UNIT-5

Undecidability and Reducibility in TOC

Decidable Problems:

A problem is decidable if we can construct a Turing machine which will halt in finite amount of time for every input and give answer as ‘yes’ or ‘no’. A decidable problem has an algorithm to determine the answer for a given input.

Examples

- **Equivalence of two regular languages:** Given two regular languages, there is an algorithm and Turing machine to decide whether two regular languages are equal or not.
- **Finiteness of regular language:** Given a regular language, there is an algorithm and Turing machine to decide whether regular language is finite or not.
- **Emptiness of context free language:** Given a context free language, there is an algorithm whether CFL is empty or not.

Undecidable Problems:

A problem is undecidable if there is no Turing machine which will always halt in finite amount of time to give answer as ‘yes’ or ‘no’. An undecidable problem has no algorithm to determine the answer for a given input.

Examples

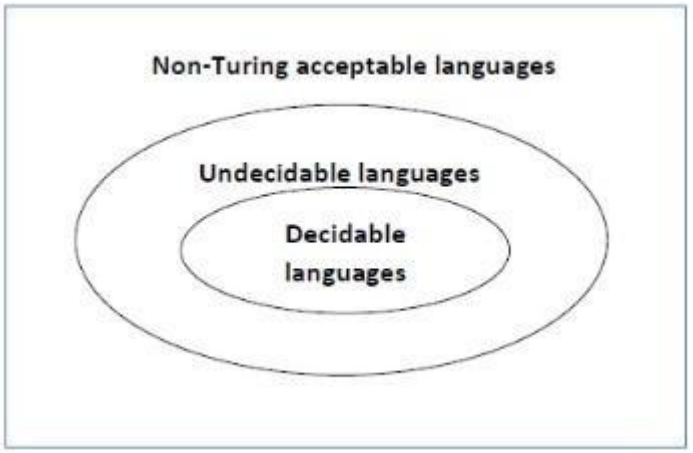
- **Ambiguity of context-free languages:** Given a context-free language, there is no Turing machine which will always halt in finite amount of time and give answer whether language is ambiguous or not.
- **Equivalence of two context-free languages:** Given two context-free languages, there is no Turing machine which will always halt in finite amount of time and give answer whether two context free languages are equal or not.
- **Everything or completeness of CFG:** Given a CFG and input alphabet, whether CFG will generate all possible strings of input alphabet (?*)is undecidable.
- **Regularity of CFL, CSL, REC and REC:** Given a CFL, CSL, REC or REC, determining whether this language is regular is undecidable.

*Note**: Two popular undecidable problems are halting problem of TM and PCP (Post Correspondence Problem). Semi-decidable Problems***

Undecidability problems:

For an undecidable language, there is no Turing Machine which accepts the Language and makes a decision for every input string w (TM can make decision for Some input string though)

A decision problem P is called "undecidable if the language L of all yes instances to P is not decidable recursive languages, but sometimes, they may be recursively enumerable languages.



Example:

The halting problem of Turing machine

The mortality problem

The mortal matrix problem

The Post correspondence problem, etc.

Church's Thesis for Turing Machine

In 1936, A method named as lambda-calculus was created by Alonzo Church in which the Church numerals are well defined, i.e. the encoding of natural numbers. Also in 1936, Turing machines (earlier called theoretical model for machines) was created by Alan Turing, that is used for manipulating the symbols of string with the help of tape.

Church Turing Thesis:

Turing machine is defined as an abstract representation of a computing device such as hardware in computers. Alan Turing proposed Logical Computing Machines (LCMs), i.e. Turing's expressions for Turing Machines. This was done to define algorithms properly. So, Church made a mechanical method named as 'M' for manipulation of strings by using logic and mathematics. This method M must pass the following statements:

- Number of instructions in M must be finite.
- Output should be produced after performing finite number of steps.
- It should not be imaginary, i.e. can be made in real life.
- It should not require any complex understanding.

Using these statements Church proposed a hypothesis called **Church's Turing thesis** that can be stated as: "The assumption that the intuitive notion of computable functions can be identified with partial recursive functions."

Or in simple words we can say that "Every computation that can be carried out in the real world can be effectively performed by a Turing Machine."

