=> Variants of Turing Machines:-

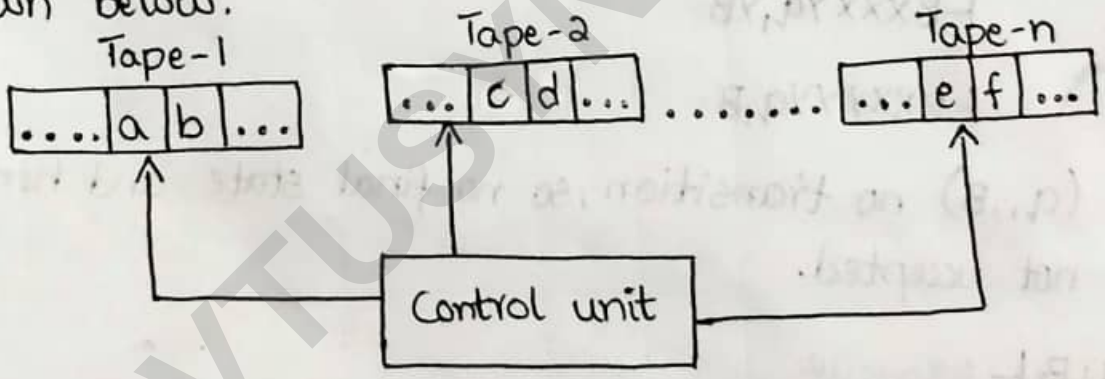
The variants of turing machines (TM) are:

i) Multi-tape Turing machine

ii) Non-deterministic Turing machine

1. Multi-tape Turing Machine:-

A multi-tape turing machine is nothing but a standard Turing Machine with more number of tapes. Each tape is controlled independently with independend read-write head. The pictorial representation of multi-tape Turing machine with n tapes is shown below.



The various components of multi-tape Turing machine are:

i) Finite control

ii) Multiple read/write (R/W) heads where each tape having its own symbols and read/write head.

Observe the following points from above Turing Machine:-

→ Each tape is divided into cells which can hold any symbol from the given alphabet. To start with the Turing Machine (TM)

should be in start state q_0

→ If the read/write head is pointing to tape-1 moves towards right, the read/write head pointing to tape-2 and tape-3 may move towards right or left depending on the transitions.

→ The move of the multi-tape Turing Machine deperds on the current state and the scanned symbol by each of the tape heads. For example, if number of tapes in Turing machine is 3 as shown in the figure and if there is a transition.

$$\delta(q, a, b, c) = (p, x, y, z, L, R, S)$$

where q is the current state. The transition can be interpreted as follows.

→ The Turing Machine in state q, will be moved to state p only when the first read/write head points to a, the second read-write head points to b and the third read/write head points to c.

→ the read/write head will be moved to left in the first case and right in the second case. But, the read/write head with respect to third tape will not be altered.

→ At the same time, the symbols a, b and c will be replaced by x, y and z.

⇒ Definition:-

The Multi-tape Turing Machine, is an n-tape machine

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

where Q is set of finite states

Σ is set of input alphabets

r is set of tape symbols

δ is transition function from $Q \times r^n$ to $Q \times r^n \times \{L, R\}^n$

q_0 is the start state

B is a special symbol indicating blank character

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

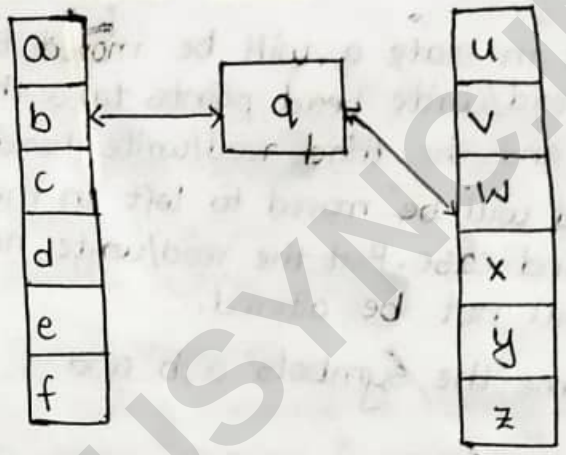
It can be easily shown that the n-tape Turing Machine in fact is equivalent to the single tape Standard Turing Machine.

