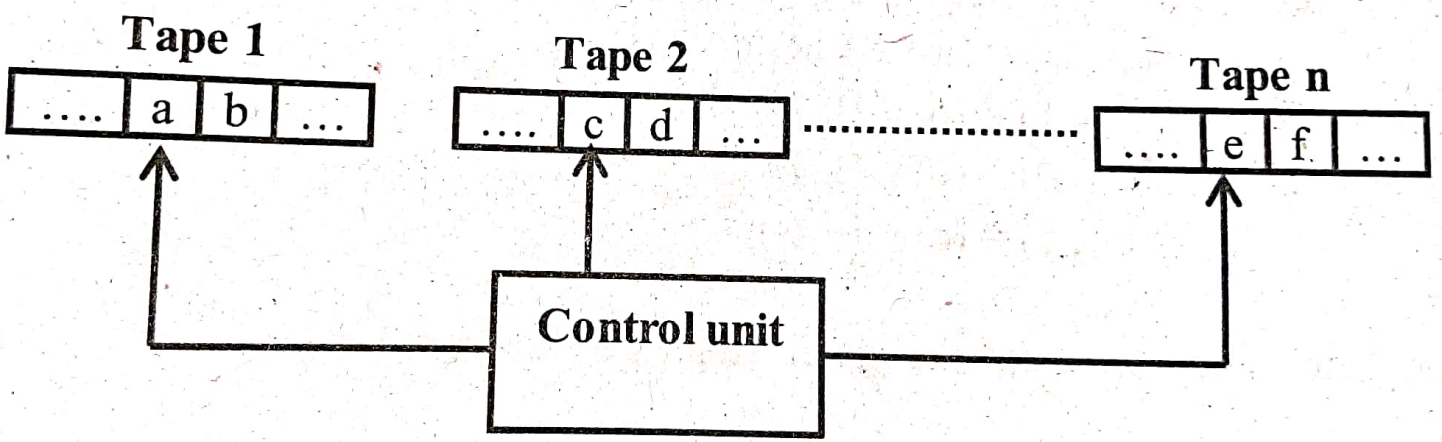
In this section, let us see "What are variants of Turing machines?" The variants of TM are:

- Multi-tape Turing machine
- Non-deterministic Turing machine

10.7.1 Multi-tape Turing Machines

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



The various components of multi-tape Turing Machine are:

- Finite control
- Multiple R/W heads where each tape having its own symbols and read/write head.

Observe the following points from above TM.

- Each tape is divided into cells which can hold any symbol from the given alphabet. To start with the TM should be in start state q_0.
- If the R/W head is pointing to tape 1 moves towards right, the R/W head pointing to tape 2 and tape 3 may move towards right or left depending on the transition.
- The move of the multi-tape TM depends on the current state and the scanned symbol by each of the tape heads. For example, if number of tapes in TM 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 TM 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 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.

The formal definition of Multi-tape Turing machine can be defined as follows:

Definition: The Multi-tape Turing Machine is an n-tape machine

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

where

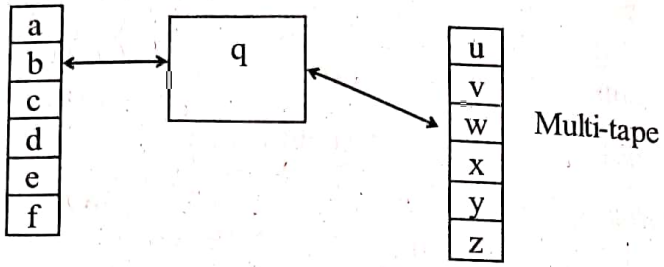
- Q is set of finite states
- Σ is set of input alphabets
- Γ is set of tape symbols
- δ is transition function from $Q \times \Gamma^n$ to $Q \times \Gamma^n \times \{L,R\}^n$
- q_0 is the start state
- B is a special symbol indicating blank character
- $F \subseteq Q$ is set of final states

