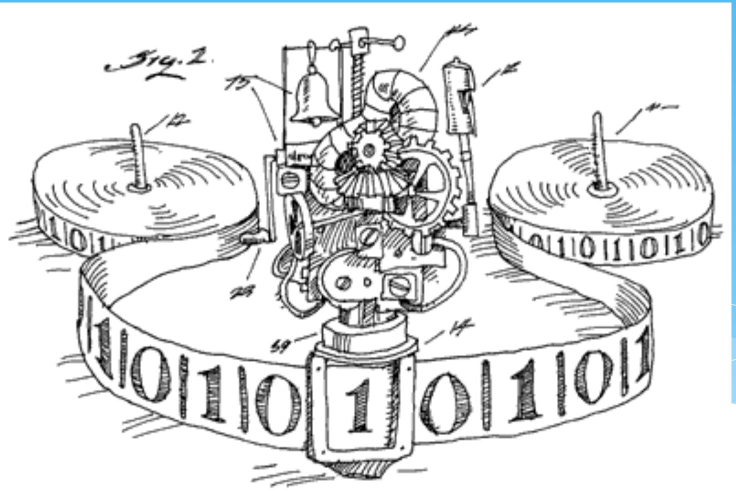
EECS 376: Foundations of Computer Science

Chris Peikert
25 January 2023



Today's Agenda

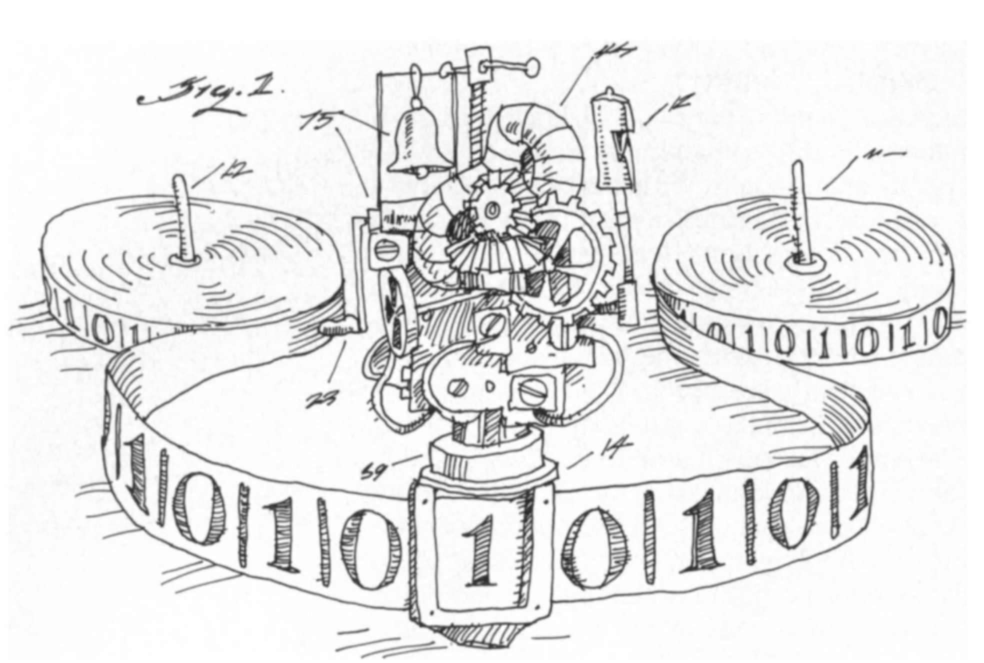
* Overview of **computability**
* **Decision** ("membership") problems
* Deterministic Finite Automata (DFA)
  * A weak class of computing "machine"
* A limit on the power of DFAs

Overview of Computability

* **Q:** Is every "problem" solvable on a "computer"? If so, how? If not, why not?
* **Modeling computation**
  * Formalize notion of "problem" and "computer"
* **The Halting problem and reductions**
  * <u>Prove</u> there are problems that cannot be solved on *any* computer <u>*that could ever exist*</u> (or so we believe)
* **Rice's Theorem**
  * "Any non-trivial program analysis is *impossible*"

"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim." - Edsger Dijkstra.

