIIB

TM behaves as follows:

$q_0IIIB \vdash^* IIIq_0B \vdash IIIIq_1B \vdash^* IIIIIBhB$

Variant of Turing Machine:

1. Two-way infinite tape:

A Turing machine with a two-way infinite tape is denoted by: $M = (Q, \Sigma, \tau, \delta, q0, B, F)$ as in the original model. As its name implies, the tape is infinite to the left as well as to the right. We denote an ID of such a device as for the one-way infinite TM. We imagine, however that there is an infinity of blank cells both to the left and right of the current non blank portion of the tape.

2. Multi-tape Turing Machine:

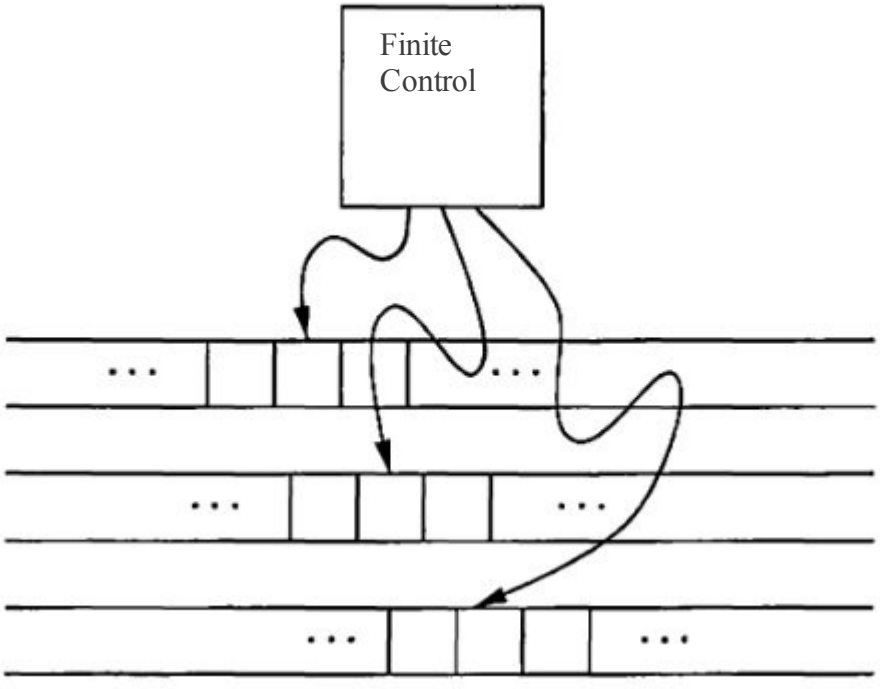


Fig: Multi-tape Turing Machine

A multi-tape Turing machine is shown in fig. It consists of a finite control with k tape heads and k tapes; each tape is infinite in both directions.

Each tape has its own head for reading/writing
Initially the input is on tape 1 and other are blank
Transition function allow for reading, writing, and moving the heads on all tapes simultaneously, i.e.,
$\delta: Q \times \tau^K \rightarrow Q \times \tau^K \times \{L \times R\}^K$
$\delta(q_i, a_1, a_2, \ldots\ldots\ldots\ldots a_k) = (q_j, b_1, b_2 \ldots\ldots\ldots\ldots\ldots b_k, L, R \ldots\ldots\ldots\ldots\ldots\ldots L)$ means that,
If the machine is in state q_i and heads 1 through k are reading symbols a1 through ak the machine goes to state q_j writes , b_1 through bk on tapes 1 through k respectively and moves each head to the left or right specified by δ.

Theorem: Show that every language accepted by a multi-tape Turing Machine is recursively enumerable.