In 1930, this statement was first formulated by Alonzo Church and is usually referred to as Church's thesis, or the Church-Turing thesis. However, this hypothesis cannot be proved. The recursive functions can be computable after taking following assumptions:

1. Each and every function must be computable.
2. Let 'F' be the computable function and after performing some elementary operations to 'F', it will transform a new function 'G' then this function 'G' automatically becomes the computable function.
3. If any functions that follow above two assumptions must be states as computable function.

Describe a universal Turing machine

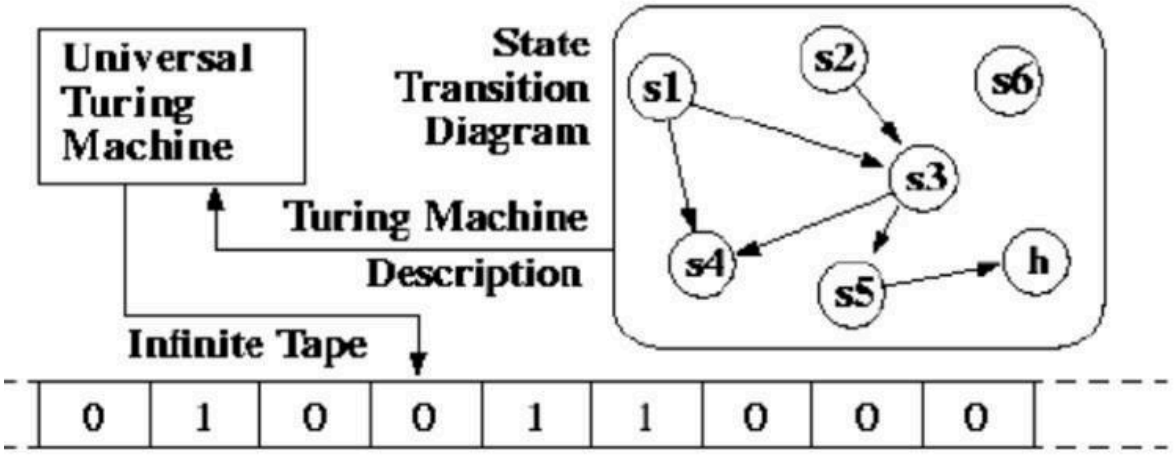
A Turing Machine is the mathematical tool equivalent to a digital computer It was Suggested by the mathematician Turing in the 30s, and has been since then the Most widely used model of computation in computability and complexity theory The model consists of an input output relation that the machine computes. The input Is given in binary form on the machine's tape, and the output consists of the contents Of the tape when the machine halts

What determines how the contents of the tape change is a finite state machine (or FSM, also called a finite automaton) inside the Turing Machine. The FSM is Determined by the number of states it has, and the transitions between them Determined by the number of states it has, and the transitions between them

At every step, the current state and the character read on the tape determine the Next state the FSM will be in the character that the machine will output on the tape (possibly the one read, leaving the contents unchanged), and which direction the Head moves in, left or right

The problem with Turing Machines is that a different one must be constructed for Every new computation to be performed, for every input output relation

This is why we introduce the notion of a universal turing machine (UTM) which Along with the input on the tape, takes in the description of a machine M. The UTM Can go on then to simulate M on the rest of the contents of the input tape A universal turing machine can thus simulate any other machine



Diagonalization Language:

- It is also a language which does not accepted by the turing machine.
- It comes under Non-Recursive enumerable language, which means the strings which are not accepted.
- Its represented as (Ld), where the strings Wi is not in L(Mi).
- Which means Wi !=L(Mi).
- Its is the technique used to find: - Non-Recursive enumerable language
- It proves UNDECIDABILITY of halting problem, which means there will no algorithm and final states to completed the specific problem.
- We have the only way to process through TM is in the binary, not only the TM but every regular expressions are should be in (0,1)'s or (a, b),……
- The strings which are in non-recursive enumerable language are converted using technique called diagonalization.