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

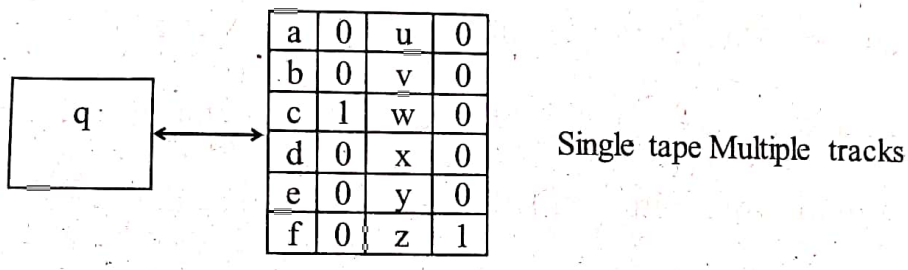
Theorem 10.1: Every language accepted by a multi-tape TM is accepted by standard Turing machine with single tape.

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

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



The above 2-tape TM can be simulated using single tape TM which has four tracks as shown in figure below:



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

10.7.2 Non-deterministic Turing Machines

The difference between nondeterministic TM and deterministic TM lies only in the definition of δ. The formal definition of *nondeterministic* TM is shown below:

Definition: The *nondeterministic* 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 TM is defined as follows.

Definition: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a nondeterministic TM. The language L(M) accepted by M is defined as

$$L(M)= \{w \mid q_0 w \vdash^* \alpha_1 p \alpha_2 \text{ where } w \in \Sigma^*, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

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 nondeterministic TM 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 nondeterministic TM in fact is no more powerful than the deterministic TM. Any language accepted by nondeterministic TM can be accepted by deterministic and both are equivalent. We can simulate a deterministic TM from a nondeterministic TM.

10.8 Linear bounded Automata (LBA)

In the previous sections, we have seen that the power of Turing Machine cannot be extended beyond the power of Standard Turing Machine by complicating the TM with multiple tapes or by using multi-dimensional tapes. Instead, the power of TM can be restricted by restricting the tape usage. An example of this is the Push Down Automaton which can be considered as a non-deterministic Turing machine. In PDA, it can be

assumed that the tape is used like a stack. We can also assume finite portion of the tape as input that leads to finite automaton. With slight alteration in usage of the tape, let us restrict the workspace on the tape using two delimiters '[' and ']'. The given string has to be enclosed between these two limiters. So, longer the string, longer the workspace. This leads to another class of machine called Linear Bounded Automaton (LBA). So, linear bounded automaton is a TM which is bounded based on the length of the input string. Thus, using TM and by restricting the usage of tape, we can obtain LBA, we construct PDA and finite automaton. It is clear from all these types of machines that TM is superset of all these machines. Now, let us "Define Linear bounded automaton"

Definition: The Linear Bounded Automaton is a TM

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

where

- Q is 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:
  - $\delta(q_i,[) = (q_j,[,R)$
  - $\delta(q_i,]) = (q_j,],L)$ f

  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:

- Σ contains two special symbols '[' and ']'.
- The symbol '[' is called the left-end marker which is entered in the leftmost cell of the input tape preventing the R/W head from getting off the left-end of the tape.
- The symbol ']' is called the right-end marker which is entered in the rightmost cell of the input tape preventing the R/W head from getting off the right-end of the tape.
- Both the end-markers should not appear on any other cell within the input tape
- R/W 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 $x, z \in \Gamma^*$.

Chapter 11
Decidability & Complexity

What are we studying in this chapter?

- The definition of an algorithm
- Decidability and decidable languages
- Undecidable languages
- Halting problem of Turing machine
- The Post's Correspondence problem
- Complexity **- 6 hours**

11.1 The Definition of an algorithm

In this section, let us see "What is an algorithm? Is the algorithm can be executed by Turing machine?"

Definition: An algorithm is defined as step by step procedure to solve a given problem. The procedure is finite sequence of instructions that have to be executed to complete a task. The algorithm is terminated after finite number of steps for any input.