3. Non Deterministic Turing Machine:

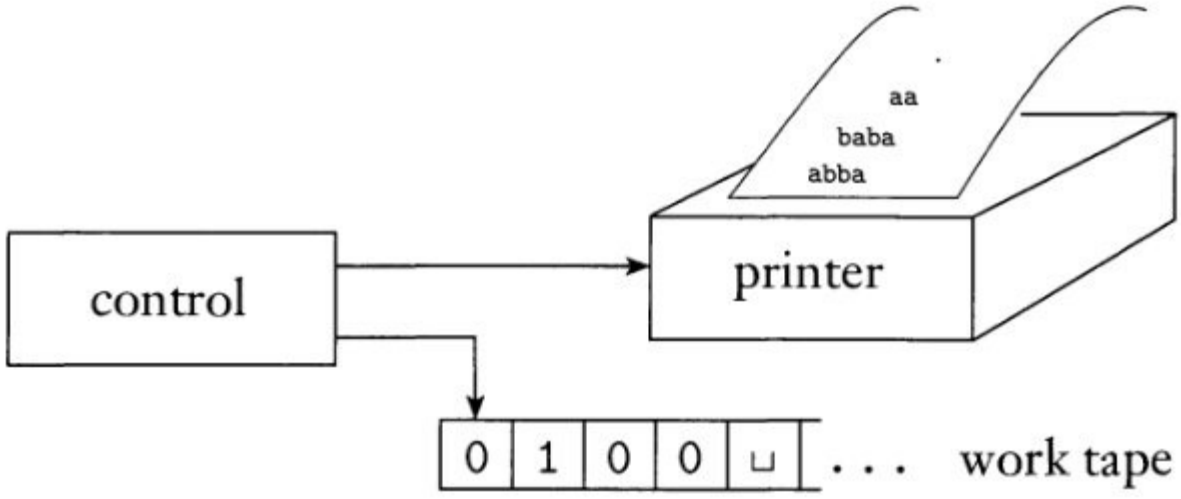
We can also imagine Turing machine that acts non-deterministically as in case of Finite Automata and push down automata. In standard Turing machine to each pair of current state(except the halt state) and the symbol being scanned, there is a unique triplet comprising the next state, unique action in terms of writing a symbol in the cell being scanned and the motion if any, to the right or left. However in case of Non-deterministic Turing machine (NDTM) , to each pair (q, a) with q as current state and a as symbol being scanned, there may be a finite set of triplets $\{ (q_i, a_i, m_i); i=1,2....\}$ of possible moves. The set of triplets may be empty that is for some particular (q,a) the TM may not have the next move.

A non-deterministic TM is a 7-tuple. $M = (Q, \Sigma, \tau, \delta, q0, B, F)$
where each component has the same meaning as of the standard turing machine expect the transition function, δ:
$\delta: (Q X \tau) \rightarrow$ power set of $(Q X \tau X \{ L, R, N \})$

A language is decidable if and only if some non-deterministic TM decides. it.

4. Enumerators:

Some people use the term *recursively enumerable language* for Turing-recognizable language. That term originates from a type of Turing machine variant called an enumerator. Loosely defined, an enumerator is a Turing machine with an attached printer. The Turing machine can use that printer as an output device to print strings. Every time the Turing machine wants to add a string to the list, it sends the string to the printer. The following figure depicts a schematic of this model.



