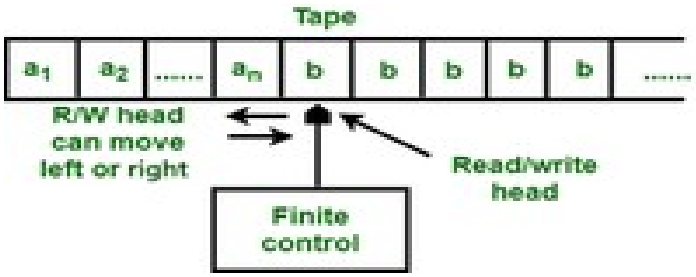
UNIT-5 TURING MACHINES

Turing Machine:

- A Turing machine is a device that manipulates symbols on a strip of tape according to a *table of rules.*
- Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside a computer.
- The "Turing" machine was described in 1936 by Alan Turing who called it an "a-machine" (automatic machine).
- The Turing machine is not intended as practical computing technology, but rather as a hypothetical device representing a computing machine. Turing machines help computer scientists understand the limits of mechanical computation.

A Turing machine consists of:



• A **tape** which is divided into cells. Each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol (written as 'B') and one or more other symbols.

The tape is assumed to be arbitrarily extendable to the left and to the right, i.e., the Turing machine is always supplied with as much tape as it needs for its computation. Cells that have not been written to before are assumed to be filled with the blank symbol.

• A **head** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. In some models the head moves and the tape is stationary.

• **Control Unit:** The reading/writing from/to the tape is determined by the control unit. The different moves performed by the machine depend on the current scanned symbol and the current state. A state register that stores the state of the Turing machine, one of finitely many. *There is one special start state with which the state register is initialized.*

The various moves performed by the machine are:

1. Change of state from one stare to another state.
2. The symbol pointing to by the read-write head can be replaced by another symbol.
3. The read-write head may move either towards left or towards right.

Some of the characteristics of a Turing machine are:

1. The symbols can be both read from the tape and written on it.
2. The TM head can move in either directions – Left or Right.
3. The tape is of infinite length.
4. The special states, Halting states and Accepting states, take immediate effect.
5. Either erase or write a symbol.

NOTE: A Turing machine can recognize all types of languages viz, RL, CFL, CSL. Apart from these languages, the Turing machine also accepts the language generated from *unrestricted grammar*. Thus it can accept any generalized language.

DEFINITION OF TURING MACHINE:

A Turing machine can be defined as follows:

Definition : The Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

Q is set of finite states

Σ is set of input alphabets

Γ is set of tape symbols

δ is transition function $Q \times \Gamma$ to $(Q \times \Gamma \times \{L, R\})$

q_0 is the initial state

B is a special symbol indicating blank character

$F \subseteq Q$ is set of final states.

TRANSITION FUNCTION OF TURING MACHINE:

The "Transition Function" for Turing Machine is given by

$\delta : Q X \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})$

When the machine is in a given state (Q) and reads a given symbol (Γ) from the tape, it replaces the symbol on the tape with some other symbol (Γ), goes to some other state (Q), and moves the tape head one square left (L) or right (R).

REPRESENTATION OF TURING MACHINES:

The Turing machine can be represented using the following notations:

1. Transition table
2. Instantaneous description
3. Transition diagram

REPRESENTATION BY INSTANTANEOUS DESCRIPTIONS (ID):

An ID of a Turing machine 'M' is defined in terms of the entire input string and the current state

An "Instantaneous Description" or "Configuration" of a Turing machine requires.

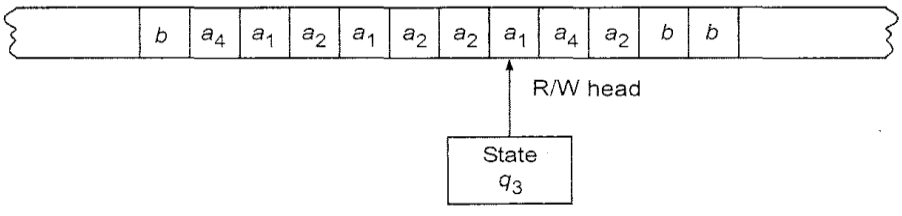
(a) the state the Turing machine is in

(b) the contents of the tape

(c) the position of the tape head on the tape.