**

Theorem:- Every language accepted by a multi-tape Turing Machine is accepted by standard Turing machine with single tape.

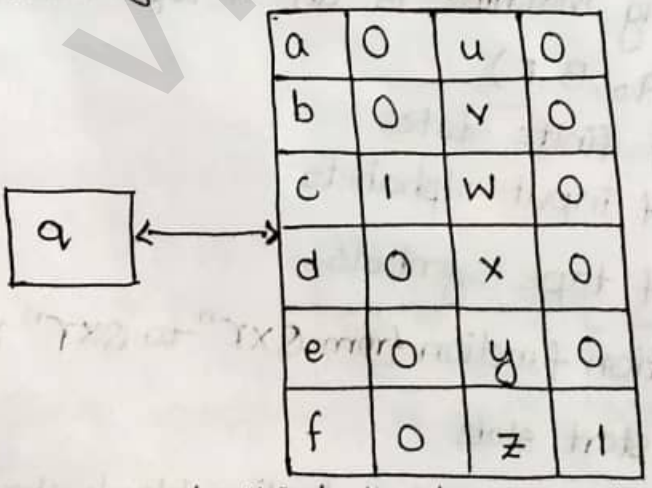
Note:- The theorem clearly indicates that every language accepted by a multi-tape Turing Machine is also accepted by a standard Turing Machine.

Proof:- The proof is by constructing a new machine. This can be shown by simulation. For example, consider a Turing machine with two tapes



Multi-tape

The above 2-tape turing machine can be simulated using single tape turing machine which has four tracks



Single tape Multiple tracks

-> The first and third tracks consist of symbols from first and second tape respectively. The second and fourth track consists of the positions of the read/write head with respect to first and second tape respectively.

→ The value 1 indicates the position of the read/write head. It is clear from the above figure that, when the machine is in state q_1, first read/write head points to the symbol c, the second read/write head points to the symbol z.

→ The machine can enter from one state to other, if and only if this transition is defined for Turing Machine with multi-tapes.

→ So, whatever transitions have been applied for multi-tape Turing Machine, if we apply the same transitions for the new machine that we have constructed, then the two machines are equivalent.

2. Non-deterministic Turing Machine:-

The difference between non-deterministic Turing Machine and deterministic Turing Machine lies only in the definition of δ.

Definition:- The non-deterministic 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 which is a subset of $Q\times\Gamma\times\{L,R\}$
q_0 is the start state
B is a special symbol indicating blank character
$F\subseteq Q$ is set of final states.

It is clear from the definition of δ that for each state q and tape symbol X, $\delta(q,X)$ is a set of triples:

$$\{(q_1,X_1,D),(q_2,X_2,D),(q_3,X_3,D)\ldots\ldots(q_i,X_i,D)\}$$

where i is a finite integer

→ D is the direction with 'L' indicating left or 'R' indicating right.

→ The machine can choose any of the triples as the next move.

The language accepted by Turing machine is defined as follows.

Definition:- Let $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$ be a non-deterministic Turing Machine. The language $L(M)$ accepted by M is defined as $L(M)=\{w \mid q_0w\vdash^*\alpha_1p\alpha_2$ where $w\in\Sigma^*$, $p\in F$ and $\alpha_1,\alpha_2\in\Gamma^*\}$

The language is thus a set of all those words W in Σ^* which causes M to move from start state q_0 to the final state p.

A non-deterministic Turing machine may have many moves that does not lead to a final state. But, we are interested in only those moves that leads to the final states. A non-deterministic Turing Machine in fact is no more powerful than the deterministic Turing Machine. Any language accepted by non-deterministic Turing Machine can be accepted by deterministic and both are equivalent. We can simulate a deterministic Turing Machine from a non-deterministic Turing Machine.