Enumerating the Binary Strings

- In many proofs involving Turing machines, we need to enumerate the binary strings and encode Turing machines so that we can refer to the *i*th binary string as w_i and the *i*th Turing machine as M_i.
- The binary strings are easy to enumerate. If *w* is a binary string, we shall treat 1*w* as the binary integer *i* so we can call *w* the *i*th string.
- Using this encoding, the empty string is the first string, 0 the second, 1 the third, 00 the fourth, 01 the fifth, and so on. Henceforth we will refer to the *i*th string as w_i

L_d is not Recursively Enumerable

- We define L_d, the diagonalization language, as follows:
    1. Let w_1, w_2, w_3, . . . be an enumeration of all binary strings.
    2. Let M_1, M_2, M_3, . . . be an enumeration of all Turing machines.
    3. Let L_d = { w_i | w_i is not in L(M_i) }.
- Theorem: - L_d is not a recursively enumerable language.
  Proof: -
    1. Suppose L_d = L(M_i) for some TM M_i.
    2. This gives rise to a contradiction. Consider what M_i will do on an input string w_i.
        - If M_i accepts w_i, then by definition w_i cannot be in L_d.
        - If M_i does not accepts w_i, then by definition w_i is in L_d.

3. Since w_i can neither be in L_d nor not be in L_d, we must conclude there is no Turing machine that can define L_d.

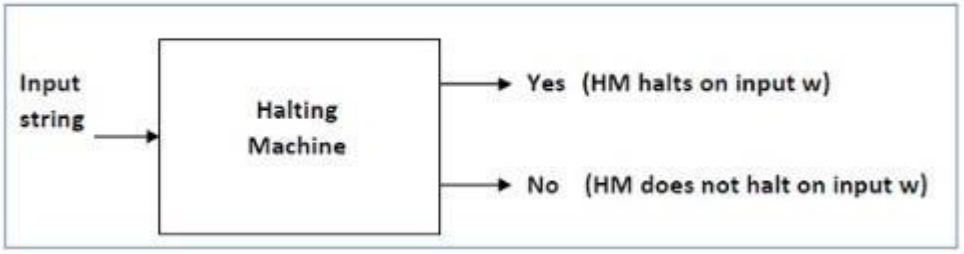
Halting problem

Input A machine with Turing and an input string w

Problem Does the Turing machine complete the w-string computation in a finite number of steps? Either yes or no must be the answer

Proof We will first assume that there is such a Turing machine to solve this problem, and then we will demonstrate that it contradicts itself. This Turing machine would be called a Halting machine that produces in a finite amount of time a 'yes' or 'no The performance comes as 'yes' if the stopping machine finishes in a finite amount of time, otherwise as 'no'

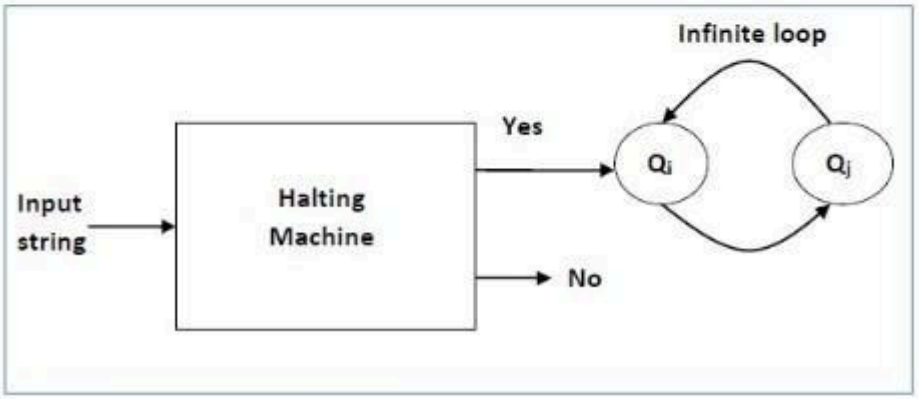
The Halting Computer block diagram is as follows



We're now going to build an inverted stopping machine (HM)' as -

- If H returns YES, then loop forever.
- If H returns NO, then halt.

The following is a 'Inverted Stopping Unit' block diagram -



In addition, as follows, a machine (HM)2 whose input itself is constructed

If (HM)2 halts on input, loop forever, Else, halt

We have a contradiction here. The stopping question is therefore undecidable

Rice's theorem:

Rice theorem states that any non-trivial semantic property of a language which is recognized by a Turing machine is undecidable. A property, P, is the language of all Turing machines that satisfy that property.

Formal Definition
If P is a non-trivial property, and the language holding the property, L_p , is recognized by Turing machine M, then L_p = {<M> | L(M) ∈ P} is undecidable.

