CMSC 28100

Introduction to Complexity Theory

Spring 2024
Instructor: William Hoza

The nature of this course

- In this course, we will study
  - The mathematical and philosophical foundations of computer science
  - The ultimate limits of computation
- This course will give you powerful conceptual tools for reasoning about computation
- There will be very little programming
- Homework and exams will be primarily proof-based

Who this course is designed for

- CS students, math students, and anyone who is curious
- Prerequisites:
  - Experience with mathematical proofs
  - CMSC 27200 or CMSC 27230 or CMSC 37000, or MATH 15900 or MATH 15910 or MATH 16300 or MATH 16310 or MATH 19900 or MATH 25500

Who this course is designed for

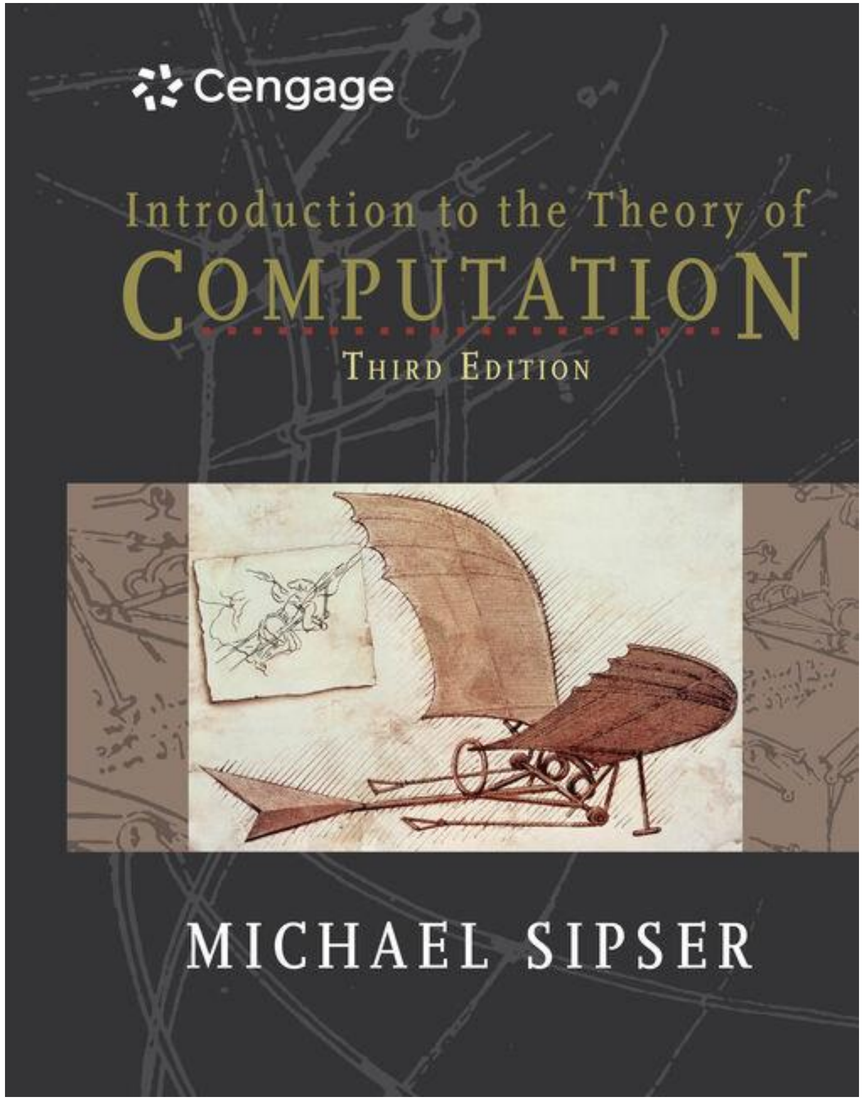
- I would like every CS student to take this course
- It's okay if you don't consider yourself "theory-oriented." You belong here
- I consider it my job to give you resources so you can learn and succeed
- I also consider it my job to persuade you that complexity theory is important, interesting, enlightening, fun, cool, and generally worthy of your attention

Class participation

- Please ask questions!
  - “How do we know ______?”
  - “Can you remind me what ______ means?”
  - “I don’t get it. Can you explain that again?”
- We are not in a hurry

Textbook

- Classic
- Popular
- High-quality
- Not free ☹



My office hours

- Mondays, 10:30am – 12:30pm, JCL 205
  - Exception: No office hours today (3/18)
- Stop by! This is the best time to discuss:
  - Questions about the course material or the homework
  - Concerns, complaints, or suggestions about how to improve the course
  - Complexity theory topics that you’re simply curious about

Course staff

- Zelin Lv (TA)
  - Office hours: Fridays, 1pm – 2pm, JCL 205
- Rohan Soni (TA)
  - Office hours: Thursdays, 2:30pm – 3:30pm, JCL 205
- Nico Marin Gamboa (Grader)
- Loren Troan (Grader)

