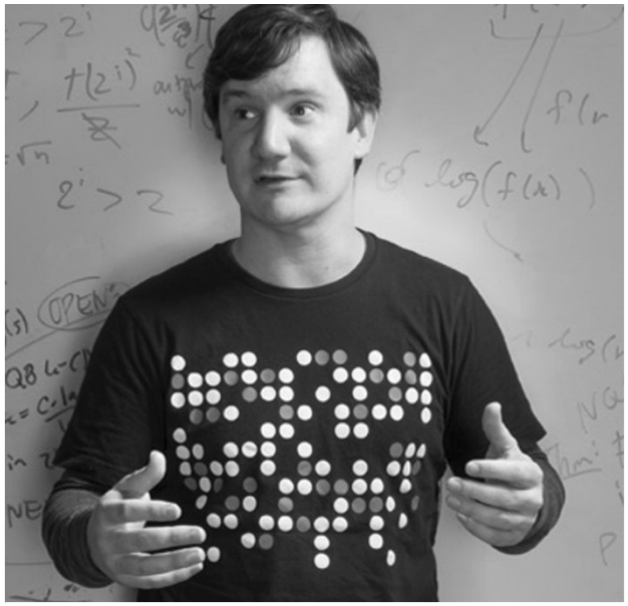
6.1400

Automata, Computability, and Complexity

csail.mit.edu/~rrw/6.1400-2025

INSTRUCTORS & TAs



Ryan Williams

Jiatu Li

Jakin Ng

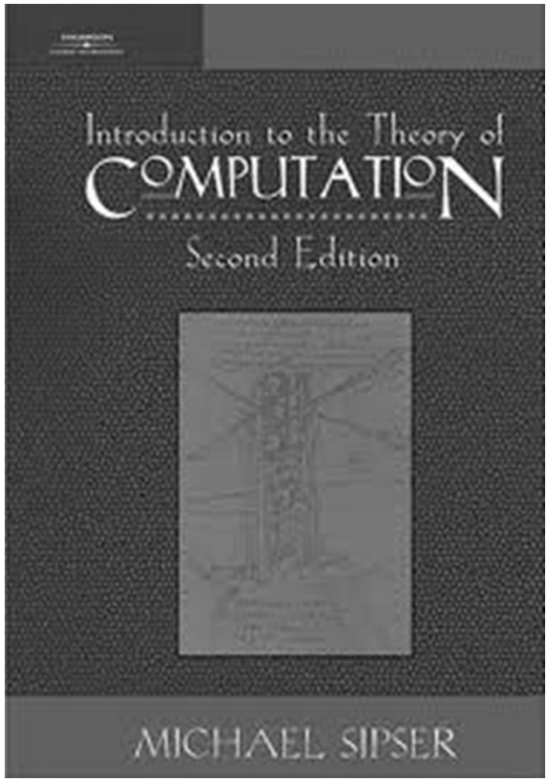
Recitations and Office Hours

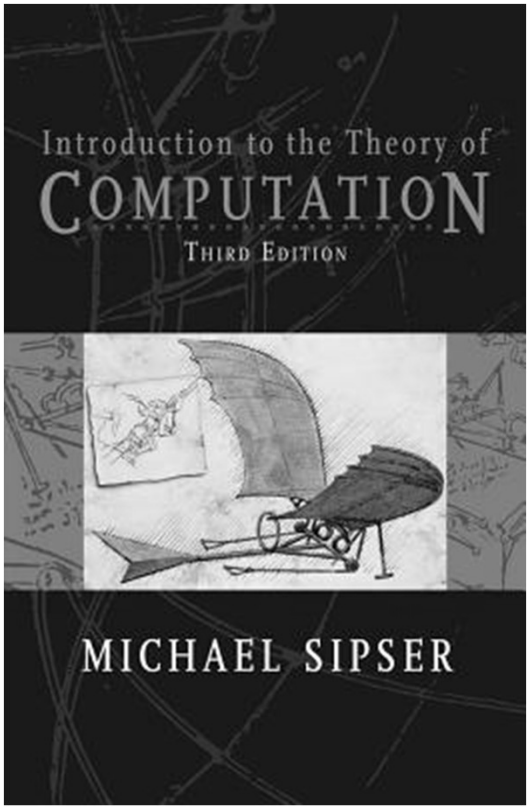
Recitations on Fridays
Jakin: 11am-noon (**4-257**)
Jiatu: 1pm-2pm (**24-121**)