This is written as a string of the form $X_i \ldots\ldots\ldots\ldots\ldots X_j\ q_m X_k \ldots\ldots\ldots\ldots\ldots X_l$

Where the x's are the symbols on the tape q_m is the current state, and the tape head is on the square containing x_k (the symbol immediately following q_m).



A snapshot of Turing machine.

The present symbol under the R/W head is a1. The present state is q3. So a1 is written to the right of q3. The nonblank symbols to the left of a1 form the string a4a1a2a1a2, which is written to the left of q3. The sequence of nonblank symbols to right of a1 is a4a2. Thus ID of M is as given below.

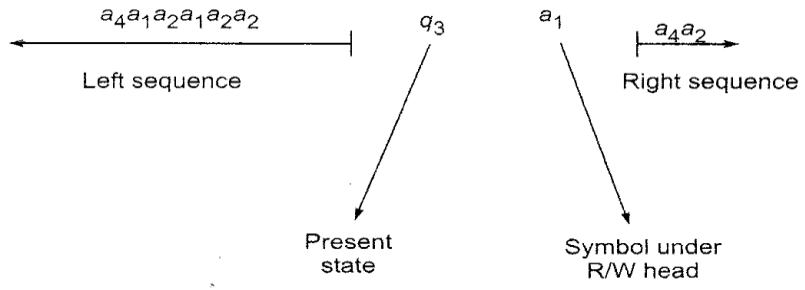


Fig. 9.3 Representation of ID.

From the above figure the ID of M is written as: $a_4a_1a_2a_1a_2$ **q_3 a_1** a_4a_2

REPRESENTATION BY TRANSITION TABLE

δ	Tape Symbols (Γ)				
States	a	b	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	(q_1, a, R)	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	(q_2, a, L)	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Consider an example TM with 5 states. The tape symbols are (a,b,X,Y,Z,B)
Q={q0,q1,q2,q3,q4} ; Σ={a,b}; Γ ={a,b,X,Y,B) ;
q0 is the initial state; B is the indicating blank character;
F={q4}, which is the final state .

1. The undefined entries indicate that there are no transitions defined or there can be a transition to dead state. (When there is a transition to the dead state, the machine halts and the input string is rejected by the machine).
2. A finite table (occasionally called an action table) of instructions usually quintuples [5-tuples] :

If $\delta(q, a_i)=(p, a_j, d_k)$ we write $p\ a_{j1}\ d_k$ under a_j–column and q-row.

- 'q' gives the state the machine currently in and
- symbol 'a_i' is the symbol currently under the head.
- 'p' denotes new state into which the TM enters
- a_j is written in the current cell.
- d_k gives the movement of head (L or R or S).

Note :The symbol |-- is used to represent the move. |--*-- represents 'sequence of moves' or 'reflexive transitive closure' of the relation |---

Let δ(q,xi)=(p,y,R) then,

$x_1x_2.....x_{i-1}\ q\ x_i,x_{i+1}.....x_n$ |--- $x_1x_2.....x_{i-1}\ y\ p\ x_{i+1}.....x_n$

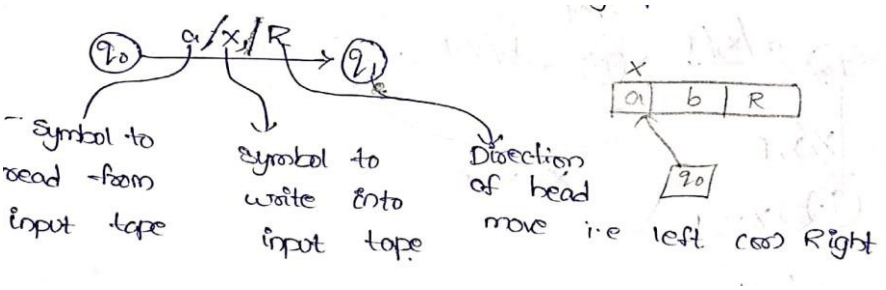
For example if δ(q5 , b)= (q8 , c, R) then a possible move can be abbab**q5b**abb |– abbab**cq8**abb

Let δ(q,xi)=(p,y,L) then,

$x_1x_2.....x_{i-1}\ q\ x_i,x_{i+1}.....x_n$ |--- $x_1x_2.....x_{i-2}\ px_{i-1}y\ x_{i+1}.....x_n$

REPRESENTATION BY TRANSITION DIAGRAM:

- The states are represented by vertices.
- Directed edges are used to represent transition of states.
- The labels are triples of the form (α,β,ϒ). When there is a direct edge from q_i to q_j with labels (α,β,ϒ) it means that $\delta(q_i,\alpha)= (q_j,\beta,\Upsilon)$ α=symbol to read from input tape, β=symbol to write into input tape, ϒ= direction of head move right or left.