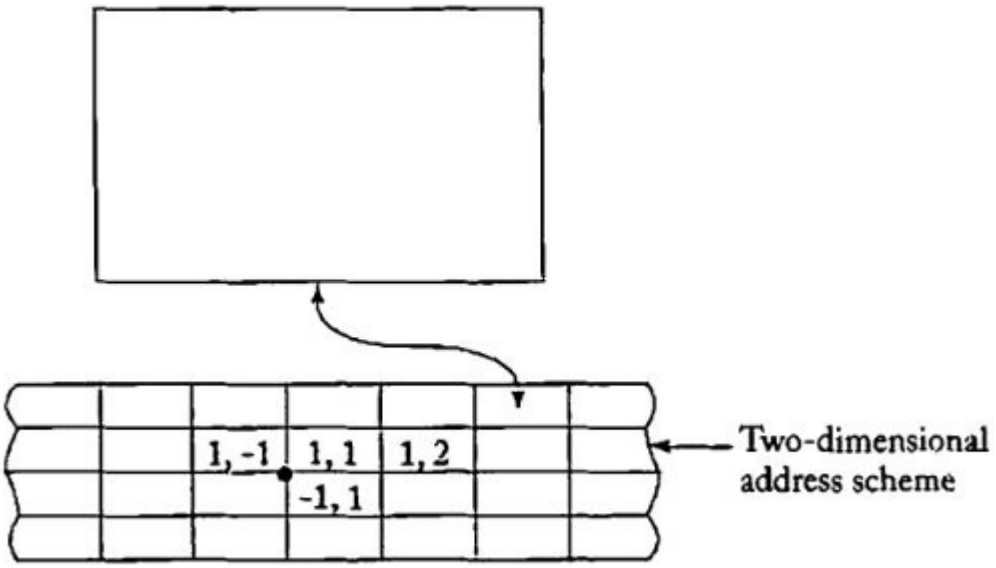
Schematic of an Enumerator

An enumerator E starts with a blank input tape. If the enumerator doesn't halt, it may print an infinite list of strings. The language enumerated by E is the collection of all the strings that it eventually prints out. Moreover, E may generate the strings of the language in any order, possibly with repetitions.

To enumerate a set means to list the elements one at a time. To say that a set is enumerable should mean that there is an algorithm for enumerating it, so far,we have seen that TM as recognizers of language and as computers of functions on the positive integers.. There is another purpose a TM can serve. It can be a generating device. A TM is said to be generating or enumerating if we are able to produce each word of any language L separated by blank symbol (initially tape is empty). The order of the string is not important and any string may be repeated indefinitely.

5. Multi-Dimensional Turing Machine:

A multidimensional Turing machine is one in which the tape can be viewed as extending infinitely in more than one dimension. A diagram of a two- dimensional Turing machine is shown in Figure below.
The formal definition of a two-dimensional Turing machine involves a transition function δ: of the form :
$\delta: Q \times \tau \rightarrow Q \times \tau \times \{L,R,U,D\}$, where U and D specify movement of the read-write head up and down.



Church's Thesis(Church's Hypothesis or Turing Thesis or.....)
In computability theory, the Church–Turing thesis (*also known as the Church–Turing conjecture, Church's thesis, Church's conjecture, and Turing's thesis)* is a combined hypothesis ("thesis") about the nature of functions whose values are effectively calculable; i.e. computable. In simple terms, it states that "everything computable is computable by a Turing machine."

It is surprising questions that, what functions cannot be computed by the Turing Machine. It is believed that there are no functions that can be defined by Humans, (whose calculations can be described by any well defined mathematical algorithm that people can be taught to perform) that cannot be computed by Turing Machine. That the TM is believed to be the ultimate calculating mechanism.

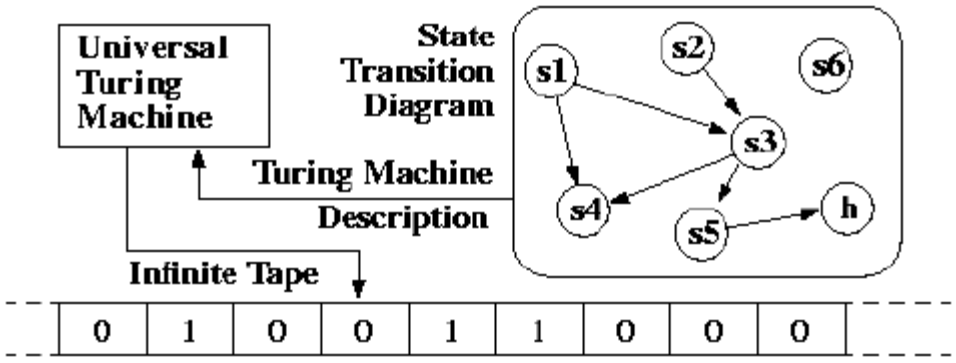
In other words Church's Thesis state that "*No computational procedure will be considered an algorithm unless it can be represented as a Turing Machine.*

Universal Turing Machines

A general purpose Turing machine is usually called Universal Turing Machine which is powerful enough to simulate the behavior of any computer, including any Turing Machine itself. More precisely, Universal Turing machine can simulate the behavior of an arbitrary Turing Machine, over any Σ.