Modeling Computation



What is a "problem"?

* We consider ***decision problems***, where the goal is to ***decide*** (say "yes" or "no") if a given input object has a certain property

Input
376

Output
No

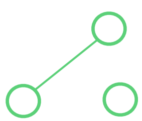
Is the given integer prime?

Input
abba

Output
Yes

Is the given string a palindrome?

Input



Output
No

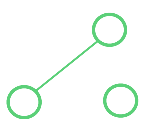
Is the given graph connected?

...The list goes on!
We need to *unify* under a common framework

Languages and Their Decision Problems

- *Any* ***finite*** object (integer, graph, …) can be ***encoded*** as a ***finite string*** of characters from a ***finite alphabet*** (binary, ASCII, …).
- A property corresponds to a *set of strings*: *a.k.a., a* ***language***
- **Q:** What are the languages for the prior decision problems?

Input



Output

No

Is the given graph connected?

Input

010
100
000

Output

No

Is the given string in the language L_{CONN}?

= set of strings $\langle G \rangle$ that encode a connected graph G

The ***decision problem*** for a language L:
Given a string x, ***decide*** if $x \in L$ (say Y/N, acc/rej, etc).

Alphabets, Strings, Languages

- An ***alphabet*** is a finite set of characters, often denoted Σ
  - Often implicit, e.g., $\Sigma =$ ASCII characters or $\Sigma = \{0,1\}$
- A (Σ-)***string*** is a finite sequence of characters from Σ
  - The ***length*** of a string x (# chars) is denoted $|x|$
  - The ***empty string*** is denoted ε; it has length $|\varepsilon| = 0$
- A (Σ-)***language*** is a (possibly infinite) set of (Σ-)strings
  - The language of *all* strings is denoted Σ^*
- **Example:** $\Sigma = \{0,1\}$, $\Sigma^* = \{\varepsilon, 0, 1, 00, \dots\}$, $\left|010\right| = 3$

What is a “computer”?

A “computer” used to mean a ***person*** who performed arithmetic calculations

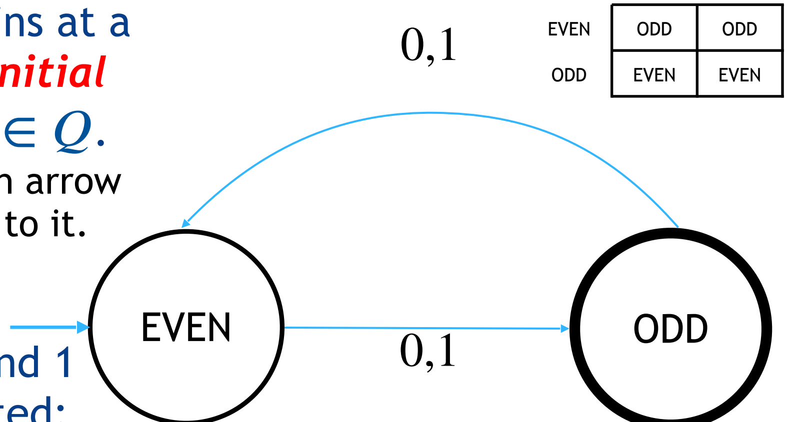
* Alan Turing defined a formal model, called ***Turing machine***, which is *widely believed* to encompass what *any* real computing process could possibly do.
* **Beautiful Idea:** Abstracts the process of a *person/device* working on a (decision) problem, with access to an *unlimited amount of “scratch” paper/memory*.
* ***Church-Turing thesis:*** Any “computer” can be simulated by a Turing machine.
* **Warm-up:** DFAs (≡ person w/o paper): deterministic finite automata

DFA Example (1/4)

Example: DFA that decides $L = \{x \in \{0,1\}^* : |x| \text{ is odd}\}$, the set of binary strings whose lengths are *odd*; alphabet $\Sigma = \{0,1\}$.

(2) It begins at a special ***initial*** state $q_0 \in Q$.
Drawn with arrow pointing to it.