Technology

- Course webpage: https://williamhoza.com/teaching/spring2024-intro-to-complexity
  - Course policies; slides
- Canvas: https://canvas.uchicago.edu/courses/55826
  - Problem sets; practice exams; official solutions
- Ed: https://edstem.org/us/courses/56687/
  - Discussions; announcements
- Gradescope: https://www.gradescope.com/courses/748307
  - Submitting homework solutions; grades and feedback

Assessment

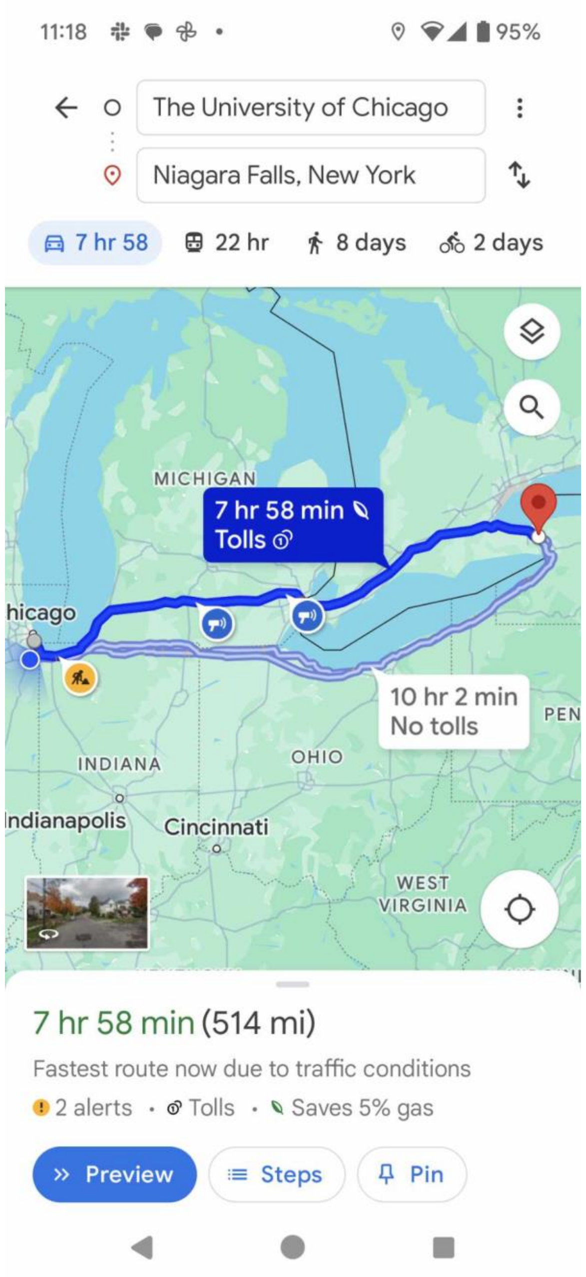
- There will be 6 or 7 problem sets throughout the quarter
- The first problem set is due **Tuesday, March 26**
- There will be a midterm exam and a final exam

The central question of this course:

Which problems can be solved through computation?

Examples

- The following problems can be solved through computation:
  - Addition
  - Multiplication
  - Shortest path



- Are there any problems that cannot be solved through computation?

Impossibility proofs

- To persuasively argue that certain problems cannot be solved through computation, we will take a mathematical approach
- We will formulate precise mathematical models
  - “Problem”
  - “Computation”
  - “Solve”
- Then we will write rigorous mathematical proofs of impossibility

Which problems

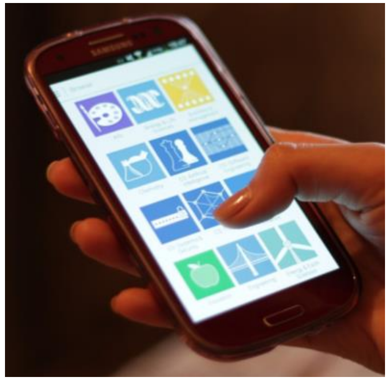
can be solved

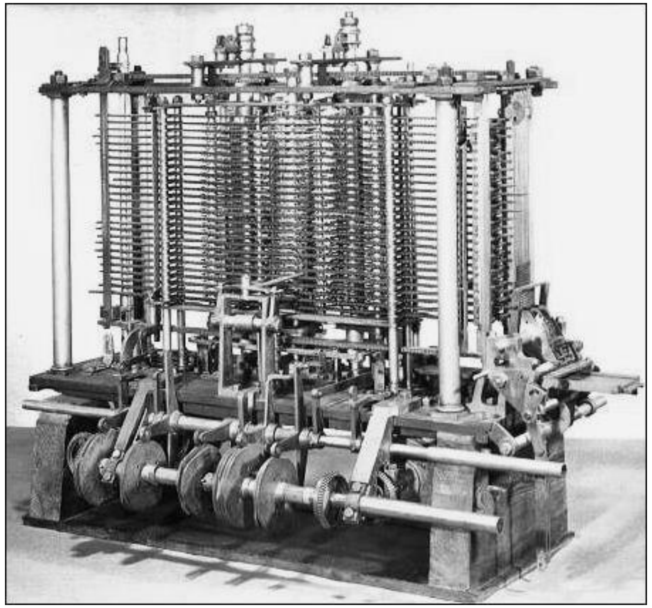
through computation?