Description and Properties

- Property of languages, P, is simply a set of languages. If any language belongs to P (L ∈ P), it is said that L satisfies the property P.
- A property is called to be trivial if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages.
- A non-trivial property is satisfied by some recursively enumerable languages and are not satisfied by others. Formally speaking, in a non-trivial property, where L ∈ P, both the following properties hold:
    - **Property 1** − There exists Turing Machines, M1 and M2 that recognize the same language, i.e. either (<M1>, <M2> ∈ L) or (<M1>,<M2> ∉ L)
    - **Property 2** − There exists Turing Machines M1 and M2, where M1 recognizes the language while M2 does not, i.e. <M1> ∈ L and <M2> ∉ L

Proof

Suppose, a property P is non-trivial and φ ∈ P.
Since, P is non-trivial, at least one language satisfies P, i.e., $L(M_0) \in P$, ∋ Turing Machine M_0.
Let, w be an input in a particular instant and N is a Turing Machine which follows −

- On input x
- Run M on w
- If M does not accept (or doesn't halt), then do not accept x (or do not halt)
- If M accepts w then run M_0 on x. If M_0 accepts x, then accept x.

A function that maps an instance ATM = {<M,w>| M accepts input w} to a N such that

- If M accepts w and N accepts the same language as M_0, Then $L(M) = L(M_0) \in p$
- If M does not accept w and N accepts φ, Then $L(N) = \varphi \notin p$

Since A_{TM} is undecidable and it can be reduced to Lp, Lp is also undecidable

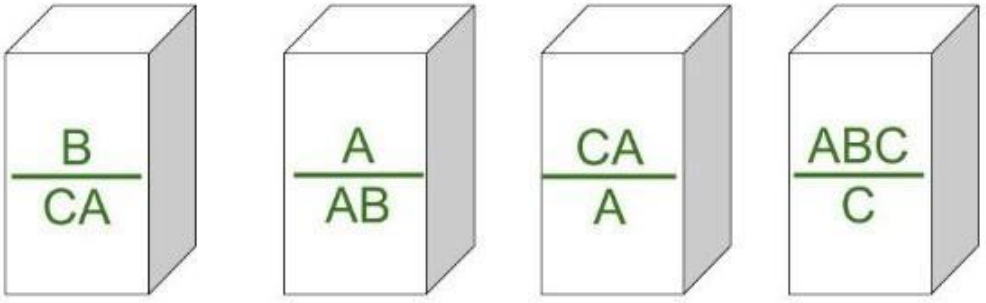
Post Correspondence Problem:

Post Correspondence Problem is a popular undecidable problem that was introduced by Emil Leon Post in 1946. It is simpler than Halting Problem. In this problem we have N number of Dominos (tiles). The aim is to arrange tiles in such order that string made by Numerators is same as string made by Denominators. In simple words, lets assume we have two lists both containing N words, aim is to find out concatenation of these words in some sequence such that both lists yield same result. Let's try understanding this by taking two lists A and B

A=[aa, bb, abb] and B=[aab, ba, b]

Now for sequence 1, 2, 1, 3 first list will yield aabbaaabb and second list will yield same string aabbaaabb. So the solution to this PCP becomes 1, 2, 1, 3. Post Correspondence Problems can be represented in two ways:

1. Domino's Form :



2. Table Form :

	Numerator	Denominator
1	B	CA
2	A	AB
3	CA	A
4	ABC	C

Lets consider following examples. **Example-1:**

	A	B
1	1	111
2	10111	10
3	10	0

Explanation –

- **Step-1:** We will start with tile in which numerator and denominator are starting with same number, so we can start with either 1 or 2. Lets go with **second** tile, string made by numerator- 10111, string made by denominator is 10.
- **Step-2:** We need 1s in denominator to match 1s in numerator so we will go with **first** tile, string made by numerator is 10111 1, string made by denominator is 10 111.
- **Step-3:** There is extra 1 in numerator to match this 1 we will add **first** tile in sequence, string made by numerator is now 10111 1 1, string made by denominator is 10 111 111.
- **Step-4:** Now there is extra 1 in denominator to match it we will add **third** tile, string made by numerator is 10111 1 1 10, string made by denominator is 10 111 111 0.

- `Final Solution - 2 1 1 3`
- `String made by numerators: 101111110`
- `String made by denominators: 101111110`
- As you can see, strings are same.

