An Introduction to the Classical Theory of Computation 2

Dr. Hugo García Tecocoatzi

Instituto Tecnológico y de Estudios Superiores de Monterrey

August 18$^{\text{th}}$ 2025

Discussion: Transition Functions in Turing Machines

Questions

- Do we define one transition function for all steps, or a new one for each pair of states?
- Is it necessary to explicitly write all transitions that end in the reject state?

Discussion: Transition Functions in Turing Machines

Questions

- Do we define one transition function for all steps, or a new one for each pair of states?
- Is it necessary to explicitly write all transitions that end in the reject state?

Clarifications

- **Single function:** The transition function δ is one global function. We list its rules $(q, a) \mapsto (q', b, D)$ separately for clarity, but together they form δ.
- **Rejection:** Not every rejecting transition needs to be written. Often we write only the key rules, and assume all undefined cases go to q_{reject}.
- Both approaches are acceptable:
  - Explicitly list all transitions, or
  - State "all other transitions $\to q_{\text{reject}}$".

Discussion: Transition Functions in Turing Machines

Questions

- Do we define one transition function for all steps, or a new one for each pair of states?
- Is it necessary to explicitly write all transitions that end in the reject state?

Clarifications

- **Single function:** The transition function δ is one global function. We list its rules $(q, a) \mapsto (q', b, D)$ separately for clarity, but together they form δ.
- **Rejection:** Not every rejecting transition needs to be written. Often we write only the key rules, and assume all undefined cases go to q_{reject}.
- Both approaches are acceptable:
  - Explicitly list all transitions, or
  - State "all other transitions $\rightarrow q_{\text{reject}}$".

$\rightarrow$ *The important thing is that the machine's behavior is unambiguous.*

Solution: TM for $L = 0\,1^n 0$

Algorithm:

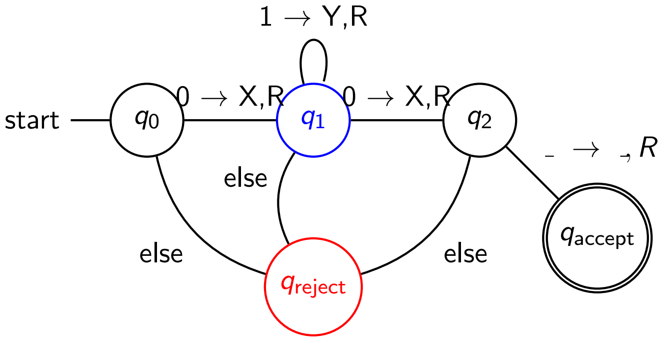
1. Verify the first symbol is 0, move right.
2. Enter a loop scanning one or more 1's.
3. Check that the next symbol is a single 0.
4. Accept if the tape ends after this final 0.

Formal definition:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

- $Q = \{q_0, q_1, q_2, q_{\text{accept}}, q_{\text{reject}}\}$
- $\Sigma = \{0, 1\}, \quad \Gamma = \{0, 1, \text{\_}\}$
- δ:
  - $(q_0, 0) \to (q_1, 0, R)$ (first symbol must be 0)
  - $(q_1, 1) \to (q_1, 1, R)$ (scan over 1's)
  - $(q_1, 0) \to (q_2, 0, R)$ (check final 0)
  - $(q_2, \text{\_}) \to (q_{\text{accept}}, \text{\_}, R)$ (end of input)
  - Otherwise $\to q_{\text{reject}}$

TM State Diagram and Examples



Tests:

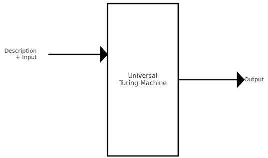
- Accept: 010, 0110, 011110
- Reject: 0, 0010, 011, 1010

Turing machine importance

- Formalizes the concept of an **algorithm**
- **Limits:** Some problems are *not* solvable by any Turing Machine.
- **Theoretical foundation** for:
  - Complexity theory.
  - Programming languages.
  - Modern computer architecture.
- **Universality:** Any computation can be expressed as a Turing Machine.

Outline

The Universal Turing Machine (UTM): A single machine capable of simulating any other Turing Machine.



It's a foundational concept in computer science that demonstrates the universality of computation

Why Universal Turing Machines?

- A normal Turing machine is built for a specific task (e.g., recognizing a language).
- But: Can we build a machine that can run the description of any other Turing Machine?
- Alan Turing answered: Yes! **The Universal Turing Machine (UTM)**.
- Conceptual precursor to modern programmable computers.

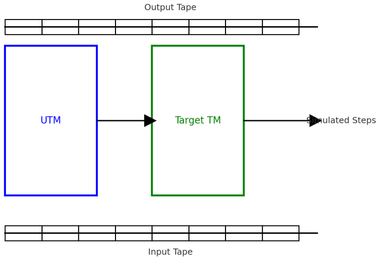
How the UTM Works

The UTM takes as input:

1. The **description of a Turing Machine** (its rules, states, alphabet).
2. The **input string** for that machine.

Then:

- Interprets the machine description.
- Simulates step-by-step execution.
- Produces the same output that the original Turing Machine would.