Computation

- You might think of "computers" as modern technology, but computation is ancient

- Computation can be performed by
  - A human being with paper and a pencil
  - A smartphone
  - A steam-powered machine



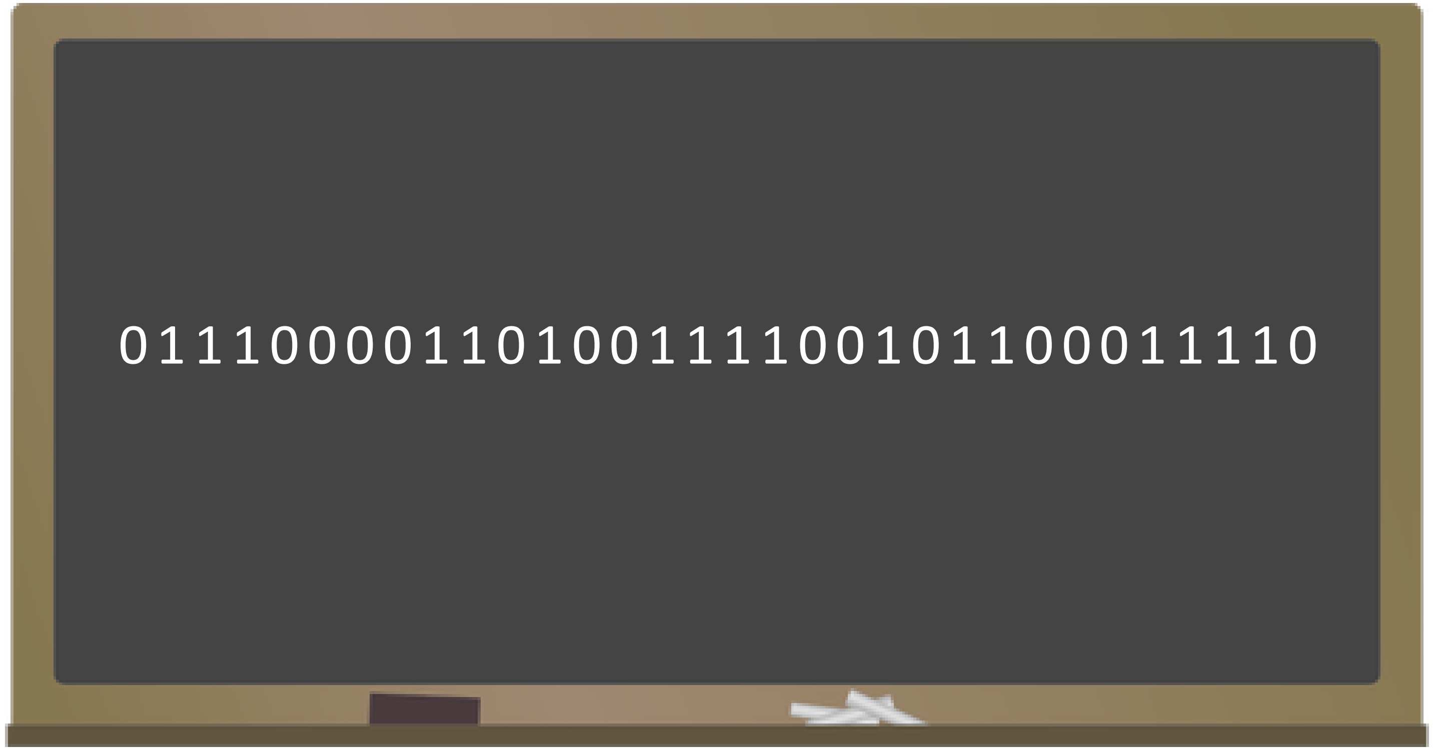


- We want a mathematical model that describes all of these and transcends any one technology

Human computation vs. technological

- Smartphones and laptops merely automate the process of computation
- They can compute faster and more reliably than a human being, but what they do is essentially the same as what we do
- Consequence: We do not need to understand electronics to understand computation 🙂
- Computation is a familiar, everyday, human act
- "Mathematical anthropology"

