input etc. of mainly the description of the process in some coded form of a sequence of steps required to solve the problem in the form of the move function δ. In case of universal turing machine, the **process part** involving δ of the turing machine M, and the inputs are expressed in the code (that is language) of the Universal turing machine. This code of the process alongwith the code of the input, is stored in the memory of the UTM. And just on the lines of the control unit of a general purpose computer the control unit of UTM, reads the codes for steps, one step at a time, decodes and execute the code for each step, until the code for the final result is stored on the tape of UTM.

Observation 2. A Turing machine Tm designed to solve a particular problem P can easily be specified by

(*i*) The initial state say $q_{0\text{Tm}}$ of the Turing machine M.

(*ii*) The next-move function δ_m of Tm, which can be described by the rules of the form : if the current state of Tm is q_i and contents of cell being scanned are a_j then the next state of Tm is q_k, the symbol to be written in the current cell is a_1 and move m_f of the Tape Head may be : To-left, To-right or None.

Thus, each of these rules for a particular Tm can be specified by **quintupoles** of the form (q_i, a_i, a_1, m_f). And hence the next-move function δ_{Tm} for machine Tm is completely specified by the set.

$$\{(q_i, a_j, q_k, a_1, m_f) : q_i, q_j \in Q_{\text{Tm}};\ a_j, a_1, \in \Gamma_{\text{Tm}};\ m_f \in \{\text{To-left, To-right, None}\}\}$$

Observation 3. Next question that arises in the context of the construction of universal Turing machine, is about the number of distinct states in UTM and number of distinct inputs/tape symbols required in the UTM, so that it can solve any solvable problem.

As UTM should be able to simulate each Turing Machine, therefore, it may appear that number of distinct states and number of distinct tape symbols in the UTM, should be at least as much as is possible in any Tm, because UTM may be required to accomplish the task of any Tm. However, by proper coding techniques we may use only two symbols to represent set of symbols.

9.12. THE HALTING PROBLEM

Let assume that we have given the description of turing machine Tm and an input w, when started in the initial configuration q_0w, perform a computation that eventually halts? Using an abbreviated way of talking about the problem, we ask wheather Tm applied to w, or simply (Tm, w) halts or does not halt. The domain of this problem is to be taken as the set of all turing machines and all w; that is, we are looking for a single turing machine that, given the description of an arbitrary Tm and w, will predict whether or not the computation of Tm applied to w will halt.

We can not find the answer by simulating the action of Tm on w, say by performing it on universal turing machine, because there is no limit on the length of the computation. If Tm enters an infinite loop, then no matter how long we wait, we can never be sure that Tm is in fact in a loop. It may be simple case of very long computation. What we need is an algorithm that can determine the correct answer

for any Tm and w by performing some analysis on the machine's description and the input. But it is clear that no such algorithm exists.

Definition

In short halting problem is :

To determine for an arbitrary given Turing machine Tm and input w,

Whether Tm will eventually halt on input w.

9.13. UNDECIDABILITY/DECIDABILITY

We know that recursive languages are those languages which are accepted by atleast one turing machine and these sets of recursive languages are subclass of regular sets, called the recursive sets.

Definition

"A problem whose language is recursive is said to be decidable".

Otherwise problem is undecidable. That is, a problem is undecidable if there exist no algorithm that takes as input an instance of the problem and determine whether the answer to that instance is "yes" or "no".

9.13.1. Facts about Turing-Decidable and Turing Acceptable Languages

1. If L is turing-decidable then L is turing-acceptable

Proof. If m decides L, then the turing machine

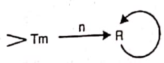

Fig. 9.36.

accepts/semidecides L.

That if Tm decides L, we enter into either state 'Y' or 'n'.

➤ If Tm halts in 'Y', this machine halts (because if $w \in L$, Tm goes to state Y and halts).

So clearly if L is decided by some turing machine Tm then L is also accepted by some turing machine.

2. If L is turing-decidable then so is $\bar{L}$.

Proof. Let Tm be a turing machine such as :

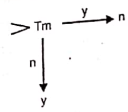

Fig. 9.37.

This machine goes to a no (n) state when Tm ended up in a yes (Y) state and vice-versa.

Therefore, there is also a turing machine which decides $\bar{L}$ (complement of the language L).

Theorem 9.8. If a language L and its complement $\bar{L}$ are both recursively enumerable, then L (and hence $\bar{L}$) is recursive.

Proof. Let Tm_1 and Tm_2 accept L and $\bar{L}$ respectively. Let us construct a turing machine Tm which simulate Tm_1 and Tm_2 simultaneously. Tm accepts w if Tm_1 accepts and rejects w if Tm_2 will accept. Thus Tm will always say either "Yes" or "No", but never say both. Note that there is not a priority limit on how long it may take before Tm_1 or Tm_2 accepts, but it is certain that one or the other will do so. Since Tm is algorithm that accepts L, it follows that L is recursive.