The UTM is like a "machine interpreter".

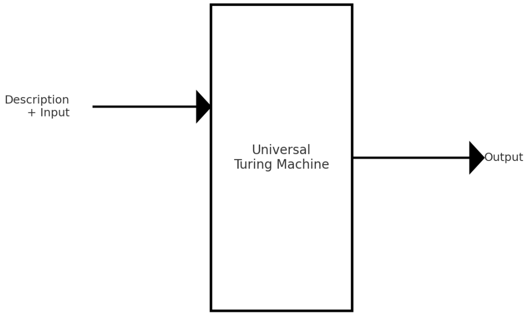
Why the UTM Matters

- **Universality:** One machine can compute anything computable.
- **Basis for Programmability:** Modern computers are essentially Universal Turing Machines.
- **Theoretical Importance:**
  - Demonstrates the idea of "software" (machine descriptions).
  - Leads to study of undecidability (e.g., the Halting Problem).
- **Philosophical impact:** Foundation of the Church–Turing Thesis.

"It states that a function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine"

Universal Turing Machine: Simulation

- A Universal Turing Machine (UTM) can simulate *any* other Turing Machine.
- Input to the UTM consists of:
  1. The **description** (encoded transition rules) of a Turing Machine M.
  2. The **input string** for M.
- The UTM processes both and reproduces exactly the computation of M.



UTM and Modern Computers

- Programs = Encoded Turing Machines.
- Data = Input to the program.
- Universal Turing Machine = Abstract model of a general-purpose computer.

The Idea

Instead of building a new machine for each task, one universal machine can run *any program*.

UTM and Programming Languages

- The **Universal Turing Machine (UTM)** is an abstract model of a programmable computer.
- In modern terms:
  - The **encoded Turing Machine** corresponds to a *program* (e.g., in C++, Java).
  - The **input string** corresponds to the *program input*.
  - The **UTM** corresponds to the *computer/CPU* that executes any program.
- This is the foundation of the **stored-program architecture** in real computers.

Analogy with Real Computers

Universal Turing Machine:

- Machine description = Encoded Turing Machine.
- Input string = Tape data.
- Output = Simulated result.

Modern Computer:

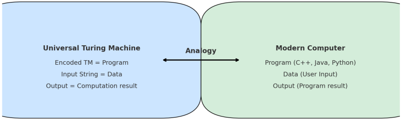
- Program = C++, Java, Python code.
- Data = Program input.
- Output = Program result.

Key Connection

A universal machine (abstract) $\longleftrightarrow$ a programmable computer (real).

Universal Turing Machine vs Modern Computers

- A **Universal Turing Machine** simulates any other Turing Machine.
- Programs are encoded as part of the input.
- This is analogous to how modern computers use:
  - A program (C++, Java, Python).
  - Data (user input).
  - Output (result of computation).



Shows that the concept of programmability was already present in Turing's model.

2.3 Gate-Based Universal Model of Classical Computation

Computation as transformations of bits using logic gates

Logic Gates as the Building Blocks of Computation

- A **bit** can be either 0 or 1.
- Computation can be modeled as applying transformations (rules) to bits.
- **Logic gates** are the basic building blocks:
  - AND
  - OR
  - NOT
- More complex circuits and computations can be built from these.

Fundamental Logic Gates

AND Gate: Output is 1 if both inputs are 1.

$$A \wedge B$$

OR Gate: Output is 1 if at least one input is 1.

$$A \vee B$$

NOT Gate: Inverts the input.

$$\neg A$$

Extensions

- **NAND:** Inverse of AND.
- **NOR:** Inverse of OR.
- **XOR (Exclusive OR):** Output is 1 if inputs are different.
- **XNOR (Exclusive NOR):** Output is 1 if inputs are the same.

Truth Tables: AND, OR, NOT

AND Gate

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

A	NOT A
0	1
1	0

Truth Tables: NAND, NOR

NAND Gate

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Truth Tables: XOR, XNOR

XOR Gate

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Example: Simple Circuit

Problem: Compute $f(A, B) = (A \vee B) \wedge \neg A$.

- Step 1: OR gate computes $A \vee B$.
- Step 2: NOT gate computes $\neg A$.
- Step 3: AND gate combines results: $(A \vee B) \wedge \neg A$.

This illustrates how combining simple gates produces more complex logical functions.

Evaluate

- $F(0, 0)$
- $F(1, 0)$
- $F(0, 1)$
- $F(1, 1)$

Universality of Logic Gates

- Any classical computation can be expressed with AND, OR, and NOT gates.
- Other universal sets exist (e.g., NAND or NOR alone).
- Logic gates are the foundation of:
    - Digital circuits.
    - Classical computer processors.
    - Modern computation theory.