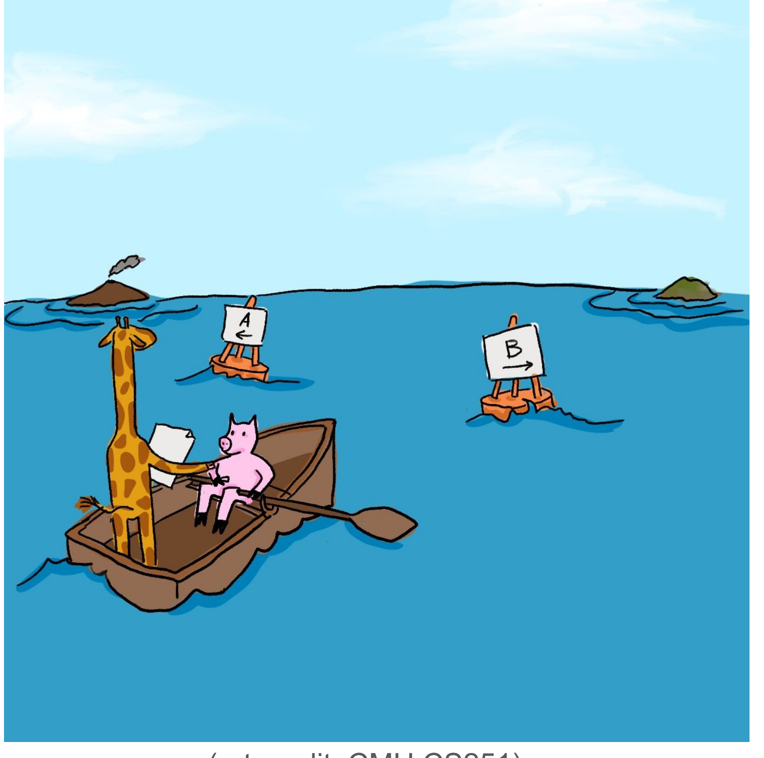
# Introduction to Computability: Deterministic Finite Automata

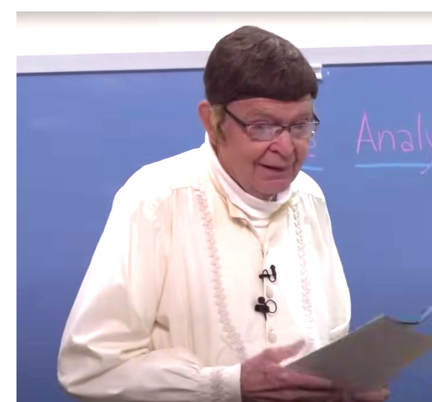



(art credit: CMU CS251)

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

# So far: Analysis of Algorithms

Don Knuth
“Father of Analysis of Algorithms”



# Techniques and Paradigms in this Course

- Divide-and-conquer, greed, dynamic programming, the power of randomness

Problems that are **easy** for a computer

- Computability

Problems that are **impossible** for a computer

- NP-completeness and approximation algorithms

Problems that are **"probably hard"** for a computer

- Cryptography

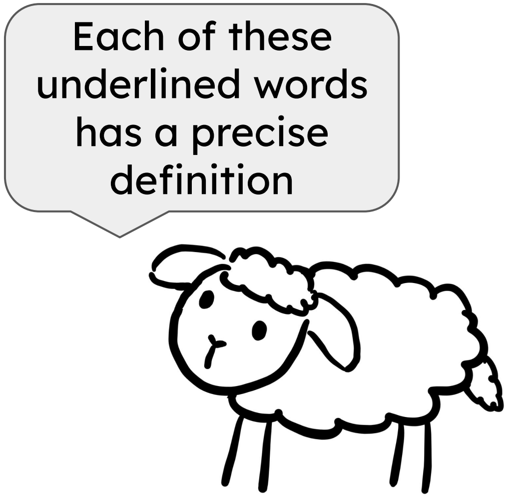
Using "probably hard" problems for our benefit (hiding secrets)

**What is a computer?**

An <u>algorithm</u> solves a <u>problem</u>

if it gives the correct <u>solution</u>

on every <u>instance</u>.




We'll define last 3 terms now.

We'll save <u>algorithm</u> for later.

What is a computational **problem**?

We’ll start with an example.

Example problem:

# MULTIPLICATION

| ***Instance*** (also known as ***input***) | ***Solution*** |
| --- | --- |
| 3, 7 | 21 |
| 610, 25 | 15250 |
| 50, 610 | 30500 |
| 15251, 252 | 3843252 |
| 12345679, 9 | 111111111 |

Example problem:

# PALINDROME

| *Instance* (also known as ***input***) | *Solution* |
|---|---|
| `a` | `Yes` |
| `10101` | `Yes` |
| `selfless` | `No` |
| `huh` | `Yes` |
| `376` | `No` |
| `emus sail i assume` | `Yes` |

A problem is a collection of instances and the solution to each instance.