	0	1
EVEN	ODD	ODD
ODD	EVEN	EVEN



Strings 0 and 1 are accepted; ε, 01, 0110 aren't

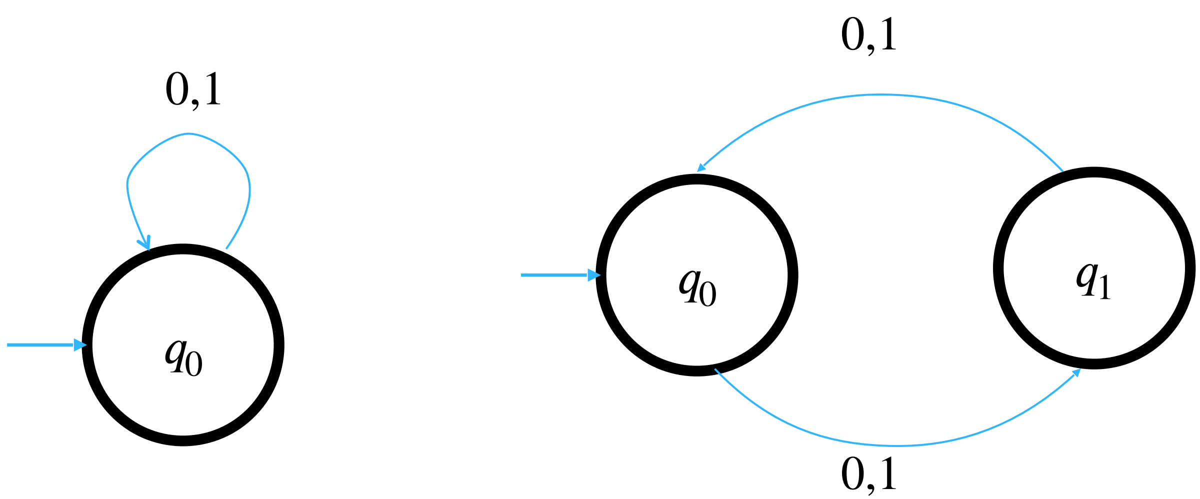
(1) A DFA has a <u>finite</u> set Q of ***states***
('memory').
Each state represented by a distinct vertex.

(3) Given an input string x, DFA reads <u>one character</u> of x at a time and changes state according to the ***transition function*** $\delta: Q \times \Sigma \to Q$
('program').
An edge $q \to q'$ labeled with $a \in \Sigma$ represents that $\delta(q, a) = q'$, i.e., at state q, go to state q' upon reading a.

(4) It ***accepts*** the input x if it stops at an ***accepting*** state after reading all of x.
Drawn as thick/outlined vertices.

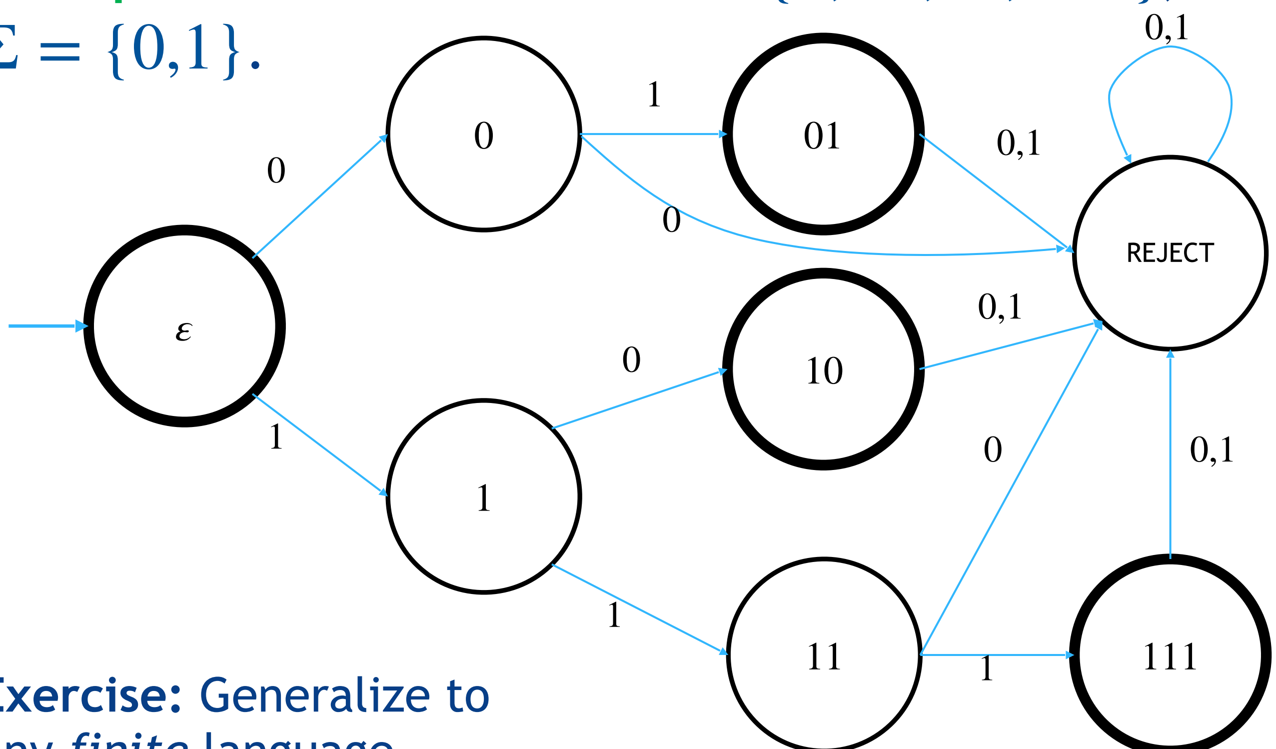
DFA Example (2/4)