⇒ Linear Bounded Automata (LBA):-

Definition:-

The Linear Bounded Automation is a Turing Machine

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

where

→ Q is a set of finite states

→ Σ is set of input alphabets which also has two special symbols '[' and ']'

→ Γ is set of tape symbols

→ δ is subset of $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ with two more transitions of the form:

i) $\delta(q_i, [) = (q_j, [, R)$

ii) $\delta(q_i,]) = (q_j,], L)$

forcing the read/write head to be within the boundaries '[' and ']'

→ q_0 is the start state

→ B is a special symbol indicating blank character

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

Observe the following points:

i) Σ contains two special symbols '[' and ']'.

ii) The symbol '[' is called the left-end marker which is entered in the leftmost cell of the input tape preventing read/write head from getting off the left-end of the tape.

iii) The symbol ']' is called the right-end marker which is entered in the rightmost cell of the input tape preventing the read/write head from getting off the right-end of the tape.

iv) Both the end-markers should not appear on any other cell within the input tape.

v) Read/write head should not print any other symbol over both end-markers

It is clear from the definition that the read/write head cannot go out of the boundaries specified as '[' and ']'. Now, the string can be accepted by LBA only if there is a sequence of moves such that $q_0[w] \vdash^* [xq_fz]$ for some $q_f \in F$ and $x, z \in \Gamma^*$

⇒ Techniques for Turing Machine construction:-

The various techniques used for constructing a Turing Machine are as follows

a) Turing Machine with stationary head

b) Storage in the state

c) Multiple track Turing Machine

d) Subroutines.

i) Turing Machine with stationary head:-

In standard Turing Machine δ is defined as:

$$\delta(q,a) = (p,y,D)$$

where D stands for direction so, D can be left or right denoted by L or R. After consuming the input symbol a, the read/write head moves either towards left or towards right. An option such that the read/write head remains in the same cell for some input symbols can be implemented. This can be implemented using the transition:

$$\delta(q,a) = (p,y,S)$$

This means that, the Turing Machine in state q, after consuming the input symbol a, replace the input symbol a by y and changes its state from q to p without updating read/write head either towards left or towards right. The formal definition of Turing Machine with stay-option is

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

Q is set of finite states, Σ is set of input alphabets, Γ is set of tape symbols, δ is transition function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$ indicating the Turing Machine may move towards left or right or stay in same position after updating the symbol on the tape, q_0 is the start state, B is special symbol indicating blank character, $F \subseteq Q$ is set of final states.

2) Storage in the state:-

In all our machines such as: Finite Automata or Push Down Automata (PDA) or Turing Machine, we used states to remember things. Apart from this, we can use a state to store a symbol as well. So, in Turing Machine where we store in a state, the state becomes a pair (q, a) where q is the state and a is the tape symbol stored in the state (q, a). So, the new set of states is given by:

$$Q \times \Gamma$$

3) Multiple Track Turing Machines:-

In a Standard Turing Machine only one tape was used to store data. In multiple track Turing Machine, a single tape is assumed to be divided into a number of tracks. If a single tape is divided into k tracks, then the tape alphabets consists of k-tuples of tape symbols. So, only difference between standard Turing machine and Turing Machine with multiple tracks is the set of tape symbols. In case of standard Turing Machine, tape symbols are denoted by symbol Γ whereas in case of Turing Machine with multiple tracks, the tape symbols are denoted by symbol Γ^k. The working remains same as that of Standard Turing Machine.

4) Subroutines:-

We know that subroutines are used in programming languages whenever a task has to be done repeatedly. The same facility can be used in Turing Machine and complicated Turing machine's can be built using subroutines. A turing machine subroutine is a set of states that performs some pre-defined task. The turing machine subroutine has a start state and a state without any moves. This state which has no moves serves as the return state and passes the control to the state which calls the subroutine.

Example:- Design a Turing Machine to multiply two unary numbers separated by the delimiter 1.