The Church-Turing thesis states that any algorithm that can be carried out (executed) by a human or a computer, can also be carried out by a Turing machine. Thus, the Turing machine is considered as an ideal theoretical model for an algorithm. The Turing machine was considered as a tool to attach Hilbert's tenth problem. Hilbert's tenth problem is a challenge to provide a general algorithm which, for any given polynomial equation with integer coefficients and a finite number of unknowns can decide whether the equation has a solution with all unknowns taking integer values.

11.2 Decidability

As an algorithm terminates eventually, the Turing machine also terminates. Observe that Turing machine halts in following two situations:

- When a Turing machine reaches a final state, the it halts
- When a Turing machine reaches a state q when the scanned input symbol is a and if the transition $\delta(q, a)$ is not defined, it halts

But, there are some Turing machines that never halt on some inputs in any of the above situations. So, we have to make a distinction between the languages that are accepted by Turing machine and halts on all inputs and a Turing machine that never halts on some input strings.

Now, let us see "What is recursively enumerable language? What is recursive language"

Definition: A language $L \subseteq \Sigma^*$ is recursively enumerable if and only if there exists a Turing machine M such that $L = T(M)$ where $T(M)$ is the language accepted by Turing machine.

Definition: A language $L \subseteq \Sigma^*$ is recursive if and only if there exists a Turing machine that satisfies the following two conditions:

- If $w \in L$ then w is accepted by Turing machine on reaching the final state and the machine halts
- If $w \notin L$, then w is rejected by Turing machine without reaching the final state and the machine halts

A machine may accept a language or it may reject a language. So, the output of the machine may be *accept* or *reject*. This problem is called *membership problem*. Formally, the membership problem can be stated as "Given a machine M and a string x, does M accept x?" The output will be *yes/no*. Given a language, the machine may have to identify whether the language is *finite* or *infinite*. All such problems with two answers *yes/no, accept/reject, finite/infinite* are called ***decision problems***. For majority of the decision problems, we can design decision algorithms.

Now, let us "Define decidable language. Define undecidable language"

Definition: A decision problem or decision language with yes/no answers is decidable if and only if the corresponding language is recursive. In other words, a *decidable language* is a recursive language with yes/no answers. Otherwise, it is *undecidable language*.

11.3 Decidable Languages

In this section, let us consider the decidability of regular and context free languages. Now, let us *"Prove that DFA is decidable language"*

Theorem: DFA is decidable

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We have to construct a Turing machine M' that always halts and accepts L(M). We know that DFA always ends in some state after

reading the string w. Now, we shall construct a Turing machine M' that simulates DFA. Let us define Turing machine as shown below:

1. If w is the string to be scanned by Deterministic Finite Automaton M, then, (M, w) is input to Turing machine M'.
2. Simulate M and input w in Turing machine M'. Here, Turing machine M' checks whether input (M, w) is valid input. If (M, w) is invalid, the Turing machine M' rejects (M, w) and halts. If (M, w) is valid input, M' writes the initial state q_0 and leftmost input symbol of w. It updates the state using δ and reads the next symbol in w.
3. If simulation ends in an accepting state of M, then Turing machine M' accepts w. If the simulation ends in nonaccepting state of M, then M' rejects w.

This, it is evident that Turing machine M' accepts (M, w) if and only if w is accepted by DFA M.

Note: If a language L is not accepted by a Turing machine, then the language is not recursively enumerable. One important problem which is not recursively enumerable that is unsolvable/undecidable decision problem is "**Halting problem**".

11.4 Halting Problem

In this section, let us see *"What is halting problem?"* The "***Halting Problem***" can informally be stated as shown below:

Halting problem statement: Given a Turing machine M and an input string w with the initial configuration q_0, after some (or all) computations do the machine M halts? In other words, if M is a Turing machine, we have to identify whether (M, w) halts or does not halt when w is applied as the input. Given the description of an arbitrary Turing machine M and the input string w, we are looking for a single Turing machine that will predict whether or not the computation of M applied to w will halt.

