# *FORMAL LANGUAGE AND AUTOMATA THEORY*

## *UNIT – VI: Computability*

*Decidable and Un-decidable Problems, Halting Problem of Turing Machines, Post's Correspondence Problem, Modified Post's Correspondence Problem, Classes of P and NP, NPHard and NP-Complete Problems.*

## ***Decidable and Un-decidable Problems***

### ***Decidable***

*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***

*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*
*A semi-decidable problem is subset of undecidable problems for which Turing machine will always halt in finite amount of time for answer as 'yes' and may or may not halt for answer as 'no'.*
*Relationship between semi-decidable and decidable problem has been shown in Figure 1.*

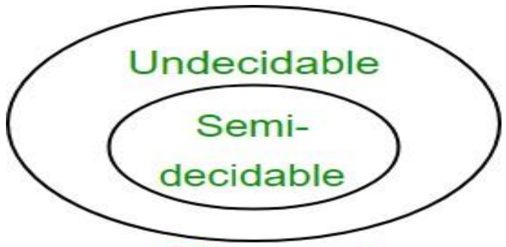


Figure 1

***Rice's Theorem***
*Every non-trivial (answer is not known) problem on Recursive Enumerable languages is un-decidable. e.g.; If a language is Recursive Enumerable, its complement will be recursive enumerable or not is un-decidable.*

***Class work:***

**1)** *What do you mean by Computability?*

***Homework:***

*Q1). Explain the Rice's Theorem.*

***Important and Previous JNTU Questions:***

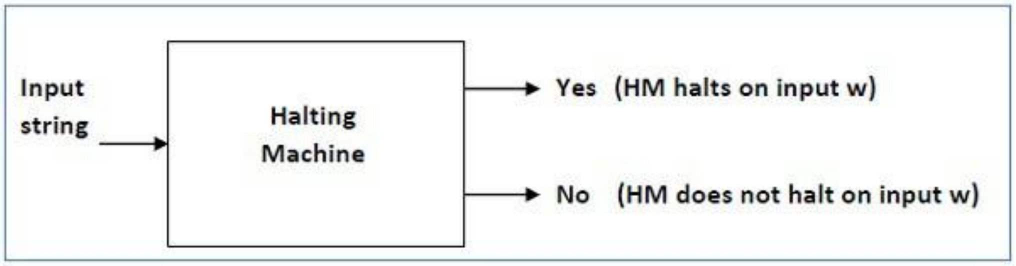
*1) What is Decidable and un-Decidable? Explain?*

***<u>Halting Problem of Turing Machines</u>***

***Input*** *− A Turing machine and an input string **w**.*

***Problem*** *− Does the Turing machine finish computing of the string **w** in a finite number of steps? The answer must be either yes or no.*

***Proof*** *− At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a* ***Halting machine*** *that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine −*



*Now we will design an* ***inverted halting machine (HM)'*** *as −*

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

*The following is the block diagram of an 'Inverted halting machine' −*

*Further, a machine (HM)2 which input itself is constructed as follows −*

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

*Here, we have got a contradiction. Hence, the halting problem is* ***undecidable****.*

## *Previous Questions*

1) Write the block diagram for Halting state Problem.
2). Whether a Halting Problem is Decidable or not? Justify.
3) What is Halting Problem of Turing Machine? Is it decidable or not? Explain?

## ***Post's Correspondence Problem, Modified Post's Correspondence Problem***

*The Post Correspondence Problem (PCP), introduced by Emil Post in 1946, is an undecidable decision problem. The PCP problem over an alphabet ∑ is stated as follows −*

*Given the following two lists, **M** and **N** of non-empty strings over ∑ −*

$M = (x_1, x_2, x_3, \ldots\ldots\ldots, x_n)$

$N = (y_1, y_2, y_3, \ldots\ldots\ldots, y_n)$

*We can say that there is a Post Correspondence Solution, if for some* $i_1, i_2, \ldots\ldots\ldots\ldots_k$, *where* $1 \le i_j \le n$, *the condition* $x_{i1} \ldots\ldots x_{ik} = y_{i1} \ldots\ldots\ldots y_{ik}$ *satisfies.*

## Example 1

*Find whether the lists*

*M = (abb, aa, aaa) and N = (bba, aaa, aa)*

*have a Post Correspondence Solution?*

## Solution

| | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| **M** | Abb | aa | aaa |
| **N** | Bba | aaa | aa |

*Here,*

$x_2x_1x_3$ ***= 'aaabbaaa'***

*and* ***$y_2y_1y_3$ = 'aaabbaaa'***

*We can see that*

***$x_2x_1x_3 = y_2y_1y_3$***

*Hence, the solution is **i = 2, j = 1, and k = 3.***

## *Example 2*

*Find whether the lists **M = (ab, bab, bbaaa)** and **N = (a, ba, bab)** have a Post Correspondence Solution?*

### *Solution*

| | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| ***M*** | *Ab* | *bab* | *bbaaa* |
| ***N*** | *A* | *ba* | *bab* |

*In this case, there is no solution because* **−**

**$|x_2x_1x_3| \neq |y_2y_1y_3|$** *(Lengths are not same)*

*Hence, it can be said that this Post Correspondence Problem is **undecidable**.*

### *Important andPrevious JNTU Questions:*

*1) What is PCP Problem? [JNTUK Regular 2017]*

***<u>Classes of P and NP, NPHard and NP-Complete Problems</u>***

- *Some problems are intractable:*
  *as they grow large, we are unable to solve them in reasonable time*
- *What constitutes reasonable time?*
  - *Standard working definition: polynomial time*
  - *On an input of size n the worst-case running time is $O(n^k)$ for some constant k*