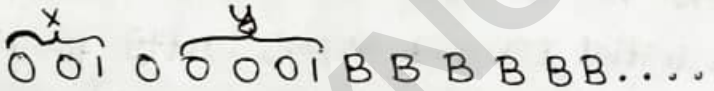
Solution:- Let us assume we have two unary numbers x and y such that x has m number of 0's and y has n number of 0's separated by the delimiter 1 as shown below

$0^m 10^n$ // m number of 0's and n number of 0's are separated by 1.

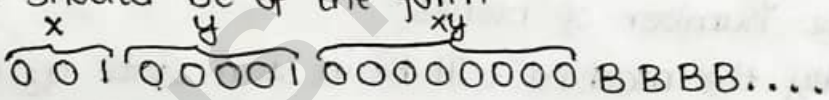
The product of x and y should be stored on the tape and the original numbers should not be destroyed. This can be visualized as shown below:

$0^m 10^n 10^{mn}$

To start with let us assume x = 00 and y = 0000 and are separated by delimiter 1. Assume y ends with 1 which can act as the delimiter for the input string which in turn is followed by infinite number of blanks (B's) as shown below:



The output should be of the form



⇒ Decidability:-

Definition of an algorithm

-> An algorithm is defined as step by step procedure to solve a given problem. The procedure is a finite sequence of instructions that have to be executed to complete a task.

-> Algorithm is terminated after finite number of steps for any input.

Decidability:-

-> Decidability is the capability of being decided, i.e, investigating the power of an algorithm i determining the validity of a given input

-> When a Turing machine reaches a final state, it halts.

-> There are Turing machines that never halt on some input.

-> Making such distinction between the languages accepted by a Turing Machine that halts on all input strings and a turing

machine that never halts on some input strings is decidability.

→ Decidable problem is the one for which we can design an algorithm (solvable problem).

→ Undecidable problem is the one for which we can not design an algorithm (unsolvable problem).

Recursively Enumerable Languages:-

→ A language $L \subseteq \Sigma^*$ is recursively enumerable iff there exists a Turing Machine 'M' such that $L = T(M)$ where $T(M)$ is the language accepted by Turing machine.

→ Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a turing machine. The language $L(M)$ accepted by M is defined as

$$L(M) = \{W \mid q_0W \vdash^* \alpha_1 p \alpha_2, \text{ where } W \in \Sigma^*, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

where q_0W is initial ID and $\alpha_1 p \alpha_2$ is the final ID

→ The string W — is the string to be scanned, should end with infinite number of blanks.

→ Initially, the machine will be in start state q_0, with read/write head pointing to first symbol of W from left.

→ After some sequence of moves, if the turing machine enters into final state and halts, then we say string W is accepted by turing machines.

→ The language accepted by turing machine is called Recursively enumerable language.

⇒ Decidable language:-

→ A language L is decidable (recursive) if L is the set of strings accepted by some Turing Machine that halts on every input.
i.e, if $W \in L$, then turing machine halts in final state (accept)
if $W \notin L$, then turing machine halts in non-final state (reject)

→ Decidable language implies to the existance of an effective algorithm which has the ability to determine whether an input is acceptable or not

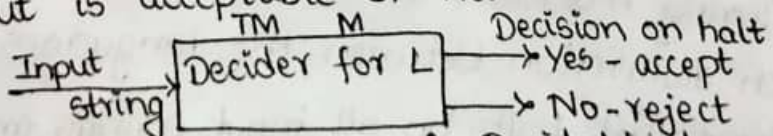


Fig. Turing Machine for Decidable language

⇒ Undecidable languages:-

→ A language L is undecidable language, if there is no turing machine which accepts the language and makes a decision for every input string W.

→ Undecidable languages imply the non-existance (doesn't exist) of an effective algorithm which has the ability to determine whether an input string is acceptable or not

Example:- i) The halting problem of turing machine
ii) The mortal matrix problem
iii) The post correspondance problem.

→ Sometimes undecidable languages may be partially decidable (Recursively Enumerable languages) but they will never be decidable

⇒ Halting problems in turing machine:-

→ The problem:- Given a turing machine, will it halt when run on some particular given input string W?