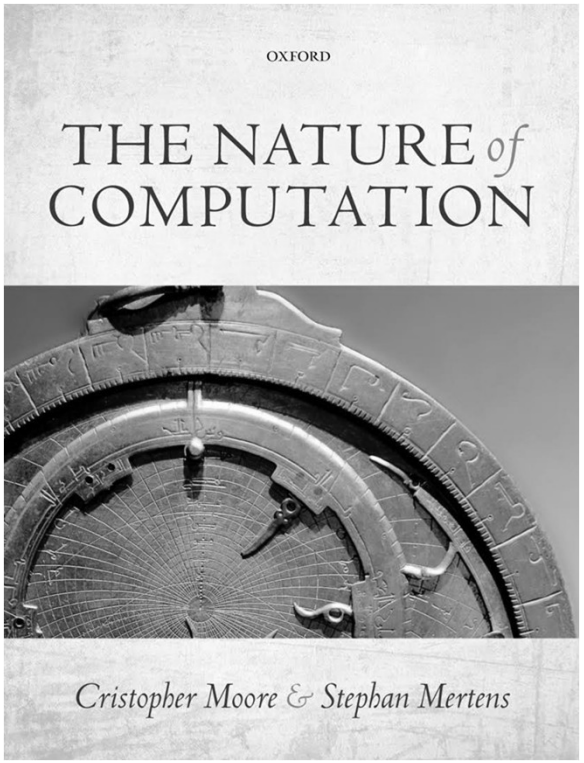
You're not required to attend recitations…
But it is *strongly* recommended
Attending lectures is also strongly recommended!

Office Hours (tentative):
Jiatu: Tuesday 4pm-5:30pm (**Stata, G5 Lounge**)
Jakin: Tuesday 12:30pm-2:00pm (**Stata, G5 Lounge**)
Ryan: Wednesday 11am-12:30, **32-G638**

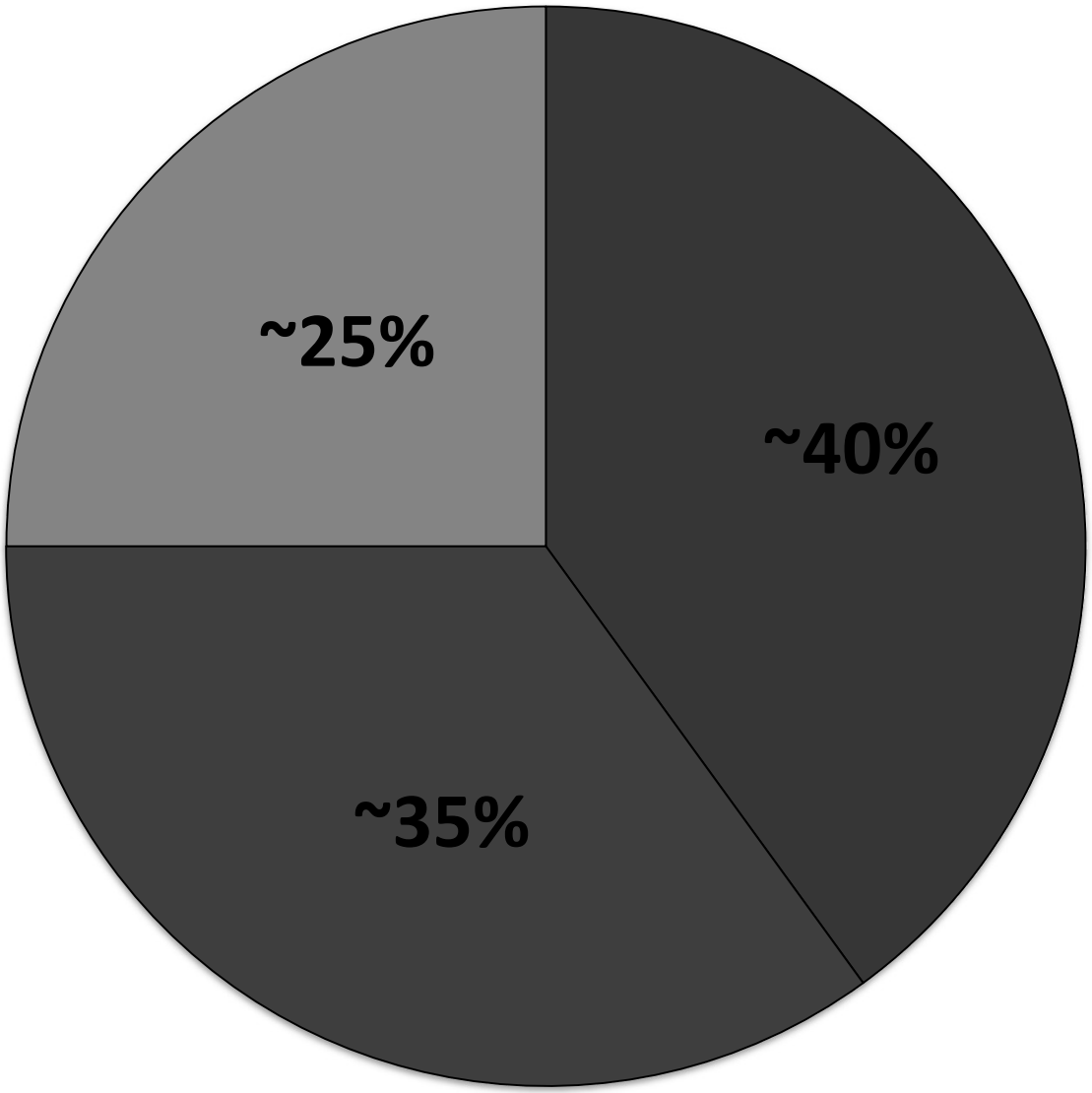
Textbook(s)