THE LANGUAGE ACCEPTANCE BY A TURING MACHINE:

The language of a Turing Machine M, L(M), is the set of strings accepted by M.
L(M) = {x | M halts and accepts x}. Turing machines are defined so that they can accept by halting on given input or by entering into recursive loop for invalid input string.

M accepts w iff the execution of M on w is terminating and ends in the accept state: $<\varepsilon, q_0, w> \vdash <l, q, r> \forall$ q declared as accepting state.

M rejects w iff the execution of M on w is terminating and ends in the non-accepting state: $<\varepsilon, q_0, w> \vdash <l, q, r> \forall$ q declared as non-accepting state.

M does not accept w iff M rejects w or M loops on w.

A Turing Machine accepts its input if it halts in a final state. There are two ways this could fail to happen:
(a) The Turing machine could halt in a nonfinal state or
(b) The Turing machine could never stop i.e., it enters an “infinite loop”.

DESIGN OF TURING MACHINES:

Follwing are the basic guidelines for designing a Turing machine.
(i) The fundamental objective in scanning a symbol by the R/W head is to 'knoow’ what to do in the future. The machine must remember the past symbols scanned. The Turing machine can remember this by going to the next unique state.
(ii) The number of states must be minimized. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of the R/W head.

TECHNIQUES FOR TM CONSTRUCTION

The following are some high-level conceptual tools to make the construction of TMs easier:

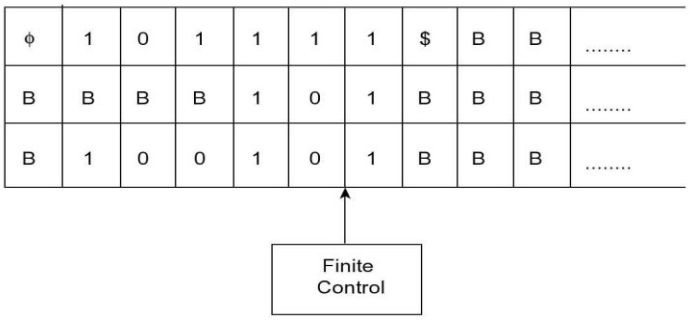
(a) Storage in Finite Control

- We are using a state, whether it is of a FA or pda or TM, to 'remember' things. We can use a state to store a symbol as well.
- So the state becomes a pair (q, a) where q is the state (in the usual sense) and ‘a’ is the tape symbol stored in (q, a). So the new set of states becomes Q x Γ.
- To account for this modification, we can define the Turing machine as M=(Q,Σ , Γ,δ ,[q0,B],B,F) where Q is of the form [q0,a] where q is a state, aϵΣ.

- The transitions are defined by ([QxΣ], Γ)->([QxΣ], Γ,{R/L}).
- For example, the transition δ([q,a],b)=([p,b],c,R) indicates that the control is in state q, and a is stored in finite control. On the input symbol b, it moves top state, changes the symbol in finite control to b, changes the cell content as c and moves one cell right.

(b) Multi-track Tape:

The tape is imagined as divided into cells where the input ot be processed is placed. We can further imagine that the tape is divided into k- tracks for some finite number k as shown below.



The reading head considers k symbols belonging to different tracks in same column and processes it. There are two special symbols Ø and $ used in the first track and they indicate the boundary of the input. The other tracks are used to place the intermediate results and the final result of the processing. The blank input is identified as 'all B's in all tracks', that is as [B,B,B]. the input at the current position of the reading head is [1,1,1].

(c) Checking off Symbols

This is one useful technique that can be used to visualize how TM would recognize the languages.