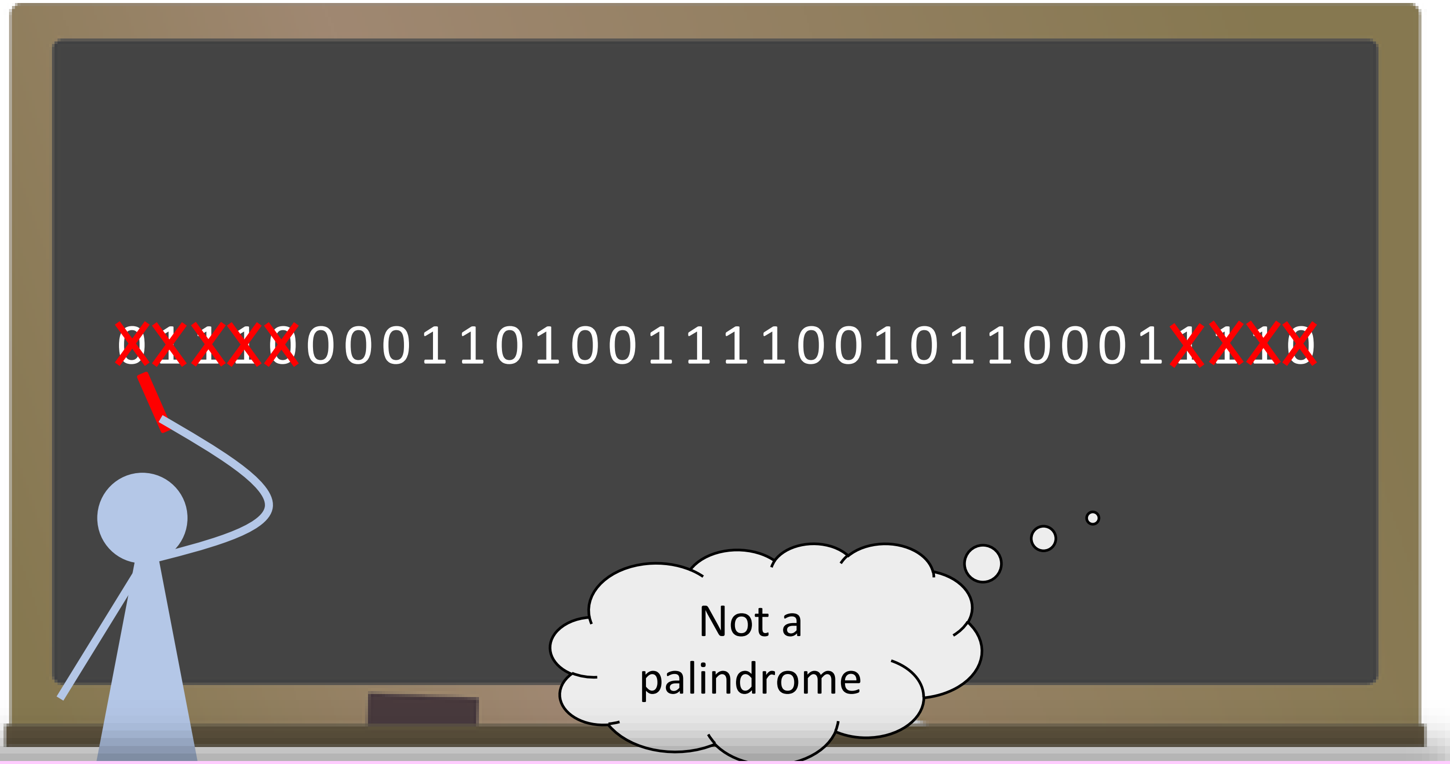
Ex: Palindromes



- Suppose a long string of bits is written on a blackboard
- Your job: Figure out whether the string is a “palindrome,” i.e., whether it is the same forwards and backwards
- What’s your approach?

Ex: Palindromes

- Idea: Compare and cross off the first and last symbols
- Repeat until we find a mismatch or everything is crossed off



At the end of the process, where might we be standing?