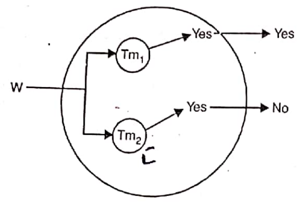

Fig. 9.41.

Theorem 9.9. If L is a recursive language than $\Sigma^* - L$ is recursive.

Proof. The required Turing machine Tm-complement can be represented by following diagram.

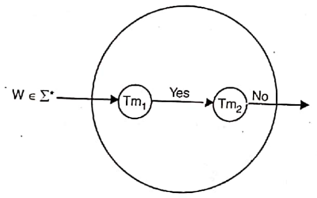

Fig. 9.42.

The machine Tm-complement functions as follows : when a string $W \in \Sigma^*$ is given an input to Tm-complement, its control passes the string to Tm_1 as input to Tm_1. As Tm_1 decides the language L, therefore, for $W \in L$ after a finite number of moves, Tm_1 outputs "Yes", which is the given as input to Tm_2, which in turn returns "No".

Similarly for $W \notin L$, Tm_2 returns "Yes". Hence there exist a turing machine Tm-complement for $\Sigma^* - L$. So it is turing-decidable, that is recursive.

9.13.6. The Post Correspondence Problem

The post correspondence problem is another undecidable problem that turns out to be a very helpful tool for proving problems in logic or in formal language theory to be undecidable.

Definition Let Σ be an alphabet with at least two letters. An instance of the post corresponding problem (for short PCP) is given by two sequences $U = (u_1, u_2, ..., u_m)$ and $V = (v_1, v_2, ..., v_m)$ of strings $u_i, v_i \in \Sigma^*$. The problem is to find whether there is a (finite) sequence

$(i_1, i_2, ..., i_p)$, with $i_j \in \{1, 2, ..., m\}$ for

$$i_j = 1, 2, ... p \text{ so that, } P \geq 1$$

$$u_{i_1}, u_{i_2} u_{i_3} ... u_{i_p} = v_{i_1} v_{i_2} ... v_{i_p}.$$

Equivalently, on instance of the PCP is a sequence of pairs

$$\begin{pmatrix} \mu_1 \\ \vdots \\ v_1 \end{pmatrix}, ..., \begin{pmatrix} \mu_m \\ \vdots \\ v_m \end{pmatrix}$$

The sequence $i_1, i_2 ... i_p$ is said to be solution to this instance of PCP.

Example 9.24. Let $\Sigma = \{0, 1\}$. Let X and Y be lists of three strings each, defined as follows :

	List X	List Y
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

In this case PCP has a solution, let $P = 4$

$$i_1 = 2, i_2 = 1, i_3 = 1 \text{ and } i_4 = 3 \text{ then}$$

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$$

$$= 101111110$$

which is the solution of instance of PCP.

Example 9.25. Prove that following instance of PCP has no solution over $\Sigma = \{0, 1\}$, X and Y be lists of three strings as follows :

	List X	List Y
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Solution. Let us assume that this instance of PCP has solution $i_1, i_2, ..., i_p$. Clearly $i_1 = 1$ since no string beginning with $w_2 = 011$ can equal a string beginning with $x_2 = 11$; no string beginning with $w_3 = 101$ can equal a string beginning with $x_3 = 011$.

We write the string from list X the corresponding string from Y. So for we have

10

101

The next selection from X must begin with a 1. Thus $i_2 = 1$ or $i_2 = 3$. But $i_2 = 1$ will not do, since no string beginning with $w_1 w_1 = 1010$ can equal a string beginning with $x_1 x_1 = 101101$. with $i_2 = 3$ we have

$$10101$$

$$101011$$

Since the string from list Y again exceeds the string from list X by the single symbol 1, a similar argument shows that $i_3 = i_4 = \ldots = 3$. Thus there is only one sequence of choices that generates compatible strings, and for this sequence string Y is always one character longer. Thus this instance of PCP has no solution.

Example 9.26. Find the solution of following instance of PCP

$$\begin{pmatrix} abab \\ ababaaa \end{pmatrix}, \begin{pmatrix} aaabbb \\ bb \end{pmatrix}, \begin{pmatrix} aab \\ baab \end{pmatrix}, \begin{pmatrix} ba \\ baa \end{pmatrix}, \begin{pmatrix} ab \\ ba \end{pmatrix}, \begin{pmatrix} aa \\ a \end{pmatrix}.$$

Solution. The solution for this instance is

$i_1 = 1, i_2 = 2, i_3 = 3, i_4 = 4, i_5 = 5, i_6 = 5, i_7 = 6$

$$ababaaabbbaabbaababaa = ababaaabbbaabbaababaa.$$