This technique uses an extra track that indicates the symbol on the other track is processed. The languages which have repeated strings or some conditions relating to other part of string can be solved with this procedure. Such languages are listed below.

i) {ww\w in Σ* }

ii) {ww^R/ w in Σ*}

iii) {wcw/ w in Σ*}

For the languages mentioned above, we can use the tape with two tracks where on one track, we place the given input, and on the other track, we place either B or √. If the upper track symbol is B, it indicates the symbol on lower track is not considered. If the symbol on upper track is √, it indicates that the symbol on the lower track is considered.

(d) Subroutines

- A Turing machine simulate any type of subroutines found in programming languages, including recursive procedures, and any of the known parameter passing mechanisms.
- We can design a TM program to serve as a subroutine.
- It has designated initial state and a return state. There states are used to indicate the call to a subroutine and return to called subroutine.
- First a TM program for the subroutine is written. This will have an initial state and a 'return' state. After reaching the return state. there is a temporary halt. For using a subroutine, new states are introduced. When there is a need for calling the subroutine, moves are effected to enter the initial state for the subroutine (when the return state of the subroutine is reached) and to return to the main program of TM.
- As an example, a TM designed to accept strings with balanced parenthesis.

(e) Shifting Over

A Turing machine can make space on its tape by shifting non-blank symbols by a finite number of cells to the right. To perform this operation, we can use the state with a small amount of storage. This storage space is used to store the symbols and then replacing the current cell on tape by blank, and to move right. Read the right symbol and replace it with the symbol stored in the finite control. To perform this operation without losing the data, it requires storage capacity to store at least two symbols.

TYPES OF TURING MACHINES:

1. Turing machine with Two - Way Infinite Tape :

This is a TM that have one finite control and one tape which extends infinitely in both directions.

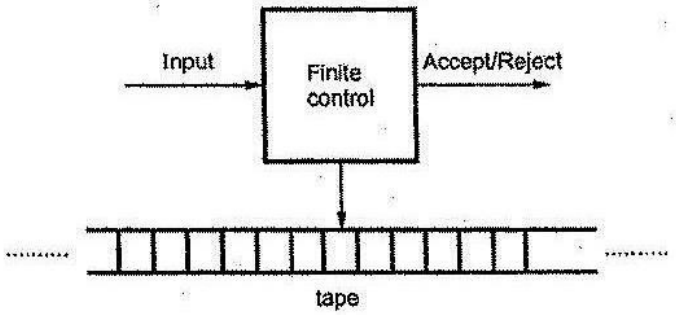


FIGURE : TM with infinite Tape

It turns out that this type of Turing machines are as powerful as one tape Turing machines whose tape has a left end.

2. Multiple Turing Machines :

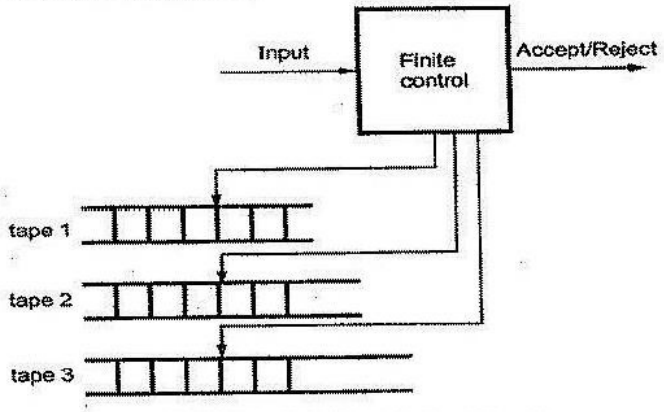


FIGURE : Multiple Turing Machines

A multiple Turing machine consists of a finite control with k tape heads and k tapes, each tape is infinite in both directions. On a single move depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can

1. Change state.
2. Print a new symbol on each of the cells scanned by its tape heads.
3. Move each of its tape heads, independently, one cell to the left or right or keep it stationary.

Initially, the input appears on the first tape and the other tapes are blank.

3. Nondeterministic Turing Machines :

A nondeterministic Turing machine is a device with a finite control and a single, one way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. Note that the non deterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and / or direction of head motion are selected from other choices. The non deterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

As with the finite automaton, the addition of nondeterminism to the Turing machine does not allow the device to accept new languages.