Note: Alan Turing in 1990 proved that the *halting problem is undecidable*, and therefore, *cannot* be solved.

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 the problem B can be used to solve the problem A. Thus,

- If A is reducible to B and B is decidable, then, A is decidable
- If A is reducible to B and B is undecidable, then A is undecidable

Now, let us "*Prove that halting problem of Turing machine is undecidable*". The following theorem proves it.

Theorem: L_{HLT} = { (M, w) : The Turing machine M halts on input w is undecidable}

Proof: Let L_{HLT} is decidable and get a contradiction. Let M_1 is a Turing machine such that $L(M_1) = L_{HLT}$ and let M_1 halt eventually halts on all (M, w). Let us construct Turing machine M_2 as follows:

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

Observe the following points to prove:

- When M_1 accepts (M, w) (see step 4), the Turing machine M halts on w.
- In the first case (first alternative in step 5) M_2 accepts (M, w)
- In the second case (second alternative in step 5) M_2 rejects (M, w).

So, it follows from definition of M_2 that the Turing machine M_2 halts eventually and hence, $L(M_2)$ = {(M, w): The Turing machine accepts w} which is undecidable. This is a contradiction. So, the Turing machine M halts on input w is undecidable.

11.5 Post Correspondence Problem

Now, let us see "What is Post correspondence problem?"

Definition: The Post correspondence problem can be stated as follows. Given two sequences of n strings on some alphabet Σ say:

$$A = w_1, w_2, \ldots, w_n \quad \text{and} \quad B = v_1, v_2, \ldots, v_n$$

we say that there exists a Post correspondence solution for pair (A, B) if there is a nonempty sequence of integers i,j,.....,k, such that

$$w_i w_j \ldots w_k = v_i v_j, \ldots . v_k.$$

The Post correspondence problem is to devise an algorithm that will inform us, for any (A, B) whether or not there exists a PC-solution.

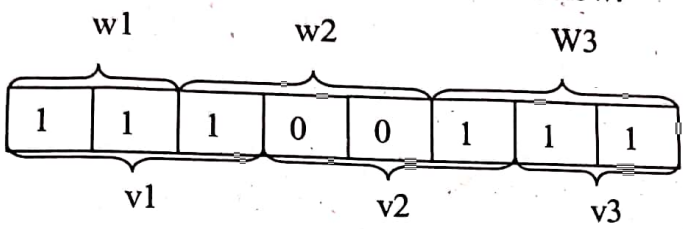
For example, Let Σ={0,1}. Let A is w1, w2, w3 as shown below:

$w_1 = 11, w_2 = 100, w_3 = 111$

Let B is v1, v2, v3 as shown below:

$v_1 = 111, v_2 = 001, v_3 = 11$

For this case, there exists a PC-solution as shown below:



If we take

$w_1 = 00, w_2 = 001, w_3 = 1000$

$v_1 = 0, v_2 = 11, v_3 = 011$

there cannot be any PC-solution simply because any string composed of elements of A will be longer than the corresponding string from B.

11.6 Complexity

When a problem or language is decidable, it means that the problem is computationally solvable in principle. But, it may require enormous amount of computation time and enormous amount of memory to solve it completely. In complexity theory, many scientists believe that P ≠ NP. But, this is still open and any one can challenge this relation. Observe the following points:

- P stands polynomial time: This class of problems can be solved by a deterministic algorithm in a polynomial time
- NP stands for non-deterministic polynomial time: This class of problems can be solved by a non-deterministic algorithm in a polynomial time.

11.6.1 Growth Rate of Functions

Given two algorithms to solve the same problem, it is necessary to compare those algorithms to find out which is more efficient. We normally use big-oh notation which is defined as shown below:

Definition: Let f(b) be the time efficiency of an algorithm. The function f(n) is said to be O(g(n)) denoted by

$$f(n) = O(g(n))$$

if and only if there exists a positive constant c and positive integer n_0 satisfying the constraint

$$f(n) \leq c*g(n) \text{ for all } n \geq n_0.$$

Theorem: If $p(n) = a_k n^k + a_{k-1}n^{k-1} + \ldots\ldots a_1 n^1 + a_0$ with $a_k > 0$, then $p(n) = O(n^k)$.