This is an example of a decision problem:

Problems where the solution is **Yes / No**.

(Also known as **True / False,**
**1 / 0,**
**accept / reject.**)

# Representing problems

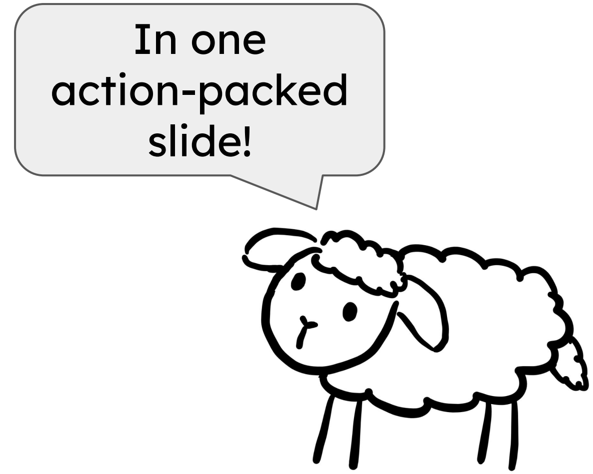
The instances of a problem can be:

- numbers
- strings
- pairs of numbers
- lists of strings
- graphs
- images
- …

These can all be conveniently encoded by **strings**.

Even just by **binary** (0/1) strings.

# String notation




**Alphabet:** **A nonempty finite set $\Sigma$ of symbols.**

$\Sigma = \{0,1\}$ is a popular choice.

**String:** **A finite sequence of 0 or more symbols.**
(or "word")

The empty string is denoted $\varepsilon$.

For any $a \in \Sigma$:

- $a^k$ means k a's
- $a^*$ means ≥0 a's
- $a^+$ means ≥1 a's

$\Sigma^k$ means all strings over $\Sigma$ of length k.

$\Sigma^*$ means **all** (finite) strings over $\Sigma$.

$\Sigma^+$ means all nonempty (finite) strings over $\Sigma$

For any $a,b \in \Sigma$: $a|b$ means a OR b

**Language:** **A collection of strings.**

I.e. any subset $L \subseteq \Sigma^*$.

The empty language is denoted $\varnothing$.

# Examples of Languages

$L = a^* = \{\varepsilon, a, aa, aaa, \dots\}$

$L = 01^*$ = all strings of one 0 followed by zero or more 1's

$L = \{x^k y^k : k \geq 0\}$ = all strings consisting of some number of x's followed by the same number of y's

Question: $L = \{\varepsilon\}$. Is $L = \emptyset$?
Answer: No. L has 1 element, $\emptyset$ has 0 elements

Question: How is $a^*|b^*$ different from $(a|b)^*$

# Representing problems

We can encode instances and solutions as strings.

Thus we can think of a **problem** as a **function**

$$f : \Sigma^* \rightarrow \Sigma^*$$

mapping instances to solutions.

A **decision problem** can be thought of as

$$f : \Sigma^* \rightarrow \{\text{No}, \text{Yes}\}$$

# Representing problems

A **decision problem** can be thought of as

$$f : \Sigma^* \rightarrow \{\text{No}, \text{Yes}\}$$

or equivalently as a **language**

$$L \subseteq \Sigma^*$$

$L = \{x \in \Sigma^* : f(x) = \text{Yes}\}$ $\qquad$ $f(x) = \begin{cases} \text{Yes if } x \in L \\ \text{No if } x \notin L \end{cases}$

E.g.: $L_{\text{PALINDROME}} = \{x \in \Sigma^* : x \text{ is a palindrome}\}$

# What is **computation**?

# What is an **algorithm**?

This lecture:

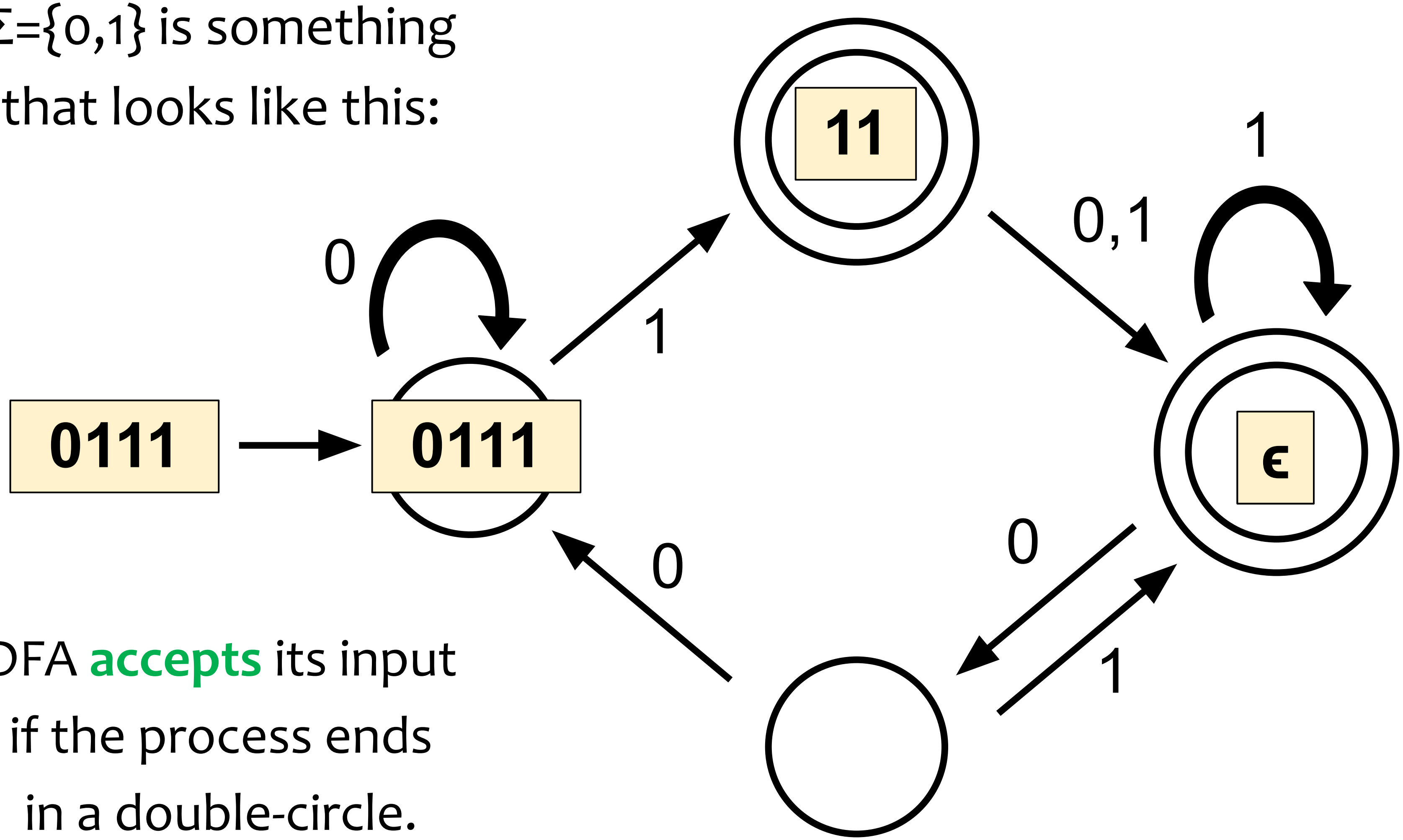
A computational model:

**Deterministic Finite Automata (DFA)**

A good warmup before we study general models of computations next lecture.

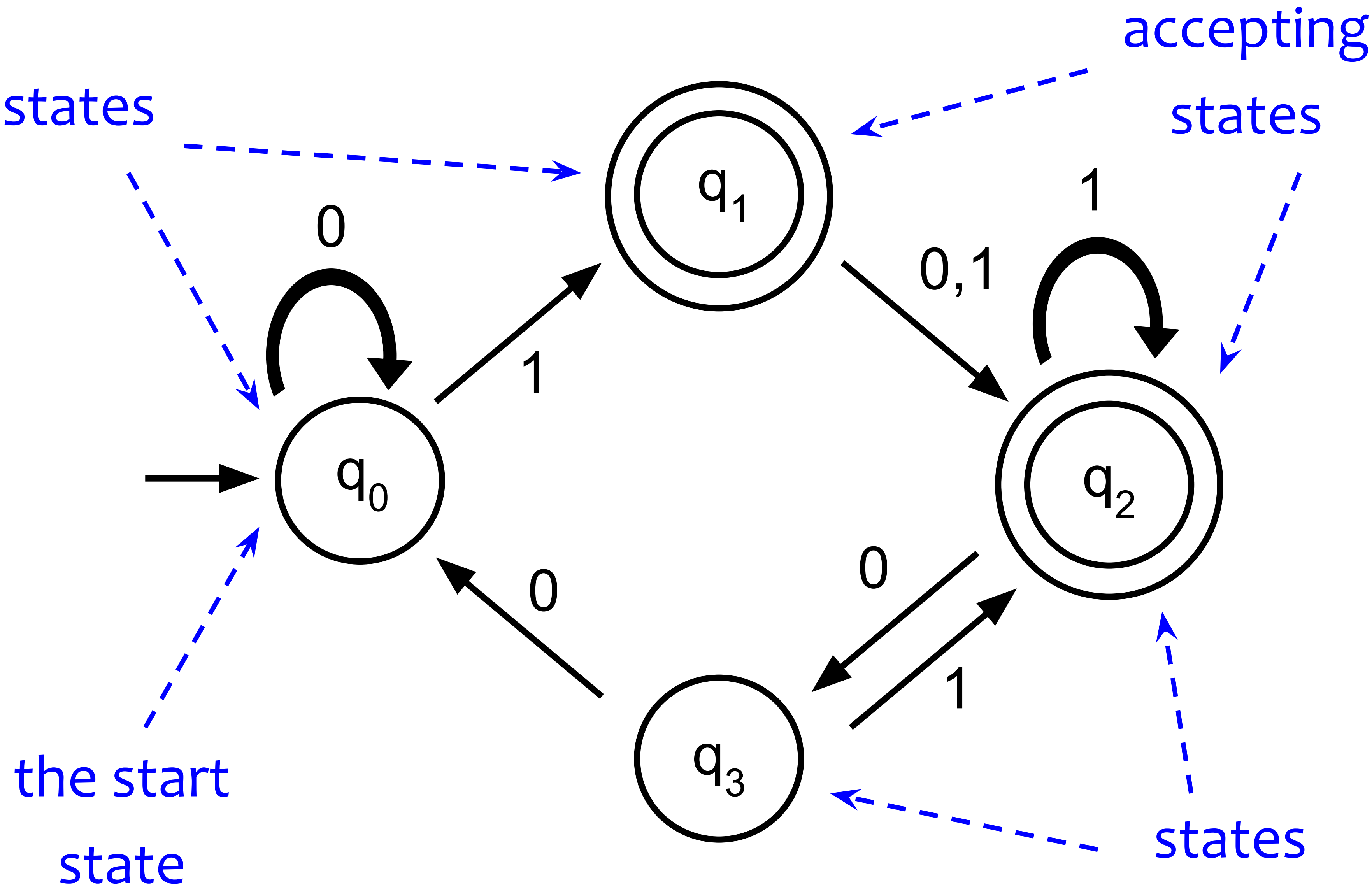
# Deterministic Finite Automata (DFAs)