4. Multidimensional Turing Machines :

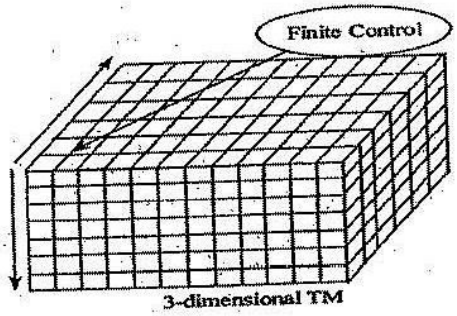


FIGURE : Multidimensional Turing Machine

The multidimensional Turing machine has the usual finite control, but the tape consists of a k - dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state and symbol scanned, the device changes state, prints a new symbol, and moves its tape head in one of 2 k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input.At any time, only a finite number of rows in any dimension contains nonblank symbols, and these rows each have only a finite number of nonblank symbols

5. Multihead Turing Machines :

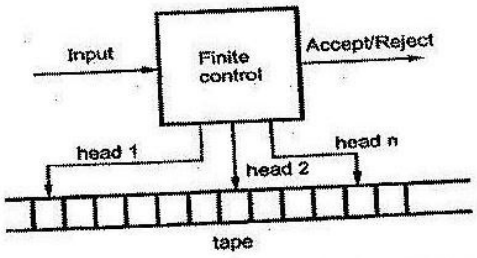


FIGURE : Multihead Turing Machine

A k - head Turing machine has some fixed number, k, of heads. The heads are numbered 1 through k, and a move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may each move independently left, right or remain stationary.

6. Off - Line Turing Machines :

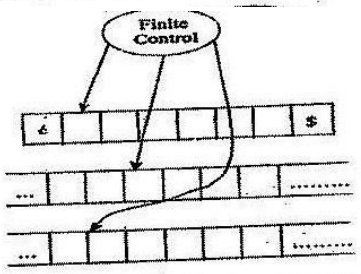
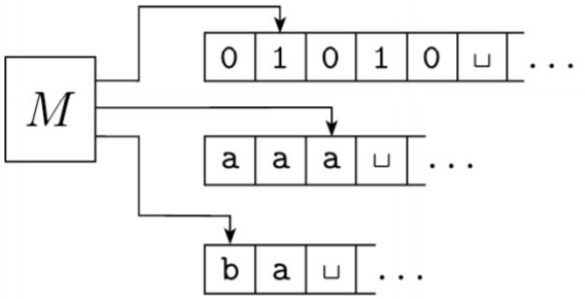


FIGURE : Off - line Turing Machine

An "Offline Turing Machine" has two tapes. One tape is read-only and contains the input, the other is read-write and is initially blank. Offline Turing machines are equivalent to Standard TMs".

7. Multi-tape Multi-head Turing Machine:



- The multi-tape Turing machine has multiple tapes and multiple heads
- Each tape controlled by separate head
- Multi-Tape Multi-head Turing machine can be simulated by standard Turing

NON-DETERMINISTIC TURING MACHINE:

A non-deterministic Turing machine has a single, one way infinite tape.

For a given state and input symbol has at least one choice to move (finite number of choices for the next move), each choice several choices of path that it might follow for a given input string.

A non-deterministic Turing machine is equivalent to deterministic Turing machine.

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Universal Turing Machine

A universal Turing machine M takes as input a Turing machine T and an input x to T and simulates (interprets) T on input x. Thus, M halts on input (T, x) iff T halts on input x, and sometimes it is also convenient to say that the output of M on input (T, x) is the same as the output of T on x.

A universal TM M_u is an automaton that, given as input the description of any TM M and a string w, can simulate the computation of M for input w. To construct such a M_u, we first choose a standard way of describing TMs. We may, without loss of generality, assume that M = (Q, {0, 1}, {0, 1, B}, d, q_1, B, q_2) where Q = {q_1, q_2, … q_n}, q_1 the initial state, and q_2 the single final state. The alphabet {0, 1, B} ∈ Γ are represented as a_1, a_2 and a_3. The directions left and right are represented as D_1 and D_2, respectively. The transitions of TM are encoded in a special binary representation where each symbol is separated by 1. For example, if there is a transition

the binary representation for the transition is given as

$$0^i10^j10^k10^l10^m$$

The binary code for the Turing machine M that has transitions $t_1, t_2, t_3 \ldots t_n$ is represented as

$$111\ t_1\ 11\ t_2\ 11\ t_3\ 11 \ldots 11\ t_n\ 111$$

COUNTER MACHINES:

A counter machine is an abstract machine used in formal logic and theoretical computer science to model computation. A counter machine comprises a set of one or more unbounded registers, eah of which can hold a single on-negative integer, and a list of arithmetic and control instructions for the machine to follow.