- » *$O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$, $O(2^n)$, $O(n^n)$, $O(n!)$*
- » ***Polynomial time**: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$*
- » ***Not in polynomial time**: $O(2^n)$, $O(n^n)$, $O(n!)$*

x *Are some problems solvable in polynomial time?*

- » *Of course: many algorithms we've studied provide polynomial-time solutions to some problems*

x *Are all problems solvable in polynomial time?*

- » *No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given*

x *Most problems that do not yield polynomial-time algorithms are either optimization or decision problems.*

x ***Optimization Problems***

- o *An optimization problem is one which asks, "What is the optimal solution to problem X?"*
- o *Examples:*
  - ▪ *0-1 Knapsack*
  - ▪ *Fractional Knapsack*
  - ▪ *Minimum Spanning Tree*

x ***Decision Problems***

- o *An decision problem is one with yes/no answer*
- o *Examples:*
  - ▪ *Does a graph G have a MST of weight $\leq W$?*

x *An optimization problem tries to find an optimal solution whereas A decision problem tries to answer a yes/no question*

x *Many problems will have decision and optimization versions*

- o *Eg: Traveling salesman problem*
  - ▪ *optimization: find hamiltonian cycle of minimum weight*
  - ▪ *decision: is there a hamiltonian cycle of weight $\leq k$*

x *Some problems are decidable, but intractable:*

*as they grow large, we are unable to solve them in reasonable time*

  - o *Is there a polynomial-time algorithm that solves the problem?*

***The Class P***

***P**: the class of decision problems that have polynomial-time deterministic algorithms.*

  - » *That is, they are solvable in O(p(n)), where p(n) is a polynomial on n*
  - » *A deterministic algorithm is (essentially) one that always computes the correct answer*

***Why polynomial**?*

  - » *if not, very inefficient*
  - » *nice closure properties*
    - ✓ *the sum and composition of two polynomials are always polynomials too*

x *The class P consists of those problems that are solvable in polynomial time.*

x *More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem*

x *The key is that n is the **size of input***

***Examples of P Problems**:*

- *Fractional Knapsack*
- *MST*
- *Sorting*
- *Others?*

***NP Problems***

- *NP is not the same as non-polynomial complexity/running time. NP does not stand for not polynomial.*
- *NP = Non-Deterministic polynomial time*
- *NP means verifiable in polynomial time*
- *Verifiable?*
  - » *If we are somehow given a 'certificate' of a solution we can verify the*

*legitimacy in polynomial time*

***<u>NP</u>**: the class of decision problems that are solvable in polynomial time on a nondeterministic machine (or with a nondeterministic algorithm)*

- *(A <u>determinstic</u> computer is what we know)*
- *A <u>nondeterministic</u> computer is one that can "guess" the right answer or solution*
  - *Think of a nondeterministic computer as a parallel machine that can freely spawn **an infinite number** of processes*
- *Thus NP can also be thought of as the class of problems*
  - *whose solutions can be verified in polynomial time*
- *Note that NP stands for "Nondeterministic Polynomial-time".*

***<u>Examples of NP Problems:</u>***

- *Fractional Knapsack*
- *MST*
- *Sorting*
- *Others?*
  - *Hamiltonian Cycle (Traveling Salesman)*
  - *Graph Coloring*
  - *Satisfiability (SAT)*

***<u>P is a subset of NP</u>***

- *Since it takes polynomial time to run the program, just run the program and get a solution*
- *But is NP a subset of P?*
  - ✓ *No one knows if P = NP or not.*

***<u>What is not in NP?</u>***

x *Undecidable problems*

- » *Given a polynomial with integer coefficients, does it have integer roots*
- » *Hilbert's nth problem*
- » *Impossible to check for all the integers*
- » *Even a non-deterministic TM has to have a finite number of states!*
- » *More on decidability later*

x *Tautology*

- » *A boolean formula that is true for all possible assignments*
- » *Here just one 'verifier' will not work. You have to try all possible values*

***NP-Complete Problem:***

- ➢ *A decision problem D is NP-complete iff*
  1. *D NP*
  2. *every problem in NP is polynomial-time reducible to D*
- ➢ *Cook's theorem (1971): CNF-sat is NP-complete*

***Reduction**:*

- ➢ *A problem R can be reduced to another problem Q if any instance of R can be rephrased to an instance of Q, the solution to which provides a solution to the instance of R*
  - » *This rephrasing is called a transformation*
- ➢ *Intuitively: If R reduces in polynomial time to Q, R is "no harder to solve" than Q*
- ➢ *Example: lcm(m, n) = m * n / gcd(m, n),*

*lcm(m,n) problem is reduced to gcd(m, n) problem*

***NP Hard and NP Complete**:*

- *If R is polynomial-time reducible to Q, we denote this R $_p$dQ*
- *Definition of NP-Hard and NP-Complete:*
  - *If all problems R* ***NP*** *are polynomial-time reducible to Q, then Q is NP-Hard*
  - *We say Q is NP-Complete if Q is NP-Hard and Q* ***NP***
- *If R $_p$dQ and R is NP-Hard, Q is also NP-Hard*

***IMP Questions***

1) *Discuss the P and NP Problems with an examples.*
2) *What is reducible means?*
3) *what are NP Complete and NP Hard.*

***Important and Previous JNTU Questions:***

1) *What are P and NP class of Languages? What is NP Complete and give examples?*

*[JNTUK Regular 2017]*