Solution: It is given that $f(n) = a_k n^k + a_{k-1}n^{k-1} + \ldots\ldots a_1 n^1 + a_0$

Each term in the summation is of the form $a_i n^i$. Since, n is non-negative, a particular term will be negative only if $a_i < 0$. So,

$$|f(n)| = |a_k n^k + a_{k-1}n^{k-1} + \ldots\ldots a_1 n^1 + a_0|$$

$$\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \ldots\ldots |a_1| n^1 + |a_0|$$

$$= n^k \left[|a_k| + \frac{|a_{k-1}|}{n} + \ldots\ldots \frac{|a_1|}{n^{k-1}} + \frac{|a_o|}{n^k} \right]$$

$$\leq n^k \left(|a_k| + |a_{k-1}| + \ldots\ldots |a_1| + |a_o| \right)$$

$$\leq c*n^k \text{ where } c = \left(|a_k| + |a_{k-1}| + \ldots\ldots |a_1| + |a_o| \right)$$

i.e., $f(n) \leq c*n^k$ where $c = \left(|a_k| + |a_{k-1}| + \ldots\ldots |a_1| + |a_o| \right)$ for $n > 1$

$$\Downarrow$$

$$f(n) \leq c*g(n) \text{ for } n > n_o \text{ where } g(n) = n^k \text{ and } n_0 = 1.$$

So, by definition, we can write

$$f(n) = O(g(n)) \text{ or } f(n) \in O(g(n))$$

i.e., **$f(n) = O(n^k)$ or $f(n) \in O(n^k)$**

11.6.2 Classes of P and NP

In this section, let us discuss P and NP problems with respect to language accepted by Turing machines.

Now, let us see "What is the time complexity of a Turing machine which accepts a string w of length n?"

Definition: Given an input string w of length n, the time complexity of a Turing machine is given by T(n). It indicates that the Turing machine halts after making maximum of T(n) moves. In this case, Turing machine halts and it is called Deterministic Turing machine.

Now, let us "Define class P with respect to Turing machine?"

Definition: Let L(M) is the language accepted by Turing machine with time complexity T(n). The *language L is in class P* if there exists some polynomial p(n) such that L = L(M) for some deterministic Turing machine M with time complexity T(n).

Example 11.1: Obtain the time complexity of a Turing machine which accepts $L = \{a^n b^n \mid n \geq 1\}$

Step 1: When we construct the Turing machine for the above language every leftmost *a* is replaced by X and every leftmost *b* is replaced by Y. So, the total number of moves required = 2n

Step 2: In step 2 of constructing the machine, we see that the above procedure is repeated till all *n* number of *a's* are replaced by X's. So, the above procedure has to be repeated *n* times.

Step 3: So, total number of moves = $2n(n) = 2n^2 \approx n^2$ (By neglecting lower order terms and constants).

So, time complexity is given by $O(n^2)$

Now, let us "Define class NP with respect to Turing machine?"

Definition: Let L(M) is the language accepted by Turing machine with time complexity T(n). The *language L is in class NP* if there exists some polynomial p(n) such that L = L(M) for some non-deterministic Turing machine M with time complexity T(n).

11.7 Quantum computation

In this section, let us discuss about *quantum computers* and *church-Turing thesis*.

11.7.1 Quantum computers

In this section, let us see how data is represented and how to build simple GATES and define quantum computers.