Def: A counter machine consists of

1. A finite set of registers $r_0, r_1, \ldots r_n$ where each register is labelled and can hold any single non-negative integer.
2. A special register to identify the current instruction to be executed is maintained and is called the state register.

3. A finite set of instructions I_1,….I_m where each instruction is in the same physical space as the registers.

RESTRICTED TURING MACHINES:

1. TM with semi-infinite tapes
2. Multistack turing machine
3. Counter machines

CHURCH'S THESIS:

- A Turing machine that is able to simulate any other Turing machine is called a universal Turing machine (UTM, or simply a universal machine).
- A more mathematically oriented definition with a similar "universal" nature was introduced by Alonzo Church, whose work on lambda calculus intertwined with Turing's in a formal theory of computation known as the Church–Turing thesis.

Def: A computation process that can be represented by an algorithm can be converted to a Turing Machine. In simple words, anything that can be done by an algorithm can be done by a Turing Machine as well. So, all algorithms can be implemented in a Turing Machine.

- Any mechanical computation can be performed by a Turing Machine
- There is a TM-n corresponding to every computable problem
- We can model any mechanical computer with a TM
- The set of languages that can be decided by a TM is identical to the set of languages that can be decided by any mechanical computing machine
- If there is no TM that decides problem P, there is no algorithm that solves problem P.

Church's original statement: "Any machine that can do certain list of operations will be able to perform all algorithms".

Church tied both "recursive functions" and "computable functions" together. Every partial recursive function is computable on TM. Computer models such RAM also give rise to partial recursive functions. So they can be simulated on TM which confirms the validity of churches hypothesis.

Important Church's hypothesis:

- First we will prove certain problems which cannot be solved using TM.
- If Churches thesis is true this implies that problems cannot be solved by any computer or any programming languages we might every develop.
- Thus in studying the capabilities and limitations of TM we are indeed studying the fundamental capabilities and limitations of any computational device we might even construct.

The applications of Church Turing Thesis are as follows:

- Church Turing Thesis is used to define an Algorithm (traditionally)
- Used in solving 10th Problem by Hilbert.
- Used in defining modern computing devices including Molecular and Quantum Computers.

Examples

- Lambda Calculus is equivalent to TM
- Unrestricted replacement grammars are equivalent to TM
- Equivalence of TM and 2-stack deterministic PDA + ε-transitions

HALTING PROBLEM:

The halting problem is decision problem which is informally stated as follows:

"Given a description of an algorithm and a description of its initial arguments, determine whether the algorithm, when executed with these arguments, ever halts. The alternative is that a given algorithm runs forever without halting."

Alan Turing proved in 1936 that there is no general method or algorithm which can solve the halting problem for all possible inputs.

The HP asks the question: Given a program and an input to the program, determine if the program will eventually stop when it is given that input?

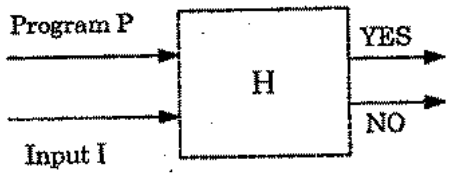
To find the solution of HP- Let the program run with the given input and if the program stops and we conclude that problem is solved. But, if the program does not stop in a reasonable amount of time, we cannot conclude that it won't stop.

Theorem: HP is un-decidable.