A: Anywhere

B: Right half only

C: Even-numbered positions only

D: Odd-numbered positions only

Respond at PollEv.com/whoza or text “whoza” to 22333

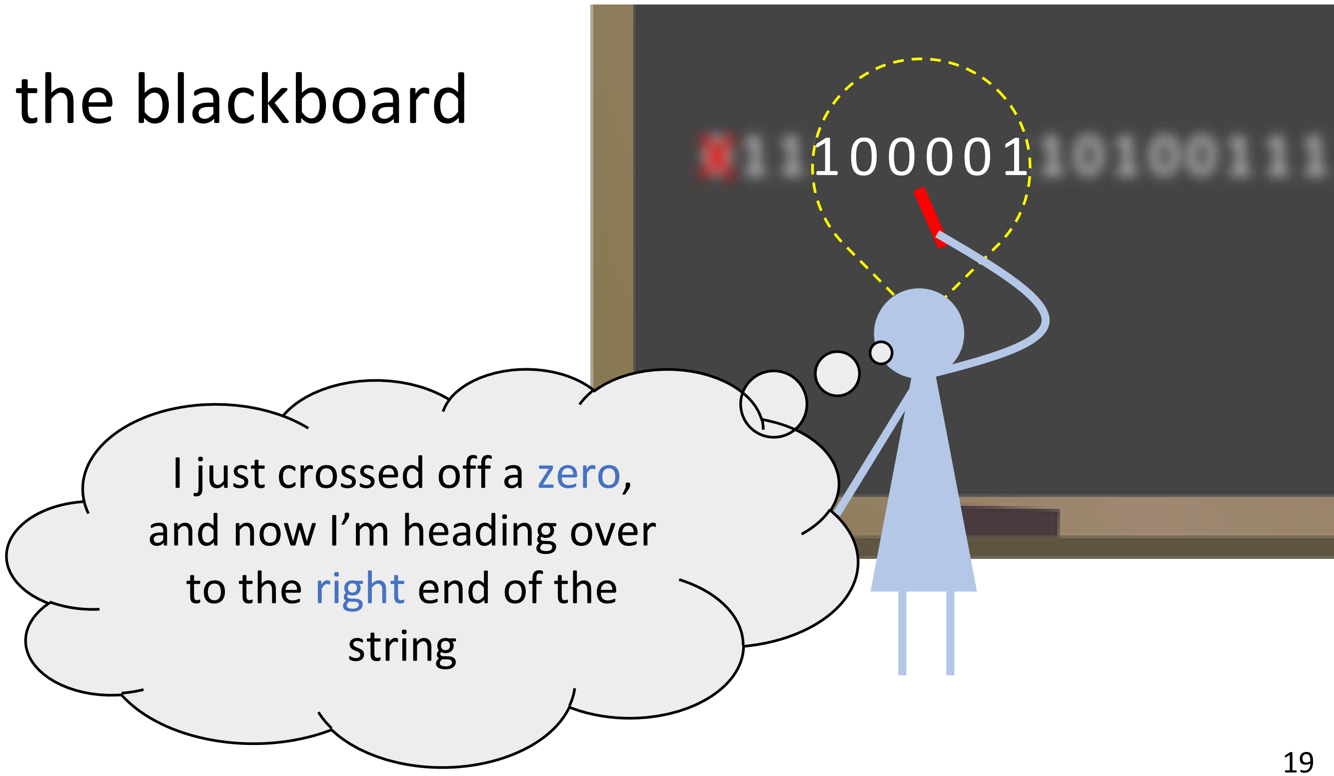
Local decisions

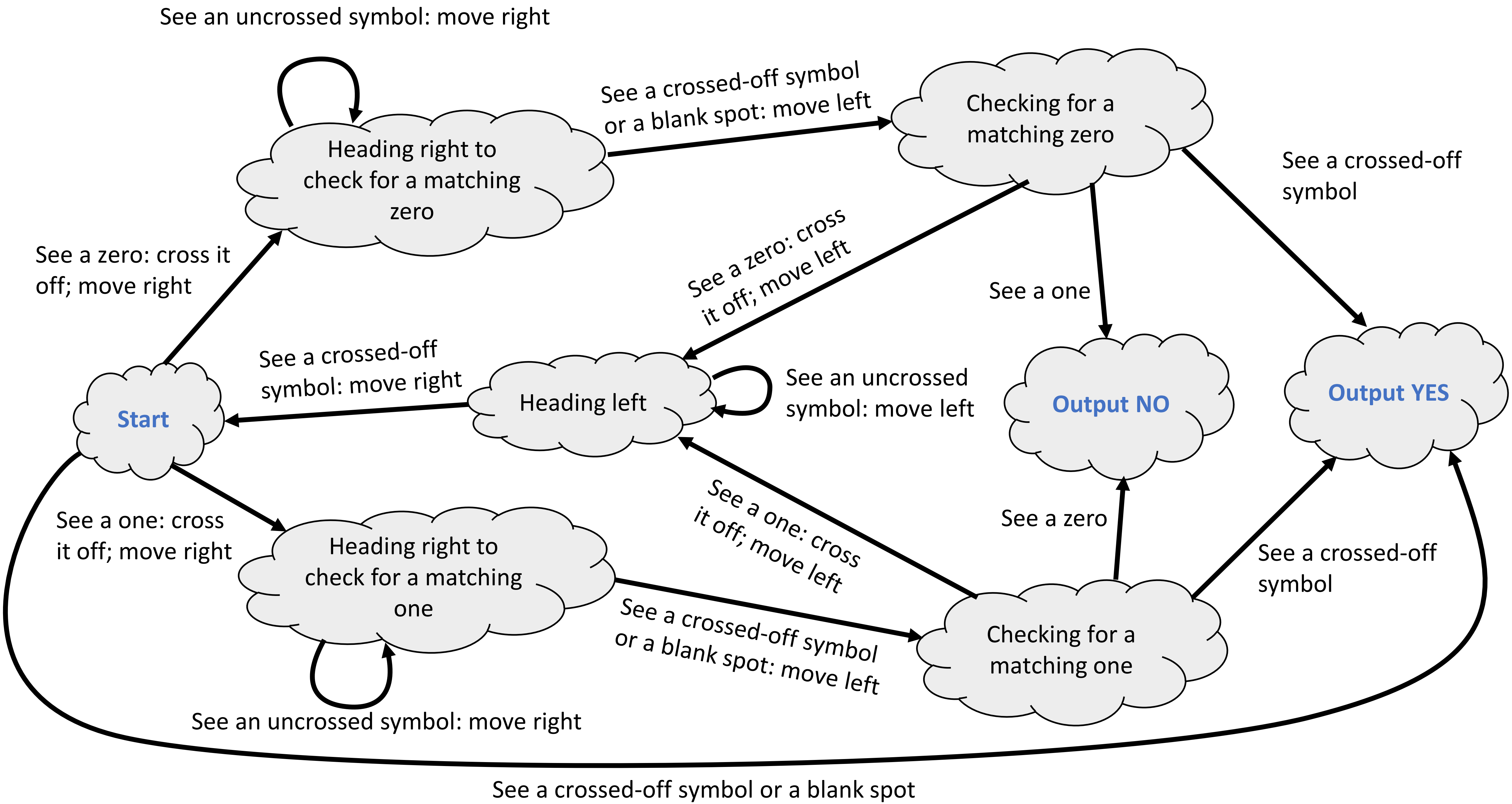
- In each step, what information do we use to decide what to do next?

