

CMSC 28100

Introduction to Complexity Theory

Autumn 2025

Instructor: William Hoza



Homework reminder

- Exercises 1-3 are due **today at 11:59pm**
- If you joined the course late and you need an extension, send me an email

Which problems
can be solved
through computation?

Deciding a language

- Let M be a Turing machine and let $Y \subseteq \{0, 1\}^*$
- We say that M **decides** Y if
 - M accepts every $w \in Y$, and
 - M rejects every $w \in \{0, 1\}^* \setminus Y$

Decidable and undecidable

- Let $Y \subseteq \{0, 1\}^*$
- We say that Y is **decidable** if there exists a Turing machine M that decides Y
- Otherwise, we say that Y is **undecidable**

Which problems
can be solved
through computation?

Which languages are decidable?

Examples

- $\text{PALINDROMES} = \{w \in \{0, 1\}^* : w \text{ is the same forward and backward}\}$
- $\text{PARITY} = \{w \in \{0, 1\}^* : w \text{ has an odd number of ones}\}$
- $Y = \{0^K \langle K \rangle : K \text{ is a positive integer}\}$

Out of those three languages, how many are decidable?

A: Zero

B: One

C: Two

D: Three

Respond at [PollEv.com/whoza](https://pollev.com/whoza) or text "whoza" to 22333

Is **every** language decidable?

Undecidability

Theorem: There exists an **undecidable** language.

- To prove this theorem, we need to rule out all possible Turing machines!
- How can we possibly do this?

The liar paradox

Are you selecting option B as your answer to this question?

A: Yes B: No

C: Yes D: Yes

Respond at [PollEv.com/whoza](https://pollev.com/whoza) or text "whoza" to 22333

Code as data

- Plan: We will construct a language Y such that trying to decide Y creates a liar paradox
- Key idea: A Turing machine M can be encoded as a binary string $\langle M \rangle$
 - “Code as data”
 - Specific encoding choice doesn’t matter for now

Turing machines analyzing Turing machines

- If M is a Turing machine...
- Then $\langle M \rangle$ can be the **input** for **another** Turing machine!
- Compilers, syntax highlighting, linters...

Self-rejecting Turing machines

- Let M be a TM
- What if we run M on $\langle M \rangle$? Strange, but legal
- Three possibilities:
 - M accepts $\langle M \rangle$
 - M rejects $\langle M \rangle$
 - M loops on $\langle M \rangle$
- **Definition:** We say that a Turing machine M is self-rejecting if M rejects $\langle M \rangle$



Self-rejecting Turing machines



- Let **SELF-REJECTORS** = $\{\langle M \rangle : M \text{ is a self-rejecting Turing machine}\}$


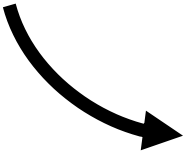
Theorem: SELF-REJECTORS is **undecidable**

- **Proof:** Let M be any Turing machine
- If M **rejects** $\langle M \rangle$, then $\langle M \rangle \in \text{SELF-REJECTORS}$
- If M **doesn't reject** $\langle M \rangle$, then $\langle M \rangle \notin \text{SELF-REJECTORS}$
- Either way, M **does not decide** SELF-REJECTORS!

Visualizing the proof: “Diagonalization”

What happens
when we run this
Turing machine...

...on this input?



| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ | ... |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----|
| M_1 | ✓ | ✓ | ✗ | ∞ | ... |
| M_2 | ✗ | ✗ | ✓ | ✓ | ... |
| M_3 | ✓ | ✓ | ✗ | ✗ | ... |
| M_4 | ∞ | ✓ | ✗ | ∞ | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

✓ = Accept

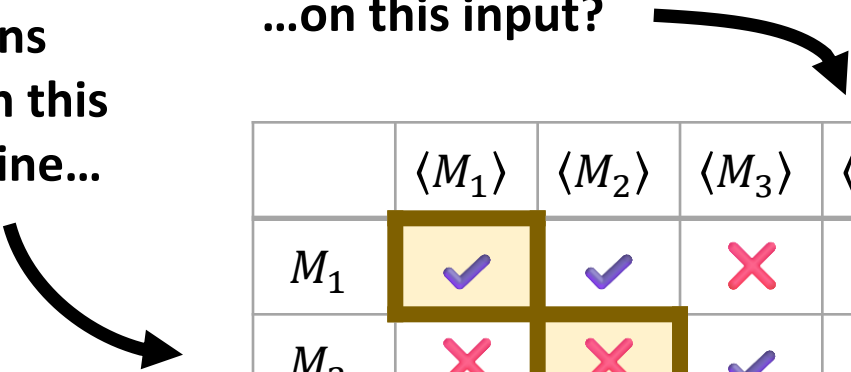
✗ = Reject

∞ = Loop

Visualizing the proof: “Diagonalization”

What happens
when we run this
Turing machine...