	A	B
1	100	1
2	0	100
3	1	00

Explanation –

- Step-1: We will start from tile 1 as it is our only option, string made by numerator is 100, string made by denominator is 1.
- Step-2: We have extra 00 in numerator, to balance this only way is to add tile 3 to sequence, string made by numerator is 100 1, string made by denominator is 1 00.
- Step-3: There is extra 1 in numerator to balance we can either add tile 1 or tile 2. Lets try adding tile 1 first, string made by numerator is 100 1 100, string made by denominator is 1 00 1.
- Step-4: There is extra 100 in numerator, to balance this we can add 1st tile again, string made by numerator is 100 1 100 100, string made by denominator is 1 00 1 1 1. The 6th digit in numerator string is 0 which is different from 6th digit in string made by denominator which is 1.

We can try unlimited combinations like one above but none of combination will lead us to solution, thus this problem does not have solution.

Modified Post Correspondence Problem:

The modified PCP (MPCP) has one additional requirement than PCP, First pair in the solution is the first pair in the list Remaining sequence in the Solution can be in any order.

The MPCP problem is to determine whether there exist a sequence of one or more integer i1,i2,i3,……im such that
Wi,wi1,..wim = xi,xi1,…xim

First element of list A:
w1,w2,w3…..wn
First element of list B:
x1,x2,x3,…..xm
example:

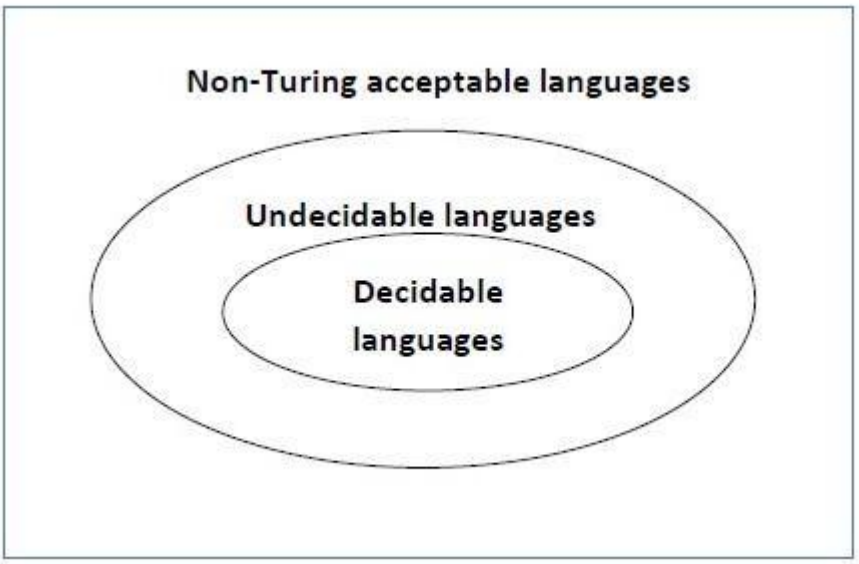
sno	list A	list B
1	10	10
2	11	011
3	110	11

MPCP has a solution

W1w3w2 = x1x3x2

undecidable problems about languages.

For an undecidable language, there is no Turing Machine which accepts the language and makes a decision for every input string **w** (TM can make decision for some input string though). A decision problem **P** is called “undecidable” if the language **L** of all yes instances to **P** is not decidable. Undecidable languages are not recursive languages, but sometimes, they may be recursively enumerable languages.



In the Theory of Computation, problems can be classified into decidable and undecidable categories based on whether they can be solved using an algorithm. A **decidable problem** is one for which a solution can be found in a finite amount of time, meaning there exists an algorithm that can always provide a correct answer. While an **undecidable problem** is one where no algorithm can be constructed to solve the problem for all possible inputs. In this article, we will discuss Decidable and Undecidable problems in detail.

What are Decidable Problems?

A problem is said to be **Decidable** if we can always construct a corresponding **algorithm** that can answer the problem correctly. We can intuitively understand Decidable issues 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** to this problem, we can easily construct 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 that **halts** on every input with an answer- **yes or no**. It is also important to know that these problems are termed **Turing Decidable** since 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 the Turing Machine rejects. Such problems are termed as **Turing Recognisable** problems.

Example

We will now consider some 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.

What are Undecidable Problems?

The problems for which we can't construct an algorithm that can answer the problem correctly in 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 intuitively 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.

Example

These are few important **Undecidable Problems**:

- **Whether a CFG generates all the strings or not:** As a Context Free Grammar (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 Context Free Language (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 cannot find from the production rules of the CFL whether it is regular or not.