Example: DFAs that decide $L = \{0,1\}^*$, the set of <u>all</u> binary strings, for alphabet $\Sigma = \{0,1\}$.



DFA Example (3/4)

Example: DFA that decides $L = \{\varepsilon, 01, 10, 111\}$, for alphabet $\Sigma = \{0,1\}$.



An inescapable "reject" state is often quite useful!

Exercise: Generalize to any *finite* language.

More Examples

L = {w : strings of length at most 5}
L = {w : all strings except the empty string}
L = {w : w contains exactly two 0's}
L = {w : every odd position of w is a 1}
L = {w : w contains the string 01}

L = {w : w does not contain the string 01}

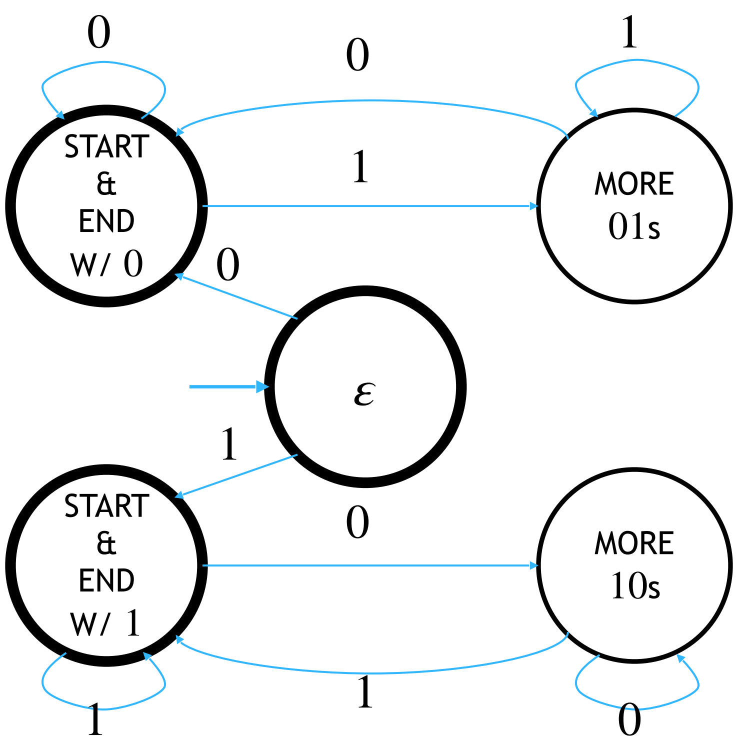
L = {w : w has equal number of 0's and 1's} (no DFA can decide this! (Later.))

#1 tip when making DFAs:
Think about maintaining *state invariants*: properties that must hold whenever certain states are reached.