Introduction to the Theory of
COMPUTATION
Second Edition
MICHAEL SIPSER


Introduction to the Theory of
COMPUTATION
Third Edition
MICHAEL SIPSER


OXFORD
THE NATURE of
COMPUTATION
Cristopher Moore & Stephan Mertens

Grades



Homework
Final
Midterm

Class participation also counts

Homework / Problem Sets / Psets / Pests

Homework will come out on most Thursdays and will be due on Wednesdays, at 11:59pm (<= 9 psets)

No late days allowed (except from S^3) but your lowest homework grade will be dropped

Use a word processor for written parts of assignments! We strongly recommend LaTeX

(You can scan any drawn figures and include in the PDF)

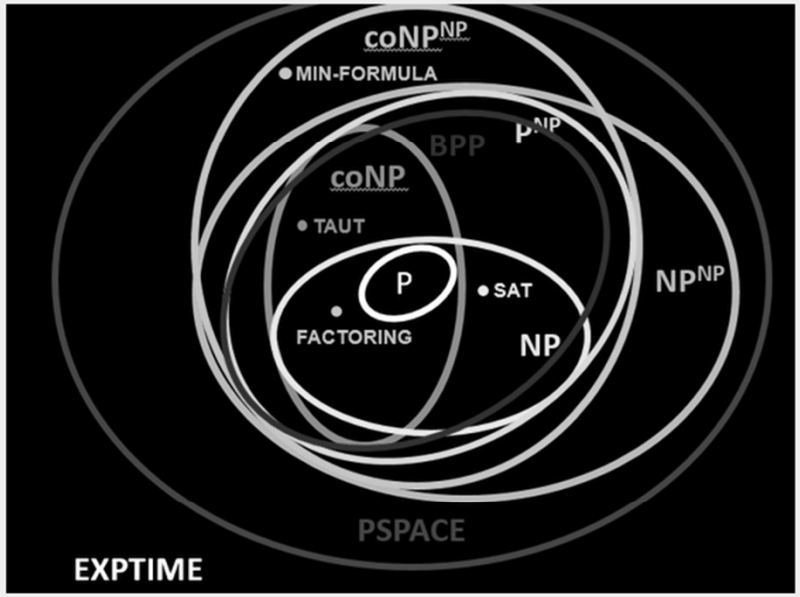
We will provide LaTeX source code for every homework assignment – fill it in with your answers!