A DFA over alphabet Σ={0,1} is something that looks like this:




DFA **accepts** its input if the process ends in a double-circle.

# Anatomy of a DFA




transition rules: the labeled arrows

# Computing with DFAs

Let M be a DFA, using alphabet $\Sigma$.

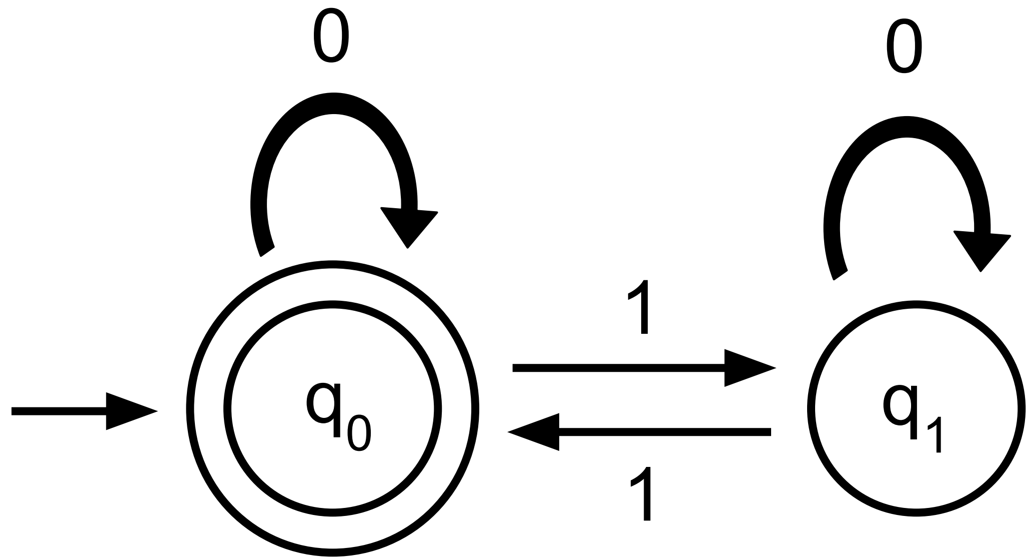
M accepts some strings in $\Sigma^*$ and rejects the rest.

Definition: $L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}$

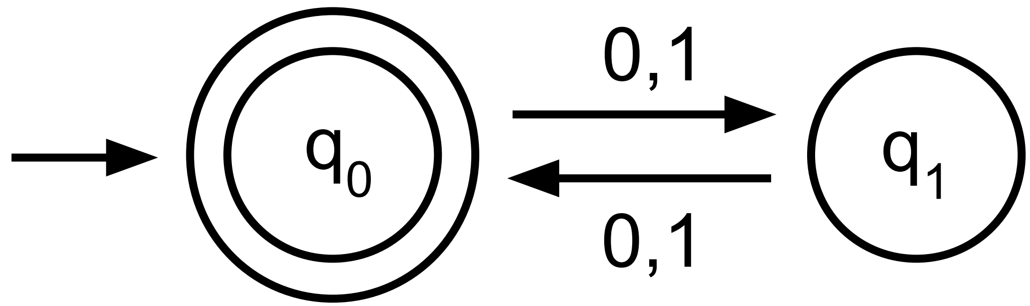
Called the "language decided by M".

If L is a language,

we say M decides L if $L(M) = L$.

