# An Introduction to the Classical Theory of Computation

Dr. Hugo García Tecocoatzi

Instituto Tecnológico y de Estudios Superiores de Monterrey

August 14$^{\text{th}}$ 2025

# About me

- **Education:** Double Ph.D. in Theoretical Particle Physics
  - University of Genoa (Italy)
  - UNAM (Mexico)
- **Research Areas:**
  - **Nuclear Physics:** Heavy-ion double charge exchange (DCE), single/two-nucleon transfer, single charge exchange reactions.
  - Member (since 2017) of the **NUMEN Collaboration**
  - **Particle Physics:**
    - First theoretical prediction of excited $\Omega_b$, confirmed by LHCb, 2020.
    - Studies on hidden-charm pentaquarks (LHCb, 2019).
    - Predictions for heavy hybrid meson strong decay widths.
    - Effective models for light and heavy-flavor baryons.
- **New Skills:** Certified in data science, AI, Python, machine learning, and generative AI.

**Upcoming Talk** (join on **Friday**) via:

- Zoom link
- YouTube live stream

# Subjects in this block

2.0 Introduction.
   - Classical vs Quantum Computing.

2.1 Turing machines.

2.2 Universal Turing machines.

2.3 Gate-based Universal model of classical computation.

2.4 Complexity theory in a nutshell: classes P, NP, NP-Complete and NP-hard.

# Classical vs Quantum: Inputs and Usage

- **Classical computers** handle most everyday tasks:
  - Text processing, databases, web services.
  - Scientific simulations of modest size.
- **Quantum computers** require inputs encoded into **quantum states**:
  - Data must often be prepared from classical sources.
  - Results are measured back into classical bits.
- **Hybrid approach in practice:**
  - Classical PC prepares data and controls the quantum hardware.
  - Quantum processor executes the quantum algorithm.
  - Classical PC processes and stores the results.

# When Quantum is Faster (and When It's Not)

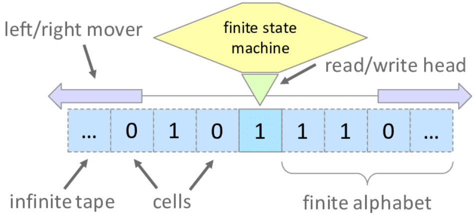
**Quantum advantage:**

- **Exponential speedups:** Shor's algorithm for factoring large integers.
- **Quadratic speedups:** Grover's algorithm for unstructured search.
- Simulating quantum systems in physics and chemistry.

**Classical advantage:**

- Most arithmetic and logical operations.
- Small or medium-size problems where quantum setup overhead is large.
- Data-heavy tasks without a known quantum speedup.

A foundational model of computation introduced by Alan Turing (1936)



# Why do we need Turing Machines?

- Before modern computers existed, mathematicians wanted to formally define what it means to compute.
- Alan Turing proposed a simple, abstract machine to capture the essence of computation.
- Still relevant today for:
  - Defining what problems are solvable.
  - Proving limits of computation.

# General Description of a Turing Machine

- A **theoretical model of computation** that forms the foundation for understanding algorithms and the limits of computation.
- Consists of:
  - An **infinite tape** divided into cells.
  - A **read/write head** that moves along the tape.
  - A **finite control unit** with a set of states and rules.
- Operates by reading the current symbol, consulting transition rules, writing a new symbol, moving the head, and changing state.
- Despite its simplicity, can simulate **any computer algorithm**.
- Powerful tool for exploring **computability** and **complexity**.

# Basic Components

A Turing Machine consists of:

1. **Infinite tape** divided into cells, each holding one symbol.
2. **Head** that can:
   - Read the current symbol.
   - Write a symbol.
   - Move one cell left or right.
3. **Finite control** with a set of states, including:
   - Start state.
   - Accept/reject states (halting conditions).

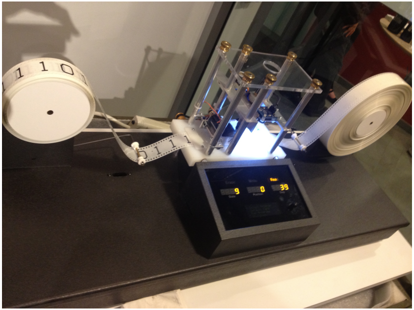


Illustration of a Turing Machine showing the tape, head, and control unit.

# Formal Definition

A Turing Machine $M$ is a 7-tuple:

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

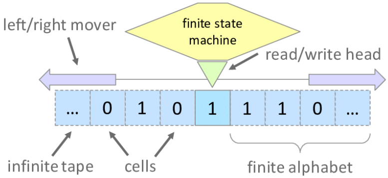
Where:

- $Q$: finite set of states.
- $\Sigma$: input alphabet (no blank symbol).
- $\Gamma$: tape alphabet ($\Sigma \subset \Gamma$, includes blank symbol).
- $\delta$: transition function:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

- $q_0$: start state.
- $q_{\text{accept}}, q_{\text{reject}} \in Q$: halting states.

1. Machine starts in $q_0$ with input on the tape.
2. Reads the symbol under the head.
3. Transition function:
   - Changes the state.
   - Writes a symbol on the tape.
   - Moves the head left (L) or right (R).
4. Repeats until reaching $q_{\text{accept}}$ or $q_{\text{reject}}$.




Example of a Turing Machine execution.

# Example: Unary Increment

**Task:** Given an input in unary notation (e.g., 111 for the number 3), produce the number incremented by 1 (1111 for 4).