Collaboration Policy

You may collaborate with others, but you must:

- *Try to solve all problems by yourself first*
- List your collaborators on each problem
- ***Write your own solutions***
- If you receive a significant idea from a source, you must ***acknowledge the source*** in your solution.
- No LLM assistance (ask on piazza instead!)

6.1400 / 18.400 - Automata, Computability, and Complexity Theory - Spring 2025



[General Info] [Problem Sets] [Lectures] [Exams]

Announcements and Q&A on Piazza

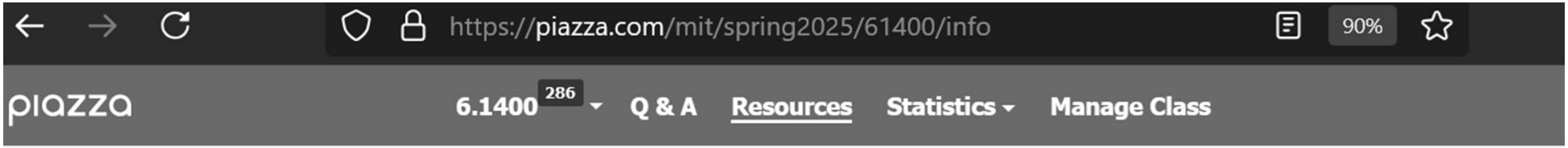
MathJax check: If you see a fancy equation here

$$NP = \bigcup_k NTIME[n^k]$$

then the math on this page is working. If you don't, then you probably disabled JavaScript or your browser is wack. Sorry.

Introduction

What is computation? Given a definition of a computational model, what problems can we hope to solve in principle with this model? Besides those solvable in principle, what problems can we hope to *efficiently* solve? This course provides a mathematical introduction to these questions. In many cases we can give completely rigorous answers; in other cases, these questions have become major open problems in both pure and applied mathematics!



Massachusetts Institute of Technology (MIT) - Spring 2025

6.1400: Computability and Complexity Theory

+ Add Syllabus

Course Information | Staff | Resources

Description

Edit

What is computation? Given a definition of a computational model, what problems can we hope to solve in principle with this model? Besides those solvable in principle, what problems can we hope to efficiently solve? This course provides a mathematical introduction to these questions. In many cases we can give completely rigorous answers; in other cases, these questions have become major open problems in both pure and applied mathematics!

By the end of this course, students will be able to classify computational problems given to them, in terms of their computational complexity (Is the problem regular? Not regular? Decidable? Recognizable? Neither? Solvable in P? NP-complete? PSPACE-complete?, etc.) They will also gain a deeper appreciation for some of the fundamental issues in computing that are independent of trends of technology, such as the Church-Turing Thesis and the P versus NP problem. Prerequisites: 6.042 or equivalent mathematical maturity.

General Information

Edit

Course webpage (with syllabus):

Announcements

Add an Announcement
Click the Add button to add an announcer

This class is about the theory of computation

What is computation?

What can and cannot be computed?

What can be *efficiently* computed?

Philosophy, mathematics, and engineering

Why take this class?

new ways of thinking about computing

different models, different perspectives

theory often drives practice

mathematical models of computation predated computers

(present-day example: we "know" a lot about quantum computing, but no large-scale quantum computers have been built yet!)

math is good for you!

defs, thms, and pfs… yum yum

some of the most important math of this century and last!

timelessness

Course Outline

1. Finite Automata: *Simple* Models

DFAs, NFAs, regular languages, regular expressions, proving no DFA exists (non-regular languages), Myhill-Nerode Theorem, computing the *minimum* DFA, streaming algorithms, communication complexity

2. Computability Theory: *Powerful* Models

Turing Machines, Universal Models and the Church-Turing Thesis, decidable/recognizable languages, undecidability, reductions and oracles, Rice's theorem, Kolmogorov Complexity, even the foundations of mathematics (what can and can't be *proved)*…

3. Complexity Theory: Time and Space Bounded Models

time complexity, classes P and NP, NP-completeness, polynomial time with oracles, space complexity, PSPACE, PSPACE-completeness, randomized complexity theory, other topics TBA