Proof: Initially, we assume that HP is decidable and the algorithm (solution) for HP is H. the halting problem solution H takes two inputs:

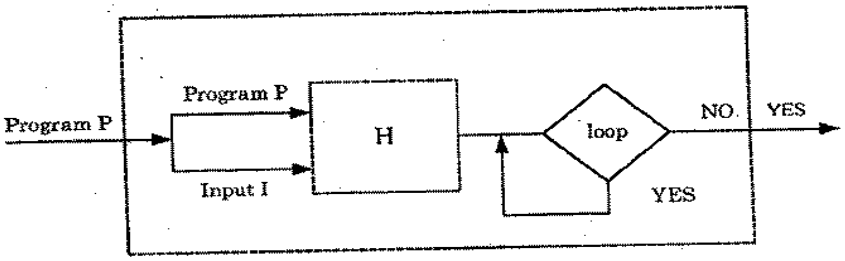
1. Description of TM 'M' i.e program P and
2. Input I for the program P.

H generates and output "YES" if H determines that P stops on input I or it outputs "NO" if H determines that P loops as shown below.



Figure(a)

Now H can be modified to take P as both inputs (the program and its input) and H should be able to determine if P will halt on P as it's input shown in below figure.



Figure(b)

Let us construct a new, simple algorithm Q that takes output of H as its input and does the following:

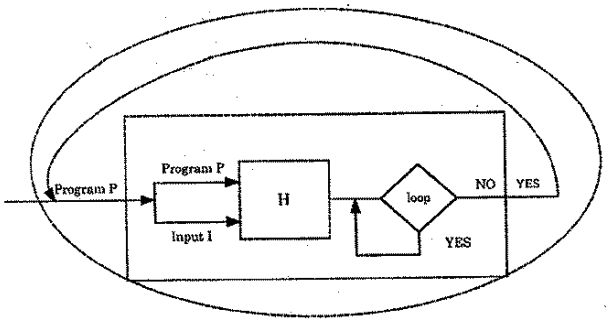
1. If H outputs "NO" then Q outputs "YES" and halts.
2. Otherwise H's output "YES" causes Q to loop forever

If means, Q does the opposite what H does.

We define Q as follows :

```
Function Q( )
{
   if ( Function H ( ) = " NO")
   {
         return ("YES");
   }
   else
   {
         while (1);              // Loop for ever
   }
}        // End of the function Q
```

Since, Q is a program, now let us use Q as the input to itself as shown in figure(c).



Figure(c)

Now, we analyse the following :

1. If H outputs "YES" and says that Q halts then Q itself would loop (that's how we constructed it).
2. If H outputs "NO" and says that Q loops then Q outputs "YES" and will halts.

Since , in either case H gives the wrong answer for Q. Therefore, H cannot work in all cases and hence can't answer right for all the inputs. This contradicts our assumption made earlier for HP. Hence, HP is undecidable.

Implications of Halting Problem: The Theorem of "Halting Problem" does not say that we can never determine whether or not a given program halts on a given input. Most of the times, a "meta-program" is used to check another program for potential infinite loops, and get this meta-program to work most of the time.

The theorem says that we cannot ever write such a meta-program and have it work all of the time. This result is also used to demonstrate that certain other programs are also impossible. The basic outline is as follows:

(i) If we could solve a problem X, we could solve the Halting problem

(ii) We cannot solve the Halting Problem

(iii) Therefore, we cannot solve problem X

RECURSIVELY ENUMERABLE LANGUAGES

A language L over the alphabet Σ is called recursively enumerable if there is a TM M that accept every word in L and either rejects (crashes) or loops for every word in language L' the complement of L.

Accept (M) = L

Reject (M) + Loop (M) = L'

When TM M is still running on some input (of recursively enumerable languages) we can never tell whether M will eventually accept if we let it run for long time or M will run forever (in loop).

Example : Consider a language (a + b) * bb (a + b) *.

TM for this language is ,

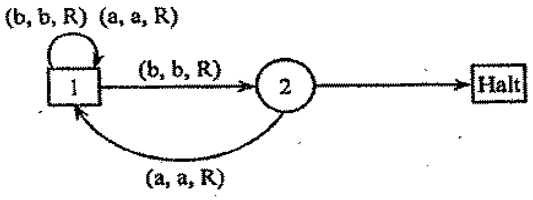


FIGURE : Turing Machine for (a + b) * bb (a + b) *

Here the inputs are of three types.

