Decidable and Undecidable problems in Theory of Computation

A problem is said to be **Decidable** if we can always construct a corresponding **algorithm** that can answer YES/NO for the problem correctly.

We can understand Decidable problems by considering a simple example. Suppose we are asked to compute all the prime numbers in the range of 1000 to 2000. To find the **solution** of this problem, we can easily devise an algorithm that can enumerate all the prime numbers in this range.

Now talking about Decidability in terms of a Turing machine, a problem is said to be a Decidable problem if there exists a corresponding Turing machine which **halts** on every input with an answer- **yes or no**. These problems are also termed as **Turing Decidable** since a there exist a Turing machine always halts on every input, accepting or rejecting it.

Semi- Decidable Problems –

Semi-Decidable problems are those for which a Turing machine halts on the input accepted by it but it can either halt or loop forever on the input which is rejected by the Turing Machine. Such problems are termed as **Turing Recognisable** problems.

Examples – We will now consider few important **Decidable problems**:

- ***Are two regular languages L and M equivalent?***

  We can easily check this by using Set Difference operation.

  L-M=Null and M-L =Null.

  Hence (L-M) U (M-L) = Null, then L,M are equivalent.

- ***Membership of a CFL?***

  We can always find whether a string exists in a given CFL by using an algorithm based on dynamic programming.

- ***Emptiness of a CFL***

  By checking the production rules of the CFL we can easily state whether the language generates any strings or not.

Undecidable Problems –

The problems for which we can't construct an algorithm that can answer the problem correctly in limited time (finite time) are termed as Undecidable Problems.

These problems may be partially decidable but they will never be decidable. That is there will always be a condition that will lead the Turing Machine into an infinite loop without providing an answer at all.

We can understand Undecidable Problems by considering ***Fermat's Theorem,*** a popular Undecidable Problem which states that no three positive integers a, b and c for any n>2 can ever satisfy the equation: a^n + b^n = c^n.

If we feed this problem to a Turing machine to find such a solution which gives a contradiction then a Turing Machine might run forever, to find the suitable values of n, a, b and c. But we are always unsure whether a contradiction exists or not and hence we term this problem as an **Undecidable Problem**.

Examples – These are few important **Undecidable Problems**:

- ***Whether a CFG generates all the strings or not?***

  As a CFG generates infinite strings, we can't ever reach up to the last string and hence it is Undecidable.
- ***Whether two CFG L and M equal?***

  Since we cannot determine all the strings of any CFG, we can predict that two CFG are equal or not.
- ***Ambiguity of CFG?***

  There exist no algorithm which can check whether for the ambiguity of a CFL. We can only check if any particular string of the CFL generates two different parse trees then the CFL is ambiguous.
- ***Is it possible to convert a given ambiguous CFG into corresponding non-ambiguous CFL?***

  It is also an Undecidable Problem as there doesn't exist any algorithm for the conversion of an ambiguous CFL to non-ambiguous CFL.
- ***Is a language Learning which is a CFL, regular?***

  This is an Undecidable Problem as we can not find from the production rules of the CFL whether it is regular or not.

Some more **Undecidable Problems** related to Turing machine:

Halting problem of Turing Machine is Undecidable

- ***Membership*** problem of a Turing Machine?
- ***Finiteness*** of a Turing Machine?
- ***Emptiness*** of a Turing Machine?

- ***Whether the language accepted by Turing Machine is regular or CFL?***