Now, let us is "What is quantum computer? Explain"

Definition: The computers that are built based on the principles of quantum mechanics are called *quantum computers*. Quantum computers are different from binary digital electronic computers based on transistors. The data in digital computers is represented using binary states 0 and 1 whereas the data in quantum computers is represented mathematically as shown below:

$$|\Psi> = \alpha|0> + \beta|1>$$

The qubit can be explained as shown below:

- Two possible states are represented as $|0>$ and $|1>$
- Unlike classical bit 0 or 1, a qubit can have infinite number of states other than $|0>$ and $|1>$ and can be represented as $|\Psi> = \alpha|0> + \beta|1>$ where $|\alpha|^2 + |\beta|^2 = 1$.
- In qubits 0 and 1 are called *computational basis states*, $|\Psi>$ is called a *superposition* and $|\Psi> = \alpha|0> + \beta|1>$ is called *quantum state.*
- In digital computer, data can be represented as 0 or 1. But, it is not possible to determine the quantum state on observation. In quantum computers we get a state $|0>$ with probability $|\alpha|^2$ and state $|1>$ with probability $|\beta|^2$
- Multiple qubits can be defined in a similar manner. For example, a two-qubit system has four computational basis states, $|00>$, $|01>$, $|10>$, $|11>$ and quantum states are represented as:

$$|\Psi> = \alpha_{00}|00> + \alpha_{01}|01> + \alpha_{10}|10> + \alpha_{11}|11>$$

where

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$$

- Normal NOT gate interchanges 0 and 1 whereas in case of qubit NOT gate, $\alpha|0> + \beta|1>$ is changed to $\alpha|1> + \beta|0>$

Thus, a quantum computer can be defined as shown below: *A quantum computer is a system built from number of quantum circuits containing wires and elementary quantum gates such as OR, NOT, AND, NOR etc, to carry out manipulation of quantum information.*

11.7.2 Church Turing Hypothesis (Church's/Church-Turing thesis)

In this section, let us concentrate on Church Turing Hypotheses which is also called Church-Turing thesis.

Various formal models of computations such as *Recursive functions* and *Post systems* were established by three prominent persons A.Church, S.C.Kleene and E.Post.

A function is called *primitive recursive* if and only if it can be constructed from the basic functions by successive composition and primitive recursion. A *Post system* is similar to unrestricted grammar consisting of an alphabet and some production rules by which successive strings can be derived.

In addition to recursive functions and Post systems, many other formal computations models have been proposed. On examination it was found that though the computational models looked quite different, they expressed the same thing. This observation was formalized in ***Church's thesis*** which is stated as follows:

Any "effective computation" or "any algorithmic" procedure that can be carried out by a human being or a team of human beings or a computer, can be carried out by some Turing machine. In other words, there is an effective procedure to solve a decision

problem P if and only if there is a Turing machine that answers *yes* on inputs $w \in P$ and *no* for $w \notin P$.

This theory maintains that all the models of computations those are proposed and yet to be proposed, are equivalent in their power to recognize languages or compute functions. This thesis predicts that it is unable to construct models of computation more powerful than the existing ones.

The above statement is known as "Church's thesis" named after the logician A.Church. Since the *Church's thesis* is closely related to Turing's thesis which states that we cannot go beyond Turing machines or their equivalent, it is also called **Church-Turing thesis**.

Since there is no precise definition for "effective computation" or there is no precise definition for "algorithmic procedure", Church's thesis is not a mathematically precise statement. So, this statement is not proved at the same time it has been not been disproved. Even though it is simply stated and not proved, now majority of scientists have accumulated enough evidence over the years that has caused Church's thesis to be generally accepted.

Exercises:

1) What is an algorithm? Is the algorithm can be executed by Turing machine?"
2) What is recursively enumerable language? What is recursive language"
3) Define decidable language. Define undecidable language"
4) Prove that DFA is decidable language"
5) What is halting problem
6) Prove that halting problem of Turing machine is undecidable
7) What is Post correspondence problem?"
8) If $p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots\ldots a_1 n^1 + a_0$ with $a_k > 0$, then prove that $p(n) = O(n^k)$.
9) What is the time complexity of a Turing machine which accepts a string w of length n?
10) Define class P with respect to Turing machine?"
11) Obtain the time complexity of a Turing machine which accepts $L = \{a^n b^n \mid n \geq 1\}$
12) Define class NP with respect to Turing machine
13) What is quantum computer? Explain
14) Explain Church-Turing thesis.