...on this input?



| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ | ... |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----|
| M_1 | ✓ | ✓ | ✗ | ∞ | ... |
| M_2 | ✗ | ✗ | ✓ | ✓ | ... |
| M_3 | ✓ | ✓ | ✗ | ✗ | ... |
| M_4 | ∞ | ✓ | ✗ | ∞ | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

✓ = Accept

✗ = Reject

∞ = Loop

Undecidable language:

✗ ✓ ✓ ✗ ...

Interpreting the theorem

- We proved that there does not exist a **Turing machine** that decides SELF-REJECTORS
- **OBJECTION:** “Yeah, but I don’t particularly care about Turing machines. Is there some **other type of algorithm** that decides SELF-REJECTORS?”
- **RESPONSE:** The Church-Turing Thesis

The Church-Turing Thesis

- Let $Y \subseteq \{0, 1\}^*$

Church-Turing Thesis:

There exists an “algorithm” / “procedure” for figuring out whether a given string is in Y **if and only if** there exists a Turing machine that decides Y .

← Intuitive notion

← Mathematically precise notion

The Church-Turing Thesis

- The Church-Turing thesis says:
 - The Turing machine model is a “correct” way of modeling arbitrary computation
 - The informal concept of an “algorithm” is successfully captured by the rigorous definition of a Turing machine
- Consequence: It is really, truly impossible to design an algorithm that decides SELF-REJECTORS or any other undecidable language!

Are Turing machines powerful enough?



- **OBJECTION:** “To encompass all possible algorithms, we should add various bells and whistles to the Turing machine model.”
- Example: **Left-Right-Stationary Turing Machine**: Like an ordinary Turing machine, except it has a transition function $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, S\}$
- S means the head **does not move** in this step
- (Exercise: Rigorously define NEXT, accepting, rejecting, etc.)

Left-right-stationary Turing machines



- The model is **still realistic**, even though we added an extra feature
- Is it a counterexample to the Church-Turing thesis?
- No!
- Let's prove that the left-right-stationary Turing machine model is **equivalent** to the original Turing machine model

Left-right-stationary Turing machines



- Let Y be a language

Theorem: There exists a left-right-stationary TM that decides Y
if and only if there exists a TM that decides Y

- **Proof:** (3 slides) The “ \Leftarrow ” direction is trivial

Left-right-stationary Turing machines



- Idea of the proof of “ \Rightarrow ” direction: Simulate S by doing L followed by R
- Details: Let $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \sqcup, \delta)$ be a left-right-stationary TM that decides Y
- New TM: $M' = (Q', q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \sqcup, \delta')$
- New set of states: $Q' = Q \cup \{\underline{q} : q \in Q\}$, i.e., two disjoint copies of Q

Left-right-stationary Turing machines



- New transition function $\delta': Q' \times \Sigma \rightarrow Q' \times \Sigma \times \{L, R\}$ given by:
 - If $\delta(q, b) = (q', b', L)$, then $\delta'(q, b) = \delta(q, b)$
 - If $\delta(q, b) = (q', b', R)$, then $\delta'(q, b) = \delta(q, b)$
 - If $\delta(q, b) = (q', b', S)$, then $\delta'(q, b) = (\underline{q'}, b', L)$
 - For every q and b , we let $\delta'(\underline{q}, b) = (q, b, R)$
- Exercise: Rigorously prove that M' decides Y

The Church-Turing Thesis

- Let $Y \subseteq \{0, 1\}^*$

Church-Turing Thesis:

There exists an “algorithm” / “procedure” for figuring out whether a given string is in Y **if and only if** there exists a Turing machine that decides Y .

← Intuitive notion

← Mathematically precise notion

Multi-tape Turing

- Another TM variant: “ k -tape
- Transition function:

$$\delta: Q \times \Sigma^k \rightarrow Q \times \Sigma^k \times \{L, R, S\}^k$$
- (Exercise: Rigorously define acceptance, rejection, etc.)

