ABHYASMITRA.IN
STUDY NOTES

THEORY OF COMPUTATIONS

Unit V — Turing Machine and Computability Theory

Topics Covered (09 Hours)

Basic Turing Machine Model, Tape & Head
Formal 7-Tuple Definition & ID Representation
Designing & Constructing Deterministic TMs (DTMs)
The Universal Turing Machine (UTM) Concept
The Church-Turing Thesis & Machine Power Comparison
The Turing Machine Halting Problem Undecidability Proof
Decidable & Undecidable Problems & Post Correspondence (PCP)
Recursive Enumerability & Reducibility Theory
Complexity Classes: Class P & Class NP Definitions
NP-Completeness, NP-Hardness & The Cook-Levin Theorem

Compiled in a simple, easy-to-understand, book style format
for students of Computer Science and Engineering

TABLE OF CONTENTS

5.1 Turing Machines: Model and Design

Although PDAs are more powerful than finite automata, they still cannot recognize languages like L = {a^n b^n c^n | n ≥ 0} which require multiple simultaneous comparisons. A Turing Machine (TM), proposed by Alan Turing in 1936, represents the ultimate mathematical model of a general-purpose computer. It has an infinite memory tape and is capable of moving left, right, reading, and writing symbols, modeling any algorithm.

The Basic Turing Machine Model

The Turing machine model consists of three main parts:

- **Infinite Tape:** A tape divided into cells, extending infinitely in both directions (or infinitely to the right). Each cell can hold exactly one symbol from the tape alphabet. Unused cells contain a special blank symbol 'B'.
- **Read/Write Head:** A pointer that points to the cell currently being scanned. It can read the symbol, write a new symbol to overwrite it, and move one cell to the Left (L) or to the Right (R).
- **Finite Control Unit:** A state transition engine that determines the next state, the symbol to write, and the direction of head movement based on the current state and scanned symbol.

Turing Machine Formal Definition (7-Tuple)

A Turing Machine is formally defined as a 7-tuple:

M = (Q, Σ, Γ, δ, q0, B, F)

Where:

1. Q is a finite, non-empty set of internal States.

2. Σ is a finite set of input symbols (Input Alphabet).

3. Γ is a finite set of tape symbols, where Σ ⊂ Γ.

4. δ is the Transition Function mapping Q x Γ → Q x Γ x {L, R}.

5. q0 ∈ Q is the initial Start State.

6. B ∈ Γ is the Blank symbol, where B ∉ Σ.

7. F ⊆ Q is the set of Final (Accepting) States.

Constructing a DTM for a^n b^n c^n

Let us trace how a Deterministic Turing Machine (DTM) decides L = {a^n b^n c^n}. We write input strings surrounded by blanks, e.g., 'aaabbbccc'. The head starts at the first 'a'. The algorithm is:

- 1. Read 'a', replace it with mark 'X', and transition to state q1.

- 2. Move right in state q1, bypassing remaining 'a's and 'Y's, until we hit the first 'b'.
- 3. Replace 'b' with mark 'Y', transition to state q2, and move right.
- 4. Move right in state q2, bypassing remaining 'b's and 'Z's, until we hit the first 'c'.
- 5. Replace 'c' with mark 'Z', transition to state q3, and move left.
- 6. Move left in state q3, bypassing all characters, until we reach the character immediately to the right of the last 'X'.
- 7. Repeat the loop. If all symbols are correctly matched to X, Y, and Z, the machine enters the final accepting state, proving the string is in the language.

5.2 Advanced Computability & The Halting Problem

The Universal Turing Machine (UTM) is a Turing machine capable of simulating any other Turing machine. It takes two inputs: the description of a target Turing machine M, and an input string w. It then simulates the execution of M on w. This is the theoretical basis of the stored-program computer.

The Church-Turing Thesis

The Church-Turing Thesis states that any algorithmic process that can be computed at all can be computed by a Turing Machine. It is a thesis, not a theorem, because 'algorithm' is an informal concept, whereas Turing machine is a formal mathematical one. However, no computer model has ever been found that is more powerful than a Turing machine.