In general, we do not always know. The best we can do is, run the program and see whether it halts.

→ The output of the turing machine can be:

i) Halt:- The machine will halt (accept/reject) after a finite number of states.

ii) No halt:- The machine will never reach halt state no matter how long it runs.

Halting problem is undecidable and therefore, cannot be solved.

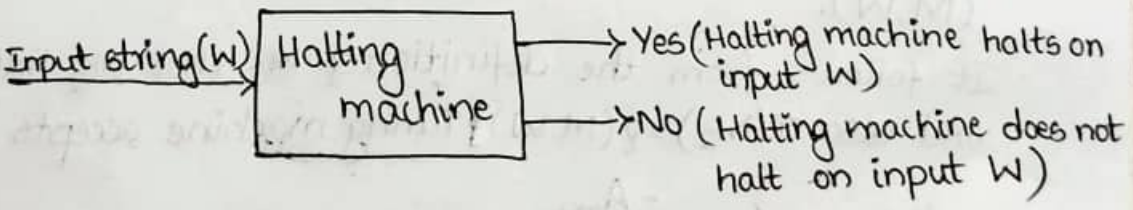


Fig. Block diagram of halting machine.

→ A reduction technique is used to prove the undecidability of halting problem of a Turing machine. Using this technique, a problem A is reducible to problem B, if a solution to problem B can be used to solve the problem A. Then,

i) if A is reducible to B and B is decidable, then A is decidable
ii) if A is reducible to B and B is undecidable, then A is undecidable

Theorem:- Prove that $HALT_{TM} = \{(M,W)\}$: The turing machine 'M' halts on input W is undecidable.

Proof:- We can assume that $HALT_{TM}$ is decidable, and get a contradiction.

→ Let M_1 be the turing machine such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M,W). We construct turing machine M_2 as follows:

1) For M_2, (M,W) is an input
2) The turing machine M_1 acts on (M,W)
3) If M_1 rejects (M,W), then M_2 rejects (M,W)
4) If M_1 accepts (M,W), simulate the turing machine M on the input string W until M halts.
5) If M has accepted W, M_2 accepts (M,W); otherwise M_2 rejects (M,W).

Observe the following points to prove:

→ When M_1 accepts (M,W) (step-4), the turing machine M halts on 'W'.
→ In the first case (first alternative of step-5) M_2 accepts (M,W)
→ In the second case (the second alternative of step-5) M_2 rejects (M,W).

It follows from the definition of M_2, that M_2 halts eventually and also $T(M_2) = \{(M,W)$ Turing machine accepts $W\}$

$= A_{TM}$

This is contradiction since A_{TM} is undecidable.

⇒ The Post Correspondence Problem (PCP):-

→ PCP is a popular undecidable problem. The PCP over an alphabet Σ is stated as follows.

→ Given the following two lists $X \times Y$ of non-empty strings over Σ

$$X = (x_1, - - - x_n)$$

$$Y = (y_1, - - - y_n)$$

→ We can say that there is a post correspondance solution, if for some $i_1, i_2 - - - - i_k$ where $1 \leq i_j \leq n$, such that the condition

$$x_{i1} - - - - x_{im} = y_{i1} - - - - y_{im} \text{ satisfies.}$$

[In simple words, lets assume, we have two lists both containing N words, aim is to find the concatenation of these words in some sequence such that both lists yield same result].

Example:-

Consider two lists X and Y

X = (aa, bb, abb) and Y = (aab, ba, b). Does the PCP with two lists have a solution?

Solution:- For sequence 1, 2, 1, 3, first list will yield aabbaaabb. Second list will yield aabbaaabb. So the solution to this PCP becomes 1, 2, 1, 3.

⇒ Applications of Turing Machines:-

1) Turing Machine is used to solve Recursively Enumerable problem

2) Used for knowing complexity theory

3) Used for neural network implementation.

4) Used for algorithmic, information theory and complexity studies, software testing, high performance computing, software engineering.

5) Used for recognizing patterns using regular expression.

6) For designing sequential circuit.