Theorems:
If L is decidable by a DFA, then so is its complement $\overline{L}$.
If L_1 and L_2 are decidable by DFAs, then so are $L_1 \cup L_2$, and $L_1 \cap L_2$.

DFA Example (4/4)

Example: DFA that decides $L = \{x \in \{0{,}1\}^* \mid$ # of 01s in x = # of 10s in $x\}$, for alphabet $\Sigma = \{0{,}1\}$.



#1 tip when making DFAs:
Think about maintaining *state invariants*: properties that must hold whenever certain states are reached.

<u>D</u>eterministic <u>F</u>inite <u>A</u>utomaton: Formal Definition

- Formally, a DFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:
  - Q is the <u>finite</u> set of ***states***
  - Σ is the ***input alphabet***
  - $\delta : Q \times \Sigma \to Q$ is the ***transition function***
  - $q_0 \in Q$ is the ***initial*** state
  - $F \subseteq Q$ is the subset of ***accepting*** states
- **Takeaway:** DFAs are a simple & weak, but well defined, kind of "computer."

The Language of a DFA

* We say that a DFA
  * ***accepts*** a <u>string</u> x if it ends at an *accepting* state, given x
  * ***decides*** a <u>language</u> L if it *accepts* every string $x \in L$ **and** *does not accept* ('rejects') every string $x \notin L$
* A language is said to be ***regular*** if some DFA decides it.
* **Q:** Is *every* language *regular*?
* *In other words*: is every *<u>decision problem</u>* "solvable" on this simple, weak model of a "computer"?
  * **No!** Intuitively, bottlenecked by DFA's finite memory

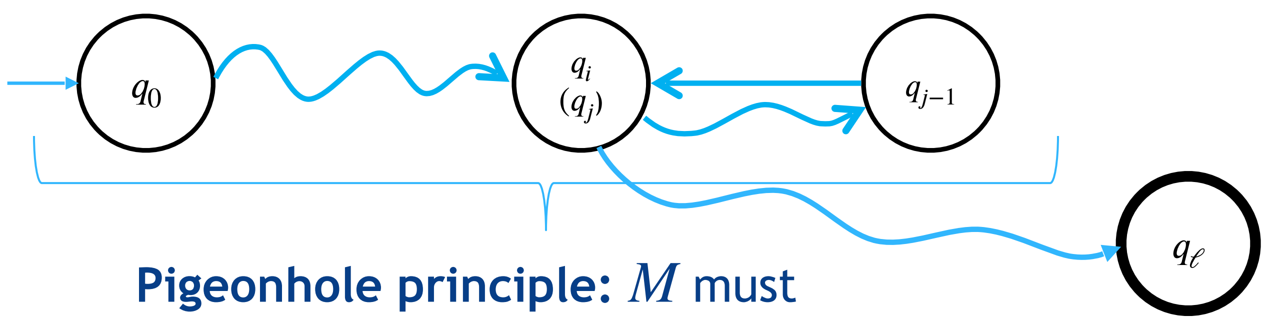
A Thought Experiment

- Imagine you're given a *huge* binary string x
  - $|x| \gg$ number of neurons in your brain
- You can read x as many times as you want, and in any order, but *you can't write anything down*
- **Q:** Can you decide if $x = 0^k 1^k$ for some k?
  - $0^k 1^k$ means k 0s followed by k 1s
- **Rabin and Scott [1959]:** 'Many-read' DFAs ≡ DFAs.
  - DFAs can read the input only once, in sequence. This theorem says that more reads don't help.

Theorem: No DFA decides $\{0^k 1^k \mid k \geq 0\}$.

No DFA decides $\{0^k 1^k \mid k \geq 0\}$

* Suppose that some DFA M decides $\{0^k 1^k \mid k \geq 0\}$.
* Let $n = \#$ of states of M, and let $x = 0^{n+1} 1^{n+1}$.
* **Claim:** We can write $x = uwv$ so that M is in the *same state* before and after reading substring $w \neq \varepsilon$.
* M must accept $uwwv \notin \{0^k 1^k \mid k \geq 0\}$. Contradiction!



Pigeonhole principle: M must *repeat* a state while reading 0^{n+1}