**Idea:**

1. Start at the leftmost symbol.
2. Move right until a blank symbol (␣) is found.
3. Write 1 in the blank.
4. Halt.

**Tape alphabet:** { 1, ␣}
**States:** { $q_0$ (start), $q_{\text{halt}}$ (halt) }

# Step-by-Step Execution

**Initial tape:** . . . [␣] [1] [1] [1] [␣] [␣] . . .
Head starts at the first cell in $q_0$.

| Step | Action |
|---|---|
| 1 | Read 1, move right. |
| 2 | Read 1, move right. |
| 3 | Read 1, move right. |
| 4 | Read ␣, write 1. |
| 5 | Enter $q_{\text{halt}}$ and stop. |

**Final tape:** . . . [␣] [1] [1] [1] [1] [␣] . . .
**Result:** Unary number incremented by 1.

# Example: Turing Machine for $L = \{0^n1^n\}$

**Goal:** Accept strings with the same number of 0's followed by the same number of 1's.

**Algorithm:**

1. Start at the leftmost cell.
2. Find the first unmarked `0`, replace it with `X`.
3. Move right until the first unmarked `1`, replace it with `Y`.
4. Move left to the start of the tape.
5. Repeat steps 2–4 until:
   - All 0's are marked `X` and all 1's are marked `Y` $\rightarrow$ **Accept**.
   - Mismatch found (wrong order or unequal counts) $\rightarrow$ **Reject**.

**Tape alphabet:** $\{0, 1, X, Y, \square\}$

# Execution Example: Input 0011

**Initial tape:** [0][0][1][1][␣]...

| Step | Tape | Action |
|---|---|---|
| 1 | [0][0][1][1][␣] | Mark first 0 → X, move right |
| 2 | [X][0][1][1][␣] | Mark next 0 → X, move right |
| 3 | [X][X][1][1][␣] | Mark first 1 → Y, move left |
| 4 | [X][X][Y][1][␣] | Move left to find unmarked 0 (none left) |
| 5 | [X][X][Y][Y][␣] | All matched: **Accept** |

**Final tape:** [X][X][Y][Y][␣]...
**Result:** String belongs to $L$.

# TM for $L = \{0^n1^n \mid n \geq 1\}$ — Formal Definition

A Turing Machine $M$ deciding $L$:

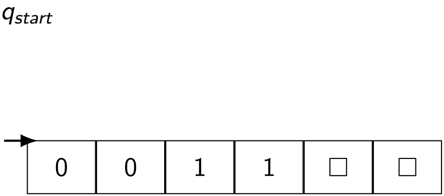
$$M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$$

- $Q = \{q_{\text{start}}, q_0, q_1, q_2, q_{\text{accept}}, q_{\text{reject}}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, X, Y, \square\}$ $\quad$ ($\square$ is blank)
- $q_{\text{start}}$ is the start state
- $q_{\text{accept}}, q_{\text{reject}}$ are halting states

**Intuition:** Pair each leftmost unmarked 0 with the rightmost next unmarked 1 by marking them $X$ and $Y$; repeat from the left. Reject on order/count mismatches.

# Tape Trace (Input 0011)

**Legend:** Head $\rightarrow$ cell with arrow; states shown above the tape.




*Mark first 0 $\rightarrow$ X and move right*

# Tape Trace (Input 0011)

**Legend:** Head $\rightarrow$ cell with arrow; states shown above the tape.

*$q_1$ (scan to first unmarked 1)*

| | | | | | |
|---|---|---|---|---|---|
| X | 0 | 1 | 1 | □ | □ |

*Skip 0/X/Y until a 1 is found*

# Tape Trace (Input 0011)

**Legend:** Head $\rightarrow$ cell with arrow; states shown above the tape.

*$q_2$ (return left after matching)*

| | | | | | |
|---|---|---|---|---|---|
| X | 0 | Y | 1 | □ | □ |

*Matched 1 $\rightarrow$ $Y$, move left to the left end*

# Tape Trace (Input 0011)

**Legend:** Head → cell with arrow; states shown above the tape.

*$q_0$ (seek next unmarked 0)*

| | | | | | |
|---|---|---|---|---|---|
| X | 0 | Y | 1 | □ | □ |

*From left blank, step right and resume pairing*

# Tape Trace (Input 0011)

**Legend:** Head $\rightarrow$ cell with arrow; states shown above the tape.

$q_2$ *(after matching the second 1)*

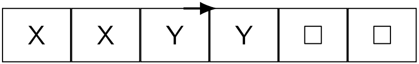
| | | | | | |
|---|---|---|---|---|---|
| X | X | Y | Y | $\square$ | $\square$ |

*All 0's have been paired and marked*

# Tape Trace (Input 0011)

**Legend:** Head $\rightarrow$ cell with arrow; states shown above the tape.

$q_{accept}$




*No unmarked 0/1 remain* $\Rightarrow$ *Accept*

# Exercise: Design a TM for $L = 0\,1^n 0$

**Language:**

$$L = \{\, 0\,1^n 0 \mid n \geq 1 \,\}$$

That is: strings that start with a single 0, followed by one or more 1's, and end with a single 0.

**Your task:**

1. **Step 1:** Write the high-level algorithm for deciding $L$.
   - Check that the first symbol is 0.
   - Verify that at least one 1 follows.
   - Ensure the final symbol is 0 and nothing else after it.
2. **Step 2:** Define the 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$.
3. **Step 3:** Draw the state diagram and label the transitions.
4. **Step 4:** Test with example inputs:

**Hint:**

- Scan the first symbol and confirm it's 0.
- Loop over consecutive 1's.
- Accept if the next symbol is a single 0 and the tape ends after it.

# Why They Matter

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

# Turing Machines: Key Takeaways

- Abstract, simple, yet powerful computational model.
- Consists of an infinite tape, a head, and finite control.
- Formalizes the concept of an *algorithm*.
- Foundation for understanding computability and complexity.