1. All words with bb = accepts (M) as soon as TM sees two consecutive b's it halts.
2. All strings without bb but ending in b = rejects (M). When TM sees a single b, it enters state2. If the string is ending with b, TM will halt at state 2 which is not accepting state. Hence it is rejected.
3. All strings without bb ending in 'a' or blank 'B' = loop (M) here when the TM sees last a it enters state 1. In this state on blank symbol it loops forever.

Recursive Language

A language L over the alphabet Σ is called recursive if there is a TM M that accepts every word in L and rejects every word in L' i. e.,

accept (M) = L
reject (M) = L'
loop (M) = ϕ.

Example :Consider a language b (a + b) * . It is represented by TM as :

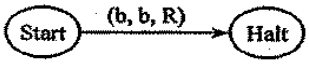


FIGURE : Turing Machine for b (a + b) *

This TM accepts all words beginning with 'b' because it enters halt state and it rejects all words beginning with a because it remains in start state which is not accepting state.

A language accepted by a TM is said to be recursively enumerable languages. The subclass of recursively enumberable sets (r. e) are those languages of this class are said to be recursive sets or recursive language.

LINEAR BOUNDED AUTOMATA

The Linear Bounded Automata (LBA) is a model which was originally developed as a model for actual computers rather than model for computational process. A linear bounded automaton is a restricted form of a non deterministic Turing machine.

A linear bounded automaton is a multitrack Turing machine which has only one tape and this tape is exactly of same length as that of input.

The linear bounded automaton (LBA) accepts the string in the similar manner as that of Turing machine does. For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of variable is bounded by its data type.

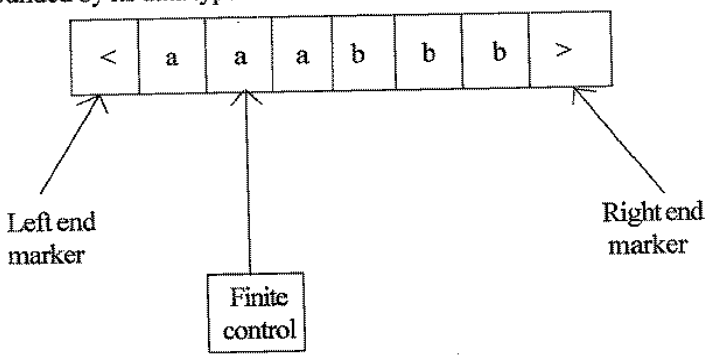


FIGURE : Linear bounded automaton

The LBA is powerful than NPDA but less powerful than Turing machine. The input is placed on the input tape with beginning and end markers. In the above figure the input is bounded by < and >.

A linear bounded automata can be formally defined as :

LBA is 7 - tuple on deterministic Turing machine with

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$ having

1. Two extra symbols of left end marker and right end marker which are not elements of Γ.
2. The input lies between these end markers.
3. The TM cannot replace < or > with anything else nor move the tape head left of < or right of >.

DIFFERENCE S BETWEEN TM AND PDA:

Push Down Automa :

1. A PDA is a nondeterministic finite automaton coupled with a stack that can be used to store a string of arbitrary length.
2. The stack can be read and modified only at its top.
3. A PDA chooses its next move based on its current state, the next input symbol and the symbol at the top of the stack.
4. There are two ways in which the PDA may be allowed to signal acceptance. One is by entering an accepting state, the other by emptying its stack.
5. ID consisting of the state, remaining input and stack contents to describe the "current condition" of a PDA.
6. The languages accepted by PDA's either by final state or by empty stack, are exactly the context - free languages.
7. A PDA languages lie strictly between regular languages and CSL's.

Turing Machines :

1. The TM is an abstract computing machine with the power of both real computers and of other mathematical definitions of what can be computed.
2. TM consists of a finite - state control and an infinite tape divided into cells.
3. TM makes moves based on its current state and the tape symbol at the cell scanned by the tape head.
4. The blank is one of tape symbols but not input symbol.
5. TM accepts its input if it ever enters an accepting state.
6. The languages accepted by TM's are called Recursively Enumerable (RE) languages.
7. Instantaneous description of TM describes current configuration of a TM by finite - length string.
8. Storage in the finite control helps to design a TM for a particular language.
9. A TM can simulate the storage and control of a real computer by using one tape to store all the locations and their contents.