In computer science, a universal Turing machine (UTM) is a Turing machine that can simulate an arbitrary Turing machine on arbitrary input. The universal machine essentially achieves this by reading both the description of machine to be simulated as well as the input thereof from its own tape. Alan Turing introduced this machine in 1936–1937. This model is considered by some (for example,Martin Davis (2000)) to be the origin of the stored program computer—used by John von Neumann (1946) for the "Electronic Computing Instrument" that now bears von Neumann's name: the von Neumann architecture. It is also known as universal computing machine,universal machine, machine U, U.

Universal Turing Machine 'U' takes two arguments, a description of a machine TM, and a description of an input string w, 'w'.
U ("m", "w") = "m(w)" Its the functional notation of the universal Turing Machine.



The most striking positive result concerning the capabilities of Turing machines is the existence of *Universal Turing Machines* (UTM). When started on a tape containing the encoding of another Turing machine, call it *T*, followed by the input to *T*, a UTM produces the same result as *T* would when started on that input. Essentially a UTM can simulate the behavior of any Turing machine (including itself).

One way to think of a UTM is as a programmable computer. When a UTM is given a program (a description of another machine), it makes itself behave as if it were that machine while processing the input.

Note again, our identification of input-output equivalence with "behaving identically". A machine *T* working on input *t* is likely to execute far fewer transitions that a UTM simulating *T* working on *t*, but for our purposes this fact is irrelevant.

In order to design such a machine, it is first necessary to define a way of representing a Turing machine on the tape for the UTM to process. To do this we will recall that Turing machines are formally represented as a collection of 4-tuples. We will first design an encoding for individual tuples, and then for sequences of tuples.

In the examples that we've seen up to this point, each problem required us to build a special-purpose Turing machine to solve only that problem. Is there a more general Turing machine that acts like a general-purpose computer? The answer is yes. We'll see that a Turing machine can be built to interpret any other Turing machine. In other words, there is a Turing machine that can take as input an arbitrary Turing machine M together with an arbitrary input for M and then perform the execution of M on its input. Such a machine is called a universal Turing machine. A universal Turing machine acts like a general purpose computer that stores a program and its data in memory and then executes the program.

Turing recognizable and Turing-decidable:
A language is Turing-recognizable (or recursively enumerable) if there is a Turing machine which recognizes it.

A language is Turing-decidable (or recursive) if there is a Turing machine which recognizes it which halts for all inputs.