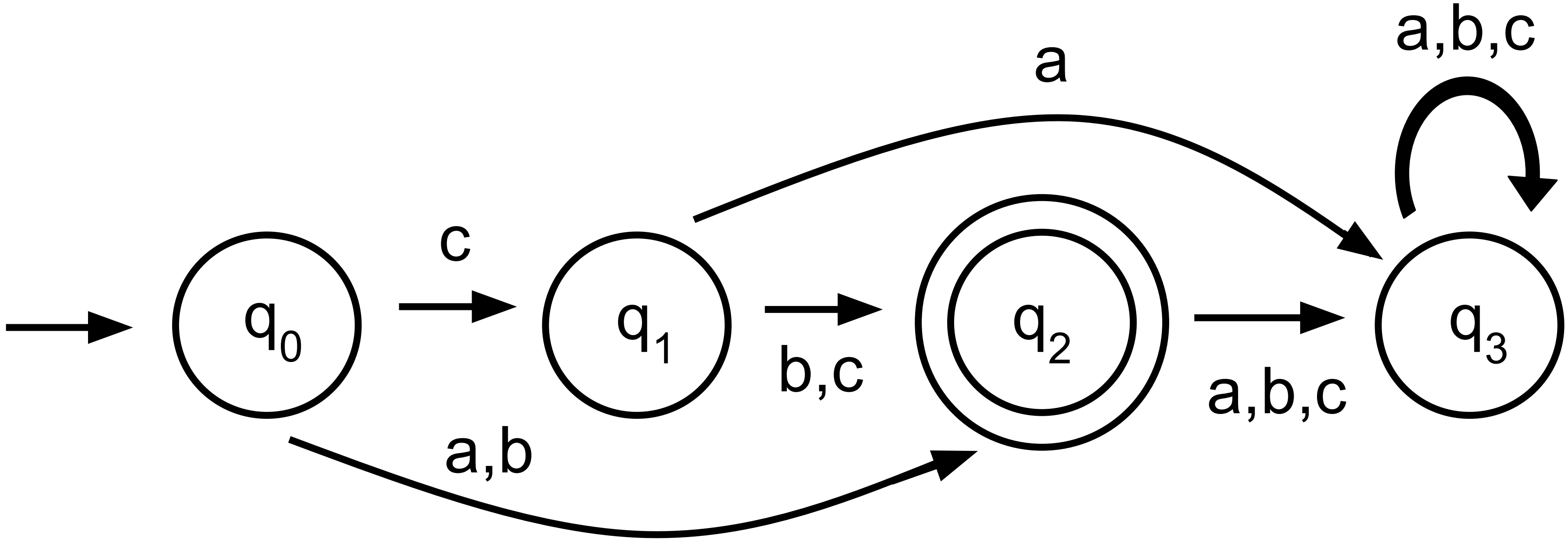



What language does this DFA decide?



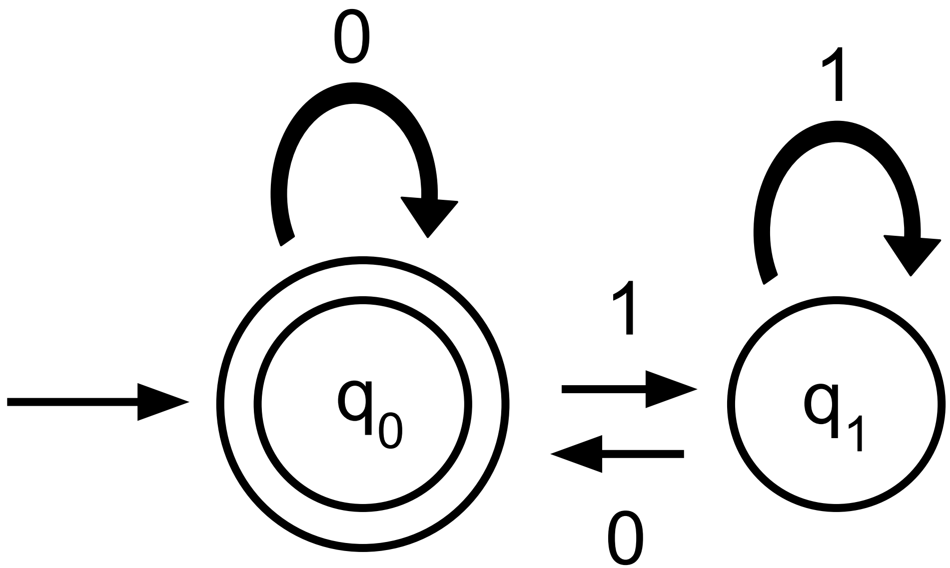

What language does this DFA decide?