1. We keep track of some information ("state") in our mind
2. We look at the local contents of the blackboard

   (one symbol is sufficient)

- We can describe the algorithm in excruciating detail using a "state diagram" (next slide)

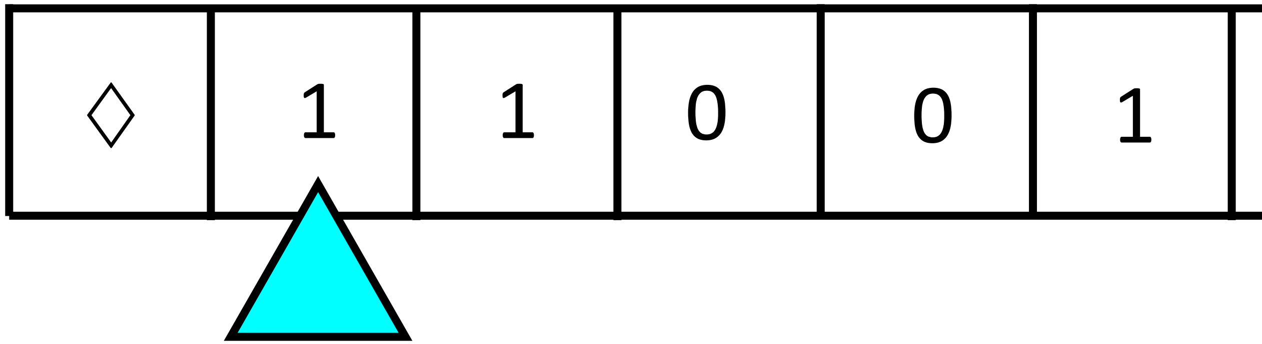



See an uncrossed symbol: move right
Heading right to check for a matching zero
See a crossed-off symbol or a blank spot: move left
Checking for a matching zero
See a crossed-off symbol
See a zero: cross it off; move right
See a zero: cross it off; move left
See a one
See a crossed-off symbol: move right
Heading left
See an uncrossed symbol: move left
Output NO
Output YES
Start
See a one: cross it off; move left
See a zero
See a one: cross it off; move right
See a crossed-off symbol
Heading right to check for a matching one
See a crossed-off symbol or a blank spot: move left
Checking for a matching one
See an uncrossed symbol: move right
See a crossed-off symbol or a blank spot

The Turing machine model

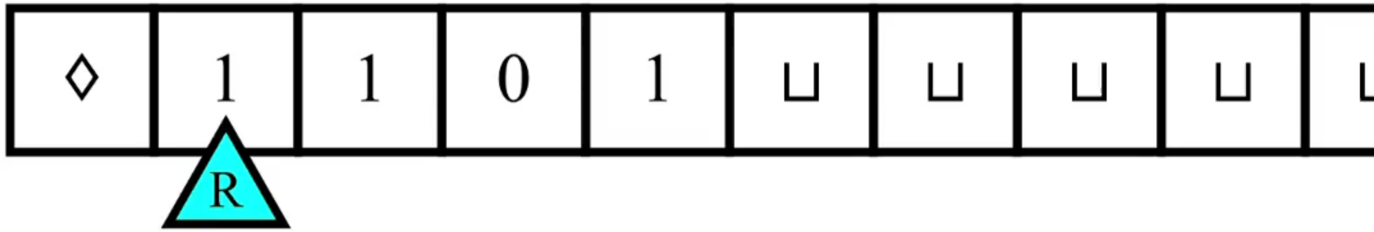
- Turing machines: A mathematical model of human computation
- Basic idea: a Turing machine is any algorithm that can be described by a state diagram similar to what we just saw

