Southeast University

Department of Computer Science and Engineering

Course Code: CSE441

Section: 03

Course Title: Theory of Computing

Assignment: Write About:

1) Decidable Languages
2) Undecidability of the Halting Problem
3) Diagonalization and Reduction
4) P, NP, NPC Class Problems

Date of Submission: 21/09/2025

Submitted To: Nahid Hasan

Lecturer, Department of Computer Science & Engineering

Initial: NH

Submitted By:

Name: Nazmul Hasan

ID: 2023100000081

Batch: 64

1. **Decidable Languages**

In computer science, a language is simply a set of strings formed from some alphabet. For example, if our alphabet is {0,1}, then strings like 0101 or 111 belong to some language depending on the rules of that language.

A decidable language (also called a *recursive language*) is one for which there exists an algorithm or a Turing Machine that can always halt and give the correct answer:

- Accept if the input string belongs to the language.
- Reject if the input string does not belong to the language.

Examples

- The set of all even binary numbers is decidable, because a program can simply check the last digit. If the last digit is 0, it accepts; otherwise, it rejects.
- All regular languages (recognized by finite automata) and all context-free languages (recognized by pushdown automata) are decidable.

Importance

Decidable languages are important because they represent problems that computers can handle reliably. In the real world, programming languages, compilers, and pattern-matching tools rely on decidable problems.

In short:
Decidable = solvable with a guarantee of termination and correctness.

2. **Undecidability of the Halting Problem**

The Halting Problem is one of the most famous problems in theoretical computer science. It asks:

Given a program (or Turing Machine) and an input, can we always know whether the program will eventually halt (stop) or run forever?

In 1936, Alan Turing proved that the Halting Problem is undecidable.

Proof Idea (in simple words)

1. Suppose we imagine a magical program H that can decide the halting problem.
    - Input: (Program P, Input x).
    - Output: "Yes" if P(x) halts, "No" if P(x) runs forever.

2. Using this, we can construct another program D that behaves paradoxically:
    - If H says that program halts, then D goes into an infinite loop.
    - If H says program loops forever, then D halts.
3. Now the question: what happens when D is given its own description as input?
    - If H predicts "halt," then D runs forever.
    - If H predicts "infinite loop," then D halts.
    - In both cases, a contradiction appears.

Therefore, the assumption that H exists must be false. Hence, the Halting Problem is undecidable.

Why It Matters

- This result shows there are limits to computation.
- No matter how powerful computers become, there will always be problems they cannot solve.
- Many other undecidable problems (e.g., equivalence of programs, program correctness in general) are linked to the Halting Problem.

3. Diagonalization and Reduction

These are two fundamental proof techniques used in computability theory.

a) Diagonalization

- Originally used by Cantor to prove that real numbers are uncountable.
- Applied in computation, it shows that some languages cannot be decided by any Turing Machine.

Idea in simple form**:**

- Suppose we list all Turing Machines in order: M1, M2, M3, ….
- We also list all possible input strings: w1, w2, w3, ….
- Create a table where each entry tells whether machine Mi accepts string wj.
- Now, construct a new language L that differs from this table on the diagonal entries.
- By construction, L cannot be decided by any of the listed machines.

This proves there exist languages that are undecidable.

b) Reduction

- Reduction is a method to prove a new problem is hard or undecidable.
- If we can transform a known hard/undecidable problem (A) into another problem (B), then problem B must also be hard/undecidable.

Example**:**

- Halting Problem is undecidable.
- Suppose we transform Halting Problem into the problem "Does program print hello?"
- If we could solve the "hello problem," we would solve the Halting Problem too.
- Since Halting is undecidable, the "hello problem" must also be undecidable.

Reduction is widely used not only for undecidability but also in complexity theory (P, NP, NP-Complete).

4. P, NP, NPC Class Problems

Not all problems are equally hard. To classify them, we use P, NP, and NPC.

P (Polynomial Time)

- P = class of problems solvable in polynomial time by a deterministic Turing Machine.
- Means they are efficiently solvable.
- Example: Sorting a list, finding the shortest path in a graph, matrix multiplication.

NP (Nondeterministic Polynomial Time)

- NP = class of problems for which a proposed solution can be verified quickly (in polynomial time).
- But finding the solution may be very hard.
- Example: Sudoku puzzle, Hamiltonian Path, Boolean SAT problem.

[**Note**: Every problem in P is also in NP, because if you can solve it quickly, you can also check the solution quickly.]

NP-Complete (NPC)

- NP-Complete problems are the hardest problems in NP.
- A problem is NP-Complete if:
    1. It is in NP.
    2. Every other NP problem can be reduced to it in polynomial time.

Examples:

- SAT (Boolean satisfiability problem).
- Traveling Salesman Decision Problem.
- 3-SAT, Clique problem.

Big Question: Is P = NP?
Nobody knows yet. If solved, it will change the future of computer science, security, and AI.