Now, we shall see "What are solvable/decidable problems and what are not solvable (unsolvable/undecidable) problems?"

8.2 A Language that is not Recursively Enumerable

Definition: The decision problems that have decision algorithms the output of which is *yes/no* are called solvable problems. According to Church-Turing thesis, an appropriate way to formulate precisely the idea of a decision algorithm is to use a Turing machine. The language L accepted by a Turing machine M is *recursively enumerable language* if and only if L = L(M). Any instance of a problem for which the Turing machine halts whether the input is accepted or rejected is called ***solvable*** or ***decidable problem***. There are so many problems that are solvable. But, there are some problems that are *not solvable*. Now, we shall see what are called *unsolvable/undecidable problems.*

8.3 Undecidable problems that are RE

Definition: The problems that run forever on a Turing machine are not *solvable*. In other words, there are some problem input instances for which Turing machines will not halt on inputs that they do not accept. Those problems are called ***unsolvable*** or ***undecidable problems.*** In general, if there is no general algorithm capable of solving every instance of the problem, then the decision problem is *unsolvable*. More precisely, if there is no Turing machine recognizing the language of all strings for various instances of the problems input for which the answer is *yes* or *no*, then the decision problem is *unsolvable.*

Let us assume M is a Turing machine with input alphabets {0, 1}, *w* is a string of 0's and 1's and M accepts *w*. If this problem with inputs restricted to binary alphabets {0, 1} is undecidable, then the general problem is undecidable and can not be solved with a Turing machine with any alphabet.

Note: The term *unsolvable* and *undecidable* are used interchangeably.

Note: Even though we are able to answer the question in many specific instances, a problem may be undecidable. It means that there is no single algorithm guaranteed to provide an answer for every case.

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**".

8.4 Halting Problem

The "***Halting Problem***" can informally be stated as "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 we have to identify whether (M, w) where M is the Turing machine, halts or does not halt when *w* is applied as the input. The domain of this problem is to be taken as the set of all Turing machines and all *w* i.e., 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.

When we state decidability or undecidability results, we must always know what the domain is, because this may affect the conclusion. The problems may be decidable on some domain but not on another.

It is not possible to find the answer for Halting problem by simulating the action of M on *w* by a universal Turing machine, because there is no limit on the length of the computation. If M enters into an infinite loop, then no matter how long we wait, we can never be sure that M is in fact in a loop. The machine may be in a loop because of a very long computation. What is required is an algorithm that can determine the correct answer for any M and *w* by performing some analysis on the machine's description and the input.

Formally, the Halting Problem is stated as "Given an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ and the input $w \in \Sigma^*$, does M halt on input *w*?"

8.5 Post's Correspondence Problem

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

$$A = w_1, w_2, \ldots, w_n$$

and

$$B = v_1, v_2, \ldots, v_n$$

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 tell us, for any (A,B) whether or not there exists a PC-solution.

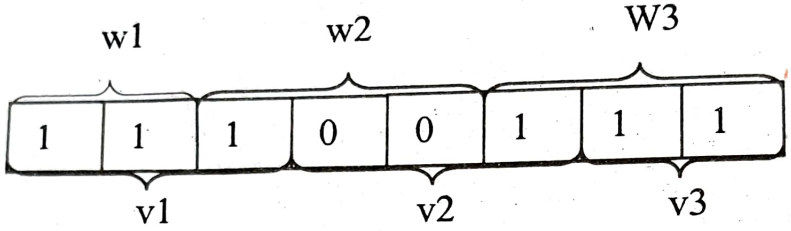
For example, Let $\Sigma = \{0.1\}$. Let A is w1, w2, w3 as shown below:

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

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 element A will be longer than the corresponding string from B.

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