Turing machines



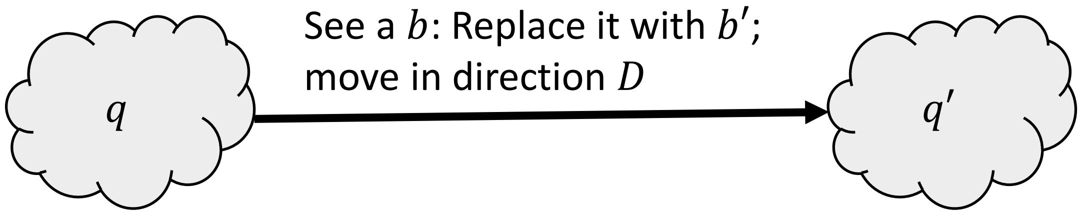
- We imagine a one-dimensional “tape” that extends infinitely to the right
- The tape is divided into “cells.” Each cell has one symbol written in it
- There is a “head” pointing at one cell of the tape
- The machine can be in one of finitely many internal “states”

Turing machines



- In each step, the machine decides
  - What to write
  - Which direction to move the head (left or right)
  - The new state
- The decision is based only on the current state and the observed symbol

Transition function



- Mathematically, the update rule is specified by a transition function

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

- Here Q is the set of states and Γ is the set of symbols
- $\delta(q, b) = (q', b', D)$ means: "If we are in state q and we read the symbol b, then our new state will be q', we will write b' (replacing b), and the head will move in the direction D (L for left or R for right)"

The input to a Turing machine

- A Turing machine represents an algorithm
- The input to a Turing machine is always a finite string of symbols

Symbols and alphabets

- An “alphabet” Σ is any nonempty, finite set of “symbols”
  - $\Sigma = \{0, 1\}$
  - $\Sigma = \{0, 1, \cancel{0}, \cancel{1}\}$
  - $\Sigma = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots, \mathrm{Z}\}$
  - $\Sigma = \{$😍, ⚾, 🐍, ⏰, 🍕$\}$

Strings

If $|\Sigma| = m$, then what is $|\Sigma^0|$?

A: $|\Sigma^0| = 0$

B: $|\Sigma^0| = m$

C: $|\Sigma^0| = 1$

D: $|\Sigma^0|$ is not well-defined

Respond at PollEv.com/whoza or text “whoza” to 22333

- Let Σ be an alphabet
- A string over Σ is a finite se
- The length of a string x is the number of symbols, denoted $|x|$
- If n is a nonnegative integer, then Σ^n is the set of length-n strings over Σ
- Example: If $\Sigma = \{0, 1\}$, then

$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

The empty string

- If Σ is any alphabet, then $|\Sigma^0| = 1$
- There is one string of length zero, called the empty string
- We use ϵ to denote the empty string
  - Denoted `""` in popular programming languages
- $\Sigma^0 = \{\epsilon\}$

Arbitrary-length strings

- Let Σ be an alphabet
- We define Σ^* to be the set of strings over Σ of any finite length:

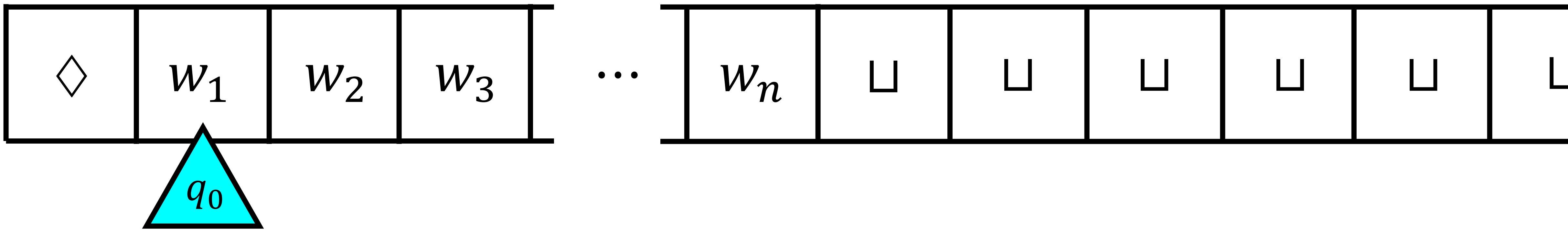
$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