M is the following DFA,
with alphabet Σ={a,b,c}:




L(M) =

M:



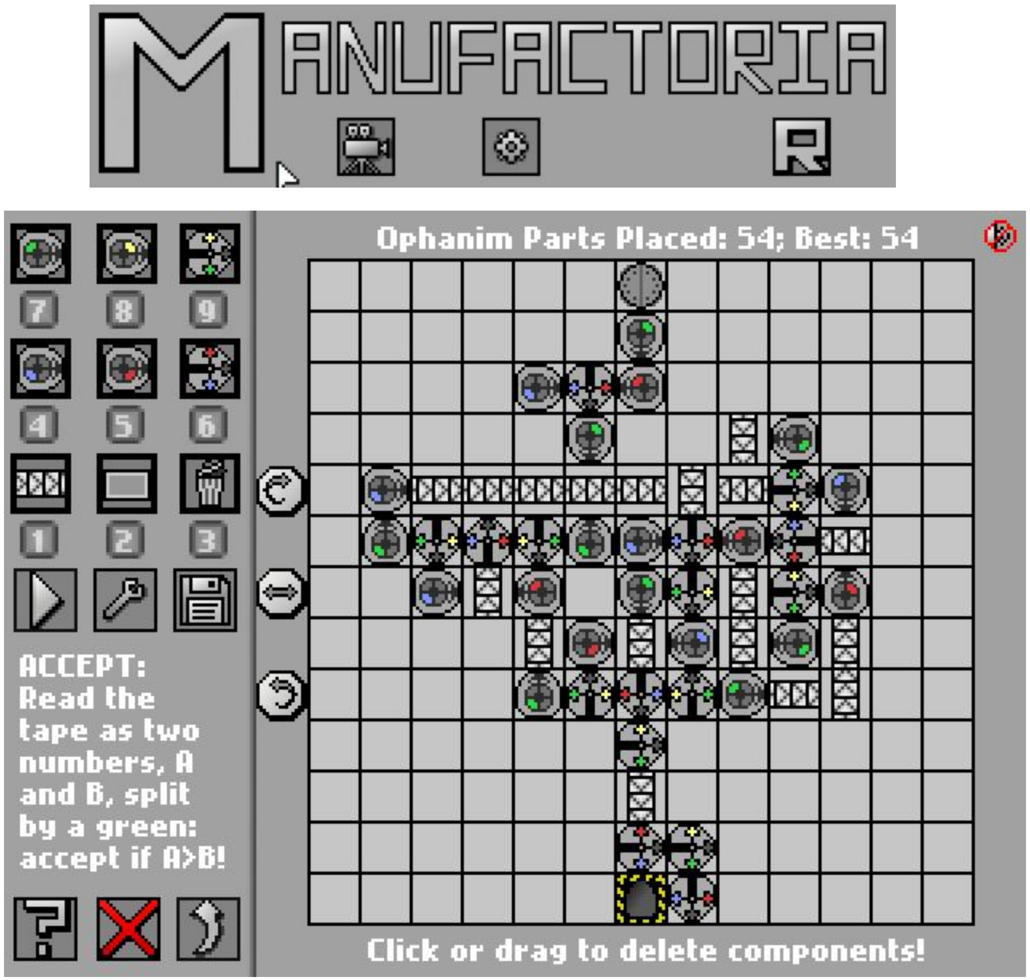

L(M) =

**Fact:** If there is a DFA that decides a language **L**, then there is also a DFA that decides the **complement of L**. Why?

E.g. **L** = {strings containing "01"}

**complement of L** = {strings NOT containing "01"}

# What got me interested in CS in the first place…




# Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:

$$M = (Q, \Sigma, \delta, q_0, F)$$

$Q$ is a nonempty finite set of states,

$\Sigma$ is an alphabet,

$\delta : Q \times \Sigma \rightarrow Q$ is the state-transition function,

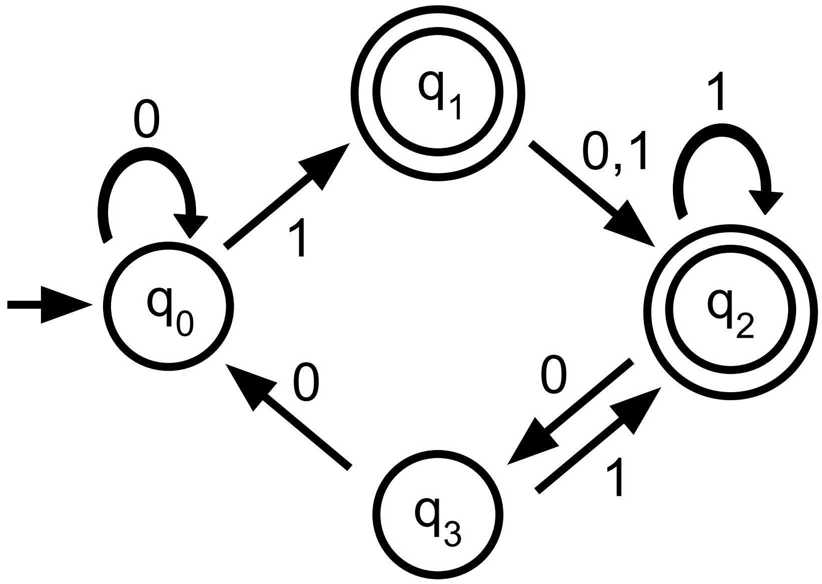
$q_0 \in Q$ is the start state,

$F \subseteq Q$ is the set of accepting states.

# Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:

$M = (Q, \Sigma, \delta, q_0, F)$




$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0,1\}$

$\delta$ we'll come back to

$q_0$ is the start state

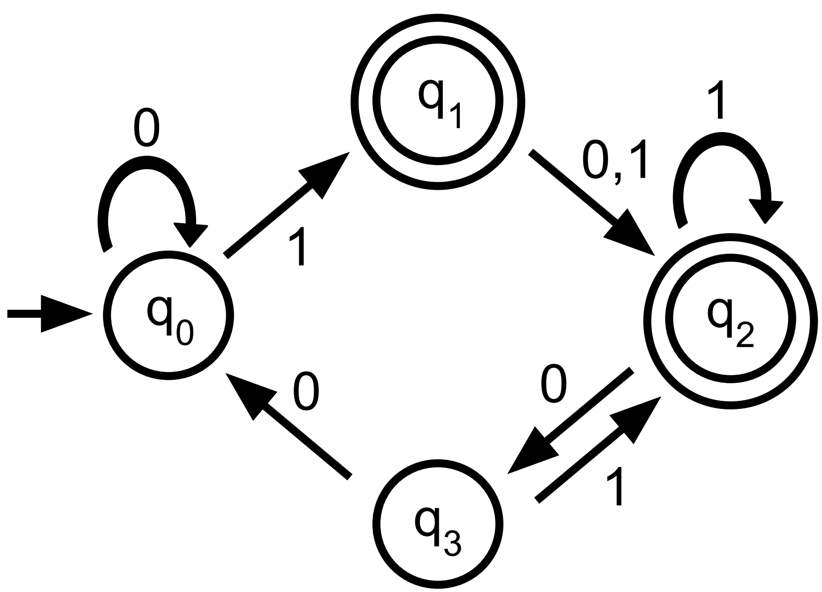
$F = \{q_1, q_2\}$

# Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:

$$M = (Q, \Sigma, \delta, q_0, F)$$

$\delta : Q \times \{0,1\} \rightarrow Q$ is…




| $\delta$ | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_2$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_0$ | $q_2$ |

1. For Σ = {a,b}, consider the language of strings with exactly one b
   a. Write using string notation
   b. Draw a DFA

1. Draw a DFA for ab*|ba*

1. Draw a DFA for L = $\{x^k y^k : k \geq 0\}$

This space is for rent

(Call 1(800)-376-DFAS)

# Impossibility Proof

Suppose for contradiction there’s a DFA **M** that decides ________

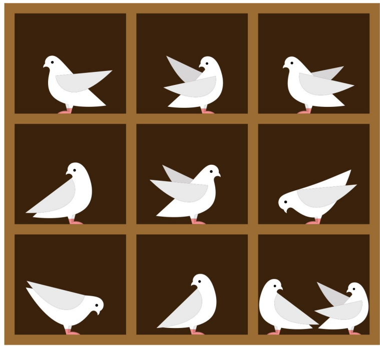
Let **s** = #states of **M**.

Consider the input string $x^{s+1}y^{s+1}$.

There are **s states** (pigeonholes) and **s+1 x’s** (pigeons), so there must be two values $i, j \leq s+1$ such that **state**($x^i$) = **state**($x^j$).

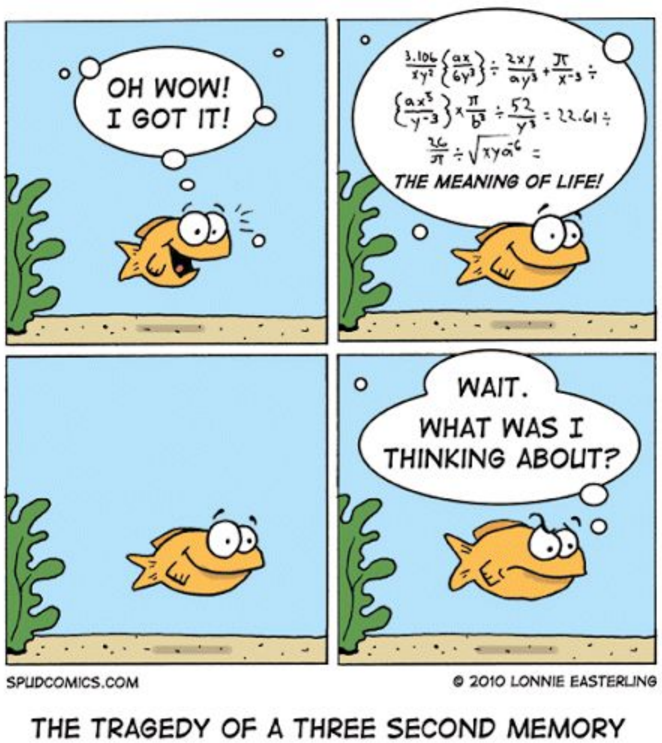
**Claim.** The two inputs $x^i y^i$ and $x^j y^i$ have the same final state. Why?