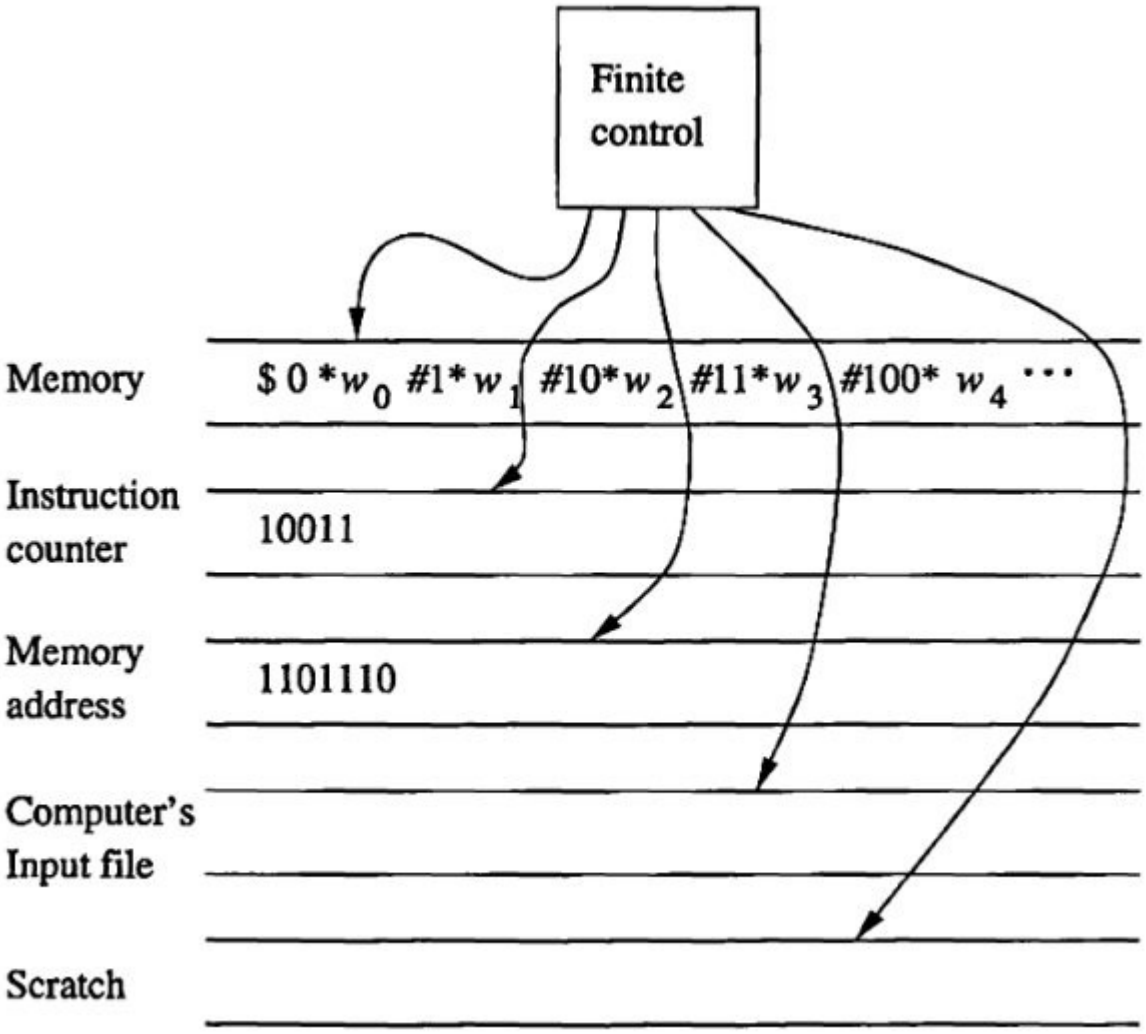
Simulating a Computer by a Turing Machine:
To begin our study of how a TM simulates a computer, let us give a realistic but informal model of how a typical computer operates.

1. First, we shall suppose that the storage of a computer consists of an indefinitely long sequence of words by each with an address. In a real computer, words might be 32 or 64 bits long, but we shall not put a limit on the length of a given word. Addresses will be assumed to be integers 0, 1, 2 and so on. We shall assume there is no limit to the number of words.

2. We assume that the program of the computer is stored in some of the words of memory. These words each represent a simple instruction, as in the machine or assembly language of a typical computer. Examples are instructions that move data from one word to another or that add one word to another.
3. We assume that each instruction involves a limited (finite) number of words, and that each instruction changes the value of at most one word.
4. A typical computer has registers^ which are memory words with especially fast access. Often, operations such as addition are restricted to occur in registers. We shall not make any such restrictions, but will allow any operation to be performed on any word.

Figure below shows how the Turing machine would be designed to simulate a computer. This TM uses several tapes, but it could be converted to a one-tape TM. The first tape represents the entire memory of the computer. We have used a code in which addresses of memory words, in numerical order, alternate with the contents of those memory words. Both addresses and contents are written in binary. The marker symbols * and # are used to make it easy to find the ends of addresses and contents, and to tell whether a binary string is an address or contents. Another marker, $, indicates the beginning of the sequence of addresses and contents. The second tape is the "instruction counter," This tape holds one integer in binary, which represents one of the memory locations on tape 1, The value stored in this

location will be interpreted as the next computer instruction to be executed. The third tape holds a "memory address" or the contents of that address after the address has been located on tape 1. To execute an instruction, the TM must find the contents of one or more memory addresses that hold data



involved in the computation. First, the desired address is copied onto tape 3 and compared with the addresses on tape 1, until a match is found. The contents of this address is copied onto the third tape and moved to wherever it is needed, typically to one of the low-numbered addresses that represent the registers of the computer.