- Example: If $\Sigma = \{0, 1\}$, then

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$$

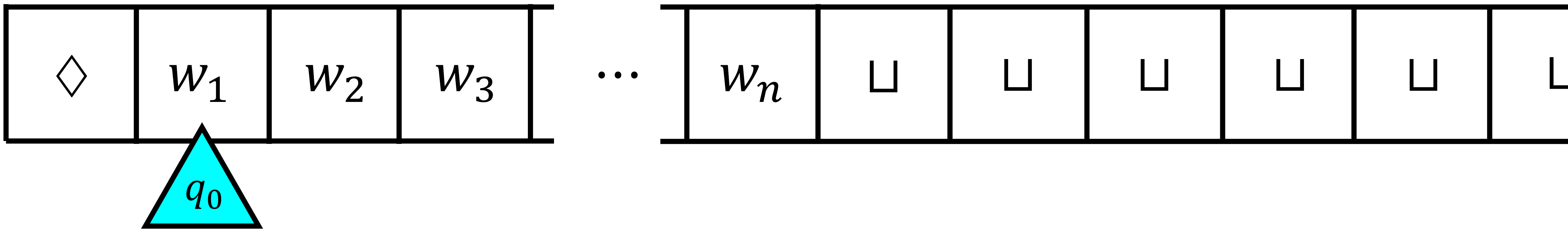
Turing machine initialization

- The tape initially contains a special "start symbol" $\Diamond$, followed by the input string w (one symbol per cell)
- All remaining cells initially contain a special "blank symbol" $\sqcup$

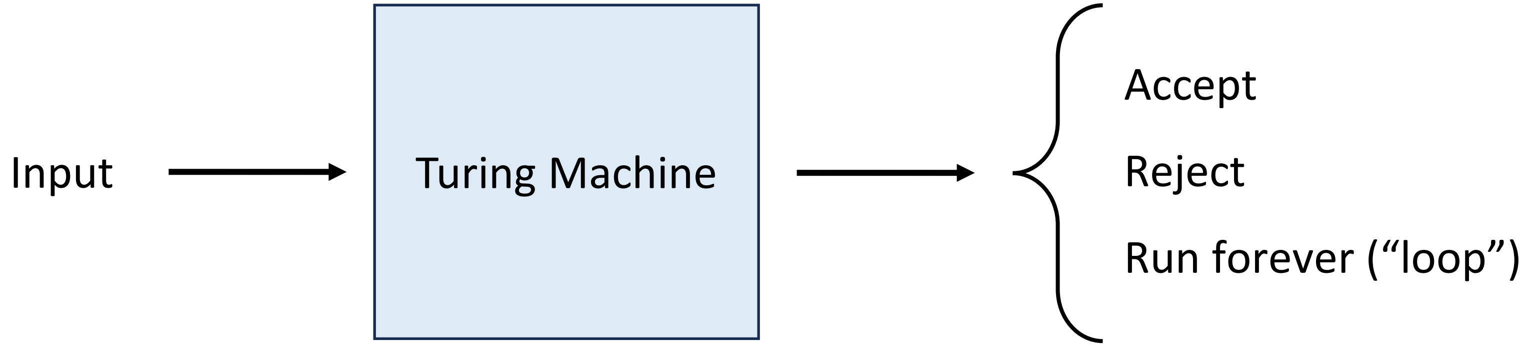


Turing machine initialization

- The head is initially at cell #2 (the first symbol of the input)
- The machine is initially in a special "start state" q_0

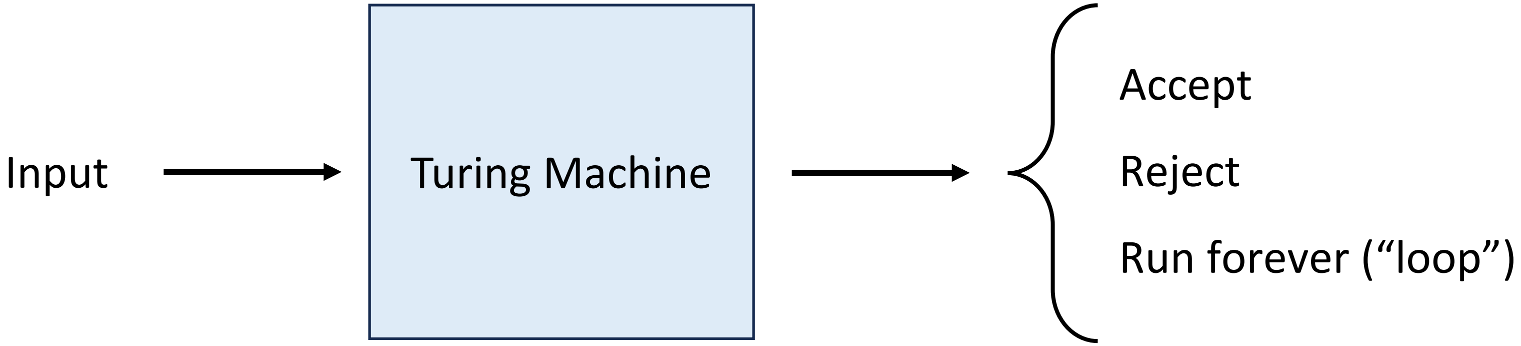


Halting states



- There are two special “halting states,” q_{accept} and q_{reject}
- If the machine ever reaches q_{accept}, this means it has accepted the input
- If the machine ever reaches q_{reject}, this means it has rejected the input
- Either way, the computation is finished. We say that the machine halts

Looping



- It is also possible that the machine runs forever without ever reaching q_{accept} or q_{reject}
- In this case, we say that the machine does not halt, does not accept the input, and does not reject the input