But **M** is supposed to accept $x^i y^i$ and reject $x^j y^i$.
Contradiction!

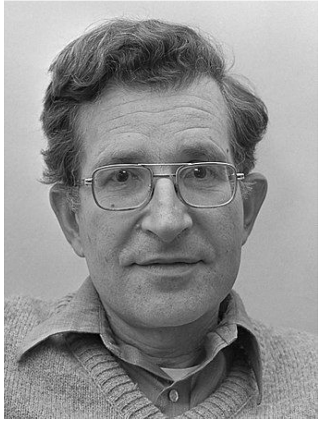


Intuitive reason why DFAs cannot decide many languages:

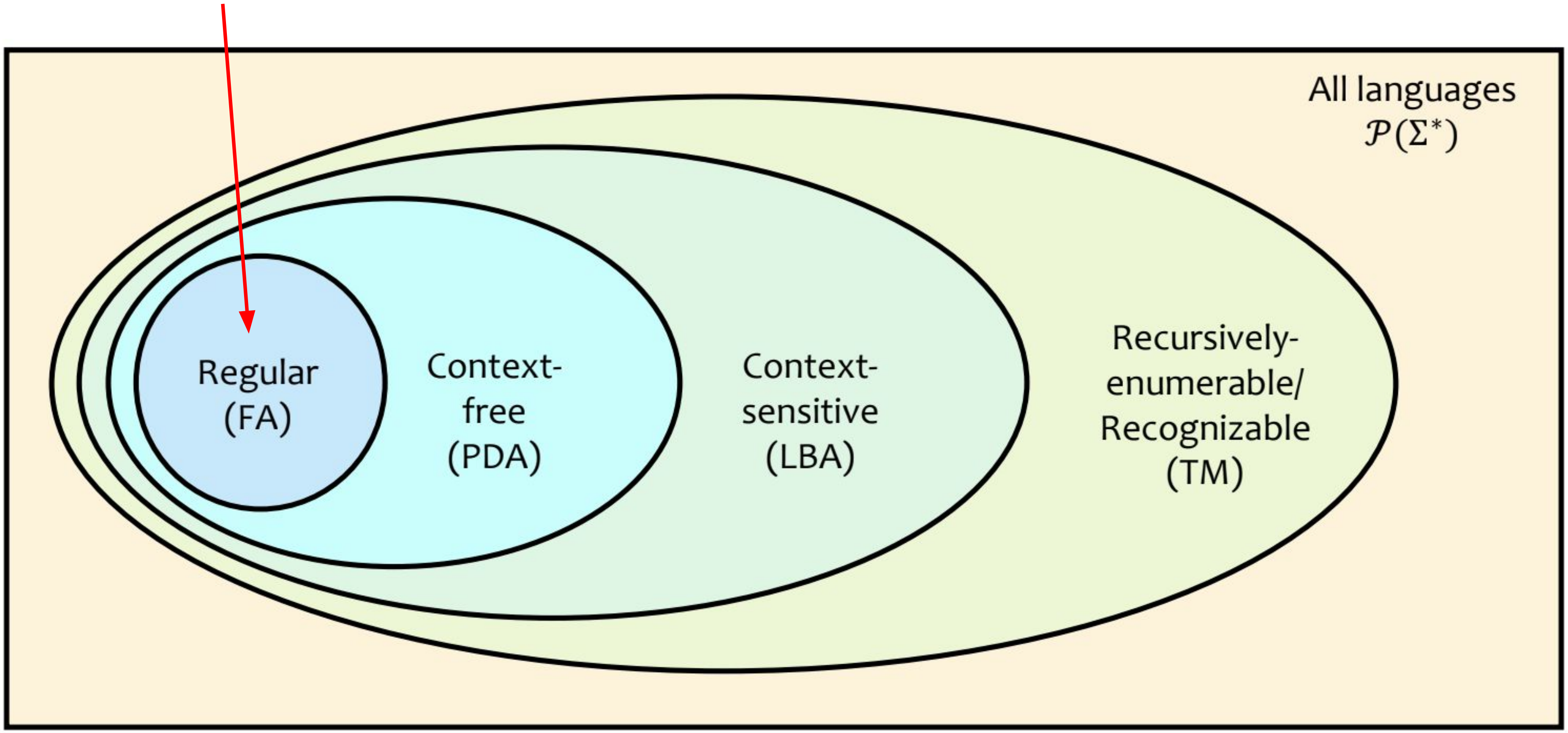
**No memory!**



# The Chomsky Hierarchy (1956)



**“Regular Language”:** Language decidable by some DFA




More powerful memory system →

# But DFAs are still used in practice:

- Software for designing and checking the behavior of digital circuits
- Lexical analyzer for compilers
- (and more)