Comparison of Automata Power

Automaton Model	Memory Type	Supported Language Class (Chomsky)	Computational Power
Finite Automata (DFA/NFA)	None (finite states only).	Regular Languages (Type 3).	Lowest. Cannot count or match nesting.
Pushdown Automata (PDA)	LIFO Stack (infinite depth).	Context-Free Languages (Type 2).	Medium. Can match nested brackets.
Turing Machine (TM)	Infinite random-access Tape.	Recursively Enumerable (Type 0).	Highest. Models any computable algorithm.

The Halting Problem

The Halting Problem is the most famous undecidable problem in computer science. It asks: is it possible to write a program H that can check any program P and its input w, and determine whether P will eventually stop running (halt) or run forever in an infinite loop?

Undecidability Proof by Diagonalization: Suppose such a program H exists. H(P, w) returns 'Halt' if P halts, and 'Loop' if P loops. Now, we construct a new program D that takes P as input. D calls H(P, P). If H says P halts, D enters an infinite loop. If H says P loops, D halts immediately. Now, what happens if we run D(D)? If H(D, D) says 'Halt', then D(D) loops forever. If H(D, D) says 'Loop', then D(D) halts. In both cases, we reach a contradiction. Therefore, H cannot exist, proving the halting problem is Undecidable.

5.3 Decidability, Reducibility, and Complexity

We categorize problems based on whether they can be solved by computers (Decidability) and how many resources they consume (Complexity).

Decidable vs. Undecidable Problems

- **Decidable Problem:** A problem that can be solved by a Turing machine that is guaranteed to halt on all inputs (a decider machine). It always returns a clean 'Yes' or 'No'.
- **Undecidable Problem:** A problem for which no halting algorithm can be written. The halting problem is the classic example.
- **Post Correspondence Problem (PCP):** An undecidable problem involving dominoes. Given a set of dominoes, each with a top string and bottom string, is there a sequence of dominoes such that the concatenated top strings match the concatenated bottom strings? PCP is undecidable, serving as a tool to prove other problems undecidable via reducibility.
- **Reducibility:** A technique to prove a new problem B is undecidable by showing that if we could solve B, we could use that solution to solve a known undecidable problem A (like the halting problem).

Complexity Classes: P vs. NP

In complexity theory, we classify decidable problems based on how their execution time scales with input size n:

- **Class P (Polynomial Time):** Problems that can be solved by a Deterministic Turing Machine in time O(n^k) for some constant k. These are considered computationally tractable (e.g., sorting, shortest path).
- **Class NP (Nondeterministic Polynomial Time):** Problems for which a candidate solution can be verified by a Deterministic Turing Machine in polynomial time, or solved by a Nondeterministic Turing Machine in polynomial time. (e.g., Sudoku, TSP).
- **P vs. NP Problem:** The greatest open problem in computer science. It asks whether every problem whose solution can be verified quickly (NP) can also be solved quickly (P). Most scientists believe P != NP.
- **NP-Complete Problems:** The hardest problems in NP. If any single NP-Complete problem can be solved in polynomial time (P), then all problems in NP can be solved in polynomial time (meaning P = NP). The Cook-Levin theorem proved that the Boolean Satisfiability (SAT) problem is NP-Complete.

Unit Summary

This unit explored Turing Machines, computability limits, and complexity theory.

- Turing Machines model general-purpose computers using an infinite tape and read/write head.
- Universal Turing Machines can simulate any other TM. The Church-Turing thesis state all algorithms are TM-computable.
- The Halting Problem is undecidable, proved using diagonalization.
- Reducibility is used to prove problems undecidable. PCP is a classic undecidable domino matching puzzle.
- Class P contains tractable problems, while Class NP contains quickly verifiable problems.
- NP-Complete problems are the hardest problems in NP; solving one in polynomial time proves P = NP.