Course Outline

1. Finite Automata: started in the 1940's

DFAs, NFAs, regular languages, regular expressions, non-regular languages, Myhill-Nerode Theorem, computing the *minimum* DFA, streaming algorithms, communication complexity

2. Computability Theory: started in the 1930's

Turing Machines, Universal Models and the Church-Turing Thesis, decidable/recognizable languages, undecidability, reductions and oracles, Rice's theorem, the recursion theorem, Kolmogorov Complexity, even the foundations of mathematics…

3. Complexity Theory: started in the 1960's

time complexity, classes P and NP, NP-completeness, polynomial time with oracles, space complexity, PSPACE, PSPACE-completeness, randomized complexity theory, other topics TBA

CS103 vs **CS154**

PART 1

Finite Automata

PART 2

Computability Theory

PART 3

Complexity Theory

CS103 gave you a decent intro to all three of these parts

If you haven't taken CS103, that's OK, but be warned...

We will cover each part in much more depth than 103

This class will emphasize

MATHEMATICAL PROOFS

A good proof should be:

Clear -- easy to understand

Correct

Problem Set 0 will help you calibrate yourself: watch for it!
(Should take little time to do)

In writing mathematical proofs, it can be very helpful to provide three levels of detail

- **First level: a short phrase/sentence giving "hints" of the proof**

  (e.g. "Proof by contradiction," "Proof by induction," "Pick the thing at random")

- **Second level: a short, one paragraph description of the main ideas**

- **Third level: the full proof (and nothing but the proof)**

Prof. Sipser wrote his much of his book in this way.
I encourage you to write your solutions in this way!

Let's do an example.

Suppose $A \subseteq \{1, 2, \ldots, 2n\}$ with $|A| = n+1$

TRUE or FALSE?
There are always two numbers x, y in A such that x divides y (x is a factor of y)

TRUE

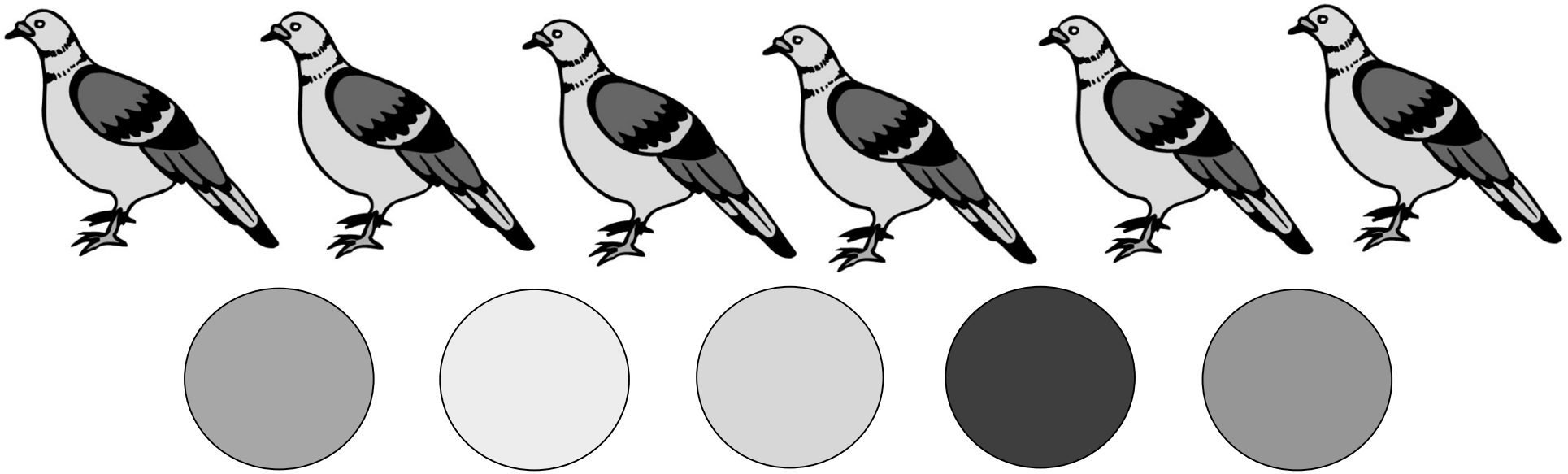
Example: $A \subseteq \{1, 2, 3, 4\}$ and $|A|=3$ *(the case of n=2)*
If 1 is in A, then 1 divides every number.
If 1 isn't in A, then A = {2,3,4}, and 2 divides 4

