Parul® University
Vadodara, Gujarat
NAAC GRADE A++

Information and Communication Technology

Undecidability

Study Guide

Prof. Riddhi Atulkumar Mehta
CSE, PIT
Parul University

5.1 Church Turing thesis

Historical Background

- **Alonzo Church** (1936): Introduced **λ-calculus**, a formal system for expressing computation via function abstraction and application.
- **Alan Turing** (1936): Independently developed the **Turing Machine**, a theoretical machine to model algorithmic processes.
- Both proved the same class of computable functions—leading to the **Church-Turing Thesis**.

The Church-Turing Thesis

- "A function is effectively computable if and only if it is computable by a Turing machine."
- "Effectively computable" = can be computed by a human or machine using an algorithm, without intuition or guesswork.
- The thesis is not a formal theorem, but a philosophical hypothesis supported by overwhelming evidence.

Formal Models of Computation

Model	Inventor	Year	Description
Turing Machine	**Alan Turing**	**1936**	**Machine with infinite tape and head for reading/writing symbols**
λ-Calculus	**Alonzo Church**	**1936**	**Formal system based on variable binding and substitution**
Recursive Functions	**Gödel/Kleene**	**1930s**	**Functions built using basic operations and recursion**
Post Systems	**Emil Post**	**1943**	**Production rules on strings (rewriting systems)**

Turing Machine

- A Turing Machine is a **mathematical model** of computation that operates on an infinite tape with a finite set of rules.
- Capable of simulating any algorithm.
- Forms the basis of modern computing models.

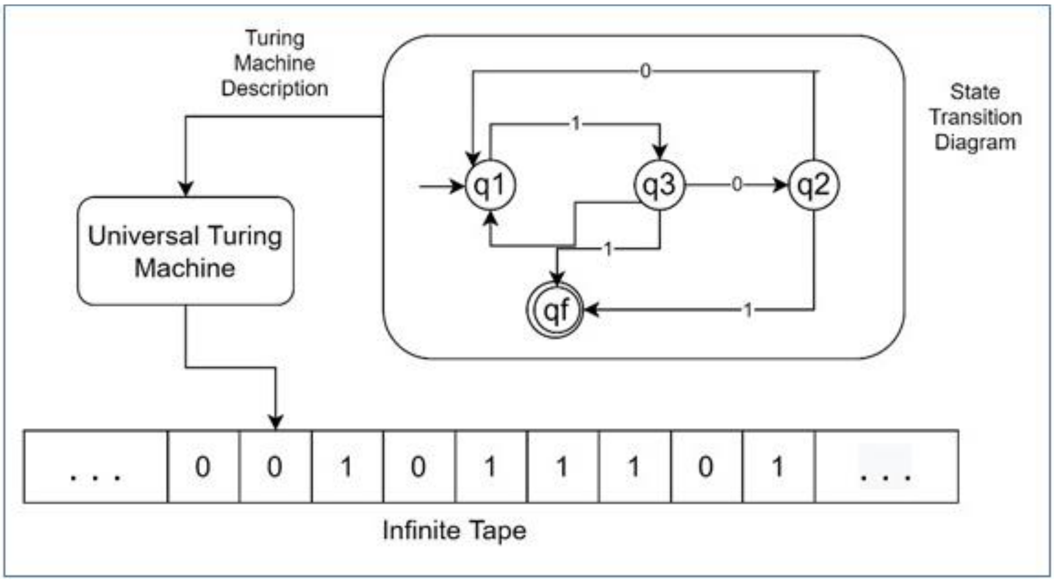
What Does “Computable” Mean?

- A function f: N→N is **computable** if there exists an algorithm (or TM) that produces f(n) for every input n.
- **Example**:
  - Computable: Addition, multiplication, sorting
  - Non-computable: Halting problem, truth of arbitrary mathematical statements

5.2 Universal Turing Machine

Universal Turing Machine

- A Universal Turing Machine is a theoretical model that can simulate any other Turing machine. Which is little complicated but we will see how it actually works.
- If we think about a regular Turing machine as a device built to perform one specific task. So, for example, we might have a Turing machine to add two numbers together or check if a word is a palindrome or not. These machines are task-specific; they do one thing and do it well.
- On the other hand, a Universal Turing Machine can perform any task that a regular Turing machine can do. By taking a description of that machine (let us call it M) and the input for that machine (let us call it X). The Universal Turing Machine, which we will denote as U, processes M and X and then outputs the result of M operating on X.
- The functional block diagram of the machine looks like this –



How Does a Universal Turing Machine Work?

To understand how a Universal Turing Machine works, let us break down its process:

- **Inputs** − The Universal Turing Machine takes two inputs:
    - A description of another Turing machine (M),
    - The input that this machine should process (X).
- **Processing** − The Universal Turing Machine reads the description of M and interprets it as a set of instructions.
- **Simulation** − Using these instructions, the Universal Turing Machine simulates the operations of M on the input X.
- **Output** − The result of this simulation is what M would produce when given X as input.

5.3 The Universal and Diagonalization Languages

The Universal Language Lu

Definition:

- Lu={⟨M,w⟩|M is a TM and M accepts w}
- It contains all encodings of TMs and inputs such that the machine accepts the input.
- Captures the behavior of **any** TM on any input.

Properties of Lu

- **Recursively Enumerable (RE)**:
  There exists a **TM (UTM)** that accepts all strings in Lu

- **Not Recursive (Decidable)**:
  There is **no TM** that can decide for *every* input whether it's in Lu.

The **Halting Problem** is reducible to Lu

Diagonalization Language Ld

Definition:

- Ld={⟨M⟩|M is a TM and M does not accept ⟨M⟩}
- Think of M being run on **its own description**.
- Ld contains all TMs that **do NOT accept themselves**.

Summary of Languages

Language	Definition	RE?	Recursive?
L_u	$\langle M, w\rangle$ where $M(w)$ accepts	✅ Yes	❌ No
L_d	$\langle M\rangle$ where M does **not** accept $\langle M\rangle$	❌ No	❌ No