CMSC 28100

Introduction to Complexity Theory

Spring 2025
Instructor: William Hoza

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 $w \in \{0,1\}^* \setminus Y$
- This is a mathematical model of what it means to “solve a problem”

Invalid inputs

- **Informal problem statement:** "Given a graph G, determine whether it is connected"
- **The same problem, formulated as a language:**

$$\text{CONNECTED} = \{\langle G \rangle : G \text{ is a connected graph}\}$$

What if we are given $w \in \{0,1\}^*$ that is not the encoding of any graph?

A: This situation cannot occur

B: It doesn't matter what we do

C: We can accept or reject, but we must not loop

D: We must reject

Respond at PollEv.com/whoza or text "whoza" to 22333

ph, we should accept

graph, we should reject

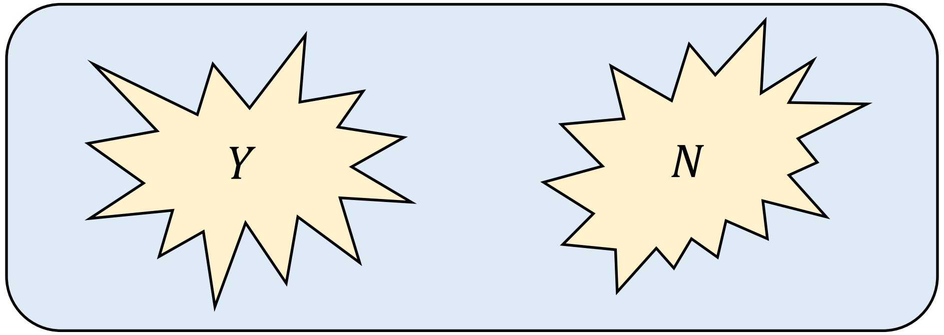
Invalid inputs

- There can exist "invalid inputs" $w \in \{0, 1\}^*$ that do not encode graphs
  - For example, suppose we are using adjacency matrices to encode graphs
  - Then $|\langle G \rangle|$ is a perfect square for every graph G
  - Therefore, 10101 is not the encoding of any graph
- Technically, $10101 \notin \text{CONNECTED} = \{\langle G \rangle : G \text{ is a connected graph}\}$
- To decide CONNECTED, a Turing machine would have to reject 10101

Checking for validity

- **OBJECTION:** "But the informal problem statement didn't say anything about rejecting invalid inputs!"
  - "Given a graph G, determine whether it is connected"
- **RESPONSE 1:** It is not hard to check whether a given string w is the encoding of a graph
- Therefore, if we are trying to understand how hard/easy the problem is, we don't need to worry about invalid inputs

Promise problems



$\{0, 1\}^*$

- **RESPONSE 2:** There are more sophisticated ways of modeling "problems"
- Definition: A promise problem is a pair $\Pi = (Y, N)$, where Y and N are disjoint subsets of $\{0, 1\}^*$
  - E.g., $Y = \{\langle G \rangle : G \text{ is a connected graph}\}$ and $N = \{\langle G \rangle : G \text{ is a disconnected graph}\}$
- We say that a Turing machine M solves Π if it accepts every $w \in Y$ and it rejects every $w \in N$

Ignoring invalid inputs

- In this course, for simplicity's sake and for historical reasons, we will focus on languages rather than promise problems

- However, for simplicity's sake, we will mostly ignore the issue of invalid inputs

Summary

- “Deciding a language” is not a perfect mathematical model of “solving a problem”…
- But it is a pretty good model

Decidable and undecidable

- Let Y be a language
- 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 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 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"
  - We'll discuss this in more detail later

Turing machines analyzing Turing machines

- After encoding a Turing machine M as a binary string $\langle M \rangle$…
- We can use $\langle M \rangle$ as the input for another Turing machine!
- Compilers, syntax highlighting, linters…

Self-rejecting Turing machines

- Let M be a TM
- A strange-but-legal thing we can do: Run M on $\langle M \rangle$
- 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 $\text{SELF-REJECTORS} = \{\langle M \rangle : M \text{ is a self-rejecting Turing machine}\}$

Theorem: SELF-REJECTORS is undecidable

Proof: Let M be any TM. We'll show that M does not decide SELF-REJECTORS

- If M rejects $\langle M \rangle$, then $\langle M \rangle \in \text{SELF-REJECTORS}$, so M ought to accept $\langle M \rangle$ ❌
- If M doesn't reject $\langle M \rangle$, then $\langle M \rangle \notin \text{SELF-REJECTORS}$, so M ought to reject $\langle M \rangle$ ❌
- In either case, M does the wrong thing!

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 that the Turing machine model is a "correct" way of modeling arbitrary computation
- The thesis says that 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 $\mathrm{SELF\text{-}REJECTORS}$ or any other undecidable language!

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

Are Turing machines too powerful? ∞

- **OBJECTION:** “The Turing machine’s infinite tape is unrealistic!”
- **RESPONSE 1:** If M decides some language, then on any particular input w, the machine M only uses a finite amount of space
- **RESPONSE 2:** We are studying idealized computation
- **RESPONSE 3:** We’re especially focused on impossibility results, so it’s better to err on the side of making the model extra powerful

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 $\delta: Q \times \Sigma \to Q \times \Sigma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}$
- S means the head does not move in this step
- (Exercise: Rigorously define NEXT, accepting, rejecting, etc.)

Left-right-stationary Turing machines

- The left-right-stationary Turing machine 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

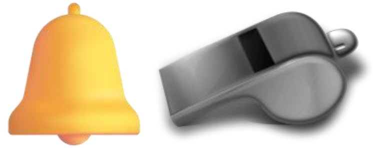
> **Theorem:** There exists a left-right-stationary TM that decides Y
>
> if and only if there exists a TM that decides Y

- **Proof:** (3 slides) The "$\Leftarrow$" direction is trivial

Left-right-stationary Turing machines

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

Left-right-stationary Turing machines



- New transition function $\delta': Q' \times \Sigma \to Q' \times \Sigma \times \{\mathrm{L}, \mathrm{R}\}$ given by:
  - If $\delta(q, b) = (q', b', \mathrm{L})$, then $\delta'(q, b) = \delta(q, b)$
  - If $\delta(q, b) = (q', b', \mathrm{R})$, then $\delta'(q, b) = \delta(q, b)$
  - If $\delta(q, b) = (q', b', \mathrm{S})$, then $\delta'(q, b) = \left(\underline{q'}, b', \mathrm{L}\right)$
  - For every q and b, we let $\delta'\left(\underline{q}, b\right) = (q, b, \mathrm{R})$
- 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