LEVEL 1

HINT 1:

THE PIGEONHOLE PRINCIPLE

If you drop 6 pigeons in 5 holes, then at least one hole will have more than one pigeon

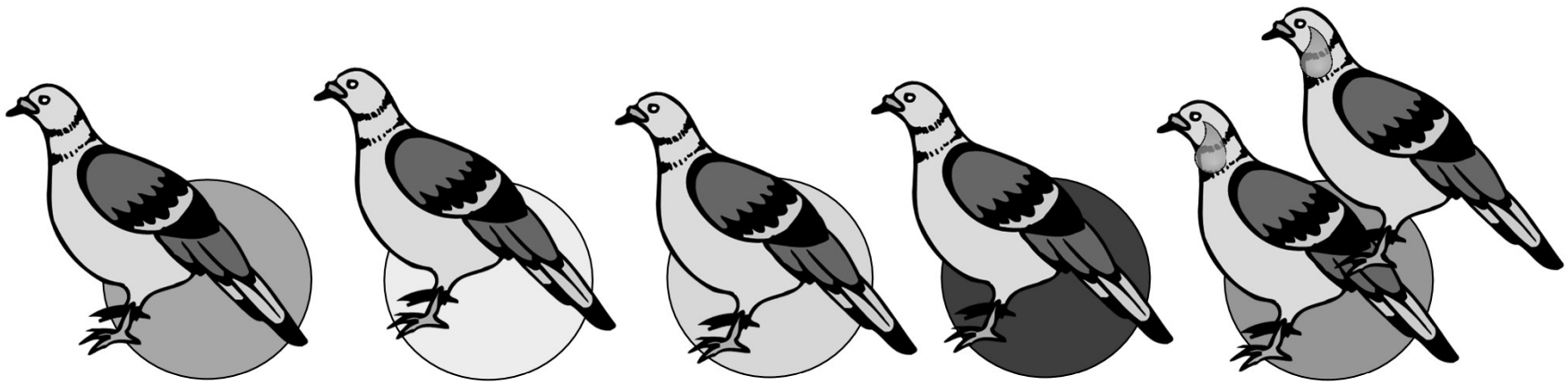


LEVEL 1

HINT 1:

THE PIGEONHOLE PRINCIPLE

If you drop 6 pigeons in 5 holes, then at least one hole will have more than one pigeon



LEVEL 1 **"We'll use the Pigeonhole Principle"**

HINT 1:

THE PIGEONHOLE PRINCIPLE

If you drop n+1 pigeons in n holes then at least one hole will have more than one pigeon

HINT 2:

Every integer a can be written as
$a = 2^k \cdot m$, where m is an odd number (k is an integer)
Call m the "odd part" of a

Examples: The odd part of 3 is 3.
Odd part of 8 is 1. Odd part of 12 is 3.

LEVEL 2

Proof Idea:

Given $A \subseteq \{1, 2, \ldots, 2n\}$ and $|A| = n+1$

Applying the pigeonhole principle,
we'll show there are elements a_1 and a_2 of A

such that $a_1 = 2^i \cdot m$ and $a_2 = 2^j \cdot m$
for some odd m and integers $i < j$

Then a_1 divides a_2

LEVEL 3

Proof:

Suppose $A \subseteq \{1, 2, \ldots, 2n\}$ with $|A| = n+1$

Write each element of A in the form $a = 2^k \cdot m$ where m is an odd number in {1, ..., 2n}

Note: There are n odd numbers in {1, …, 2n}

Since $|A| = n+1$, there are two distinct numbers in A with the same odd part, by P.H.P.

Let a_1 and a_2 have the same odd part m, where $a_1 < a_2$. Then $a_1 = 2^i \cdot m$ and $a_2 = 2^j \cdot m$ where $i < j$, so a_1 divides a_2. QED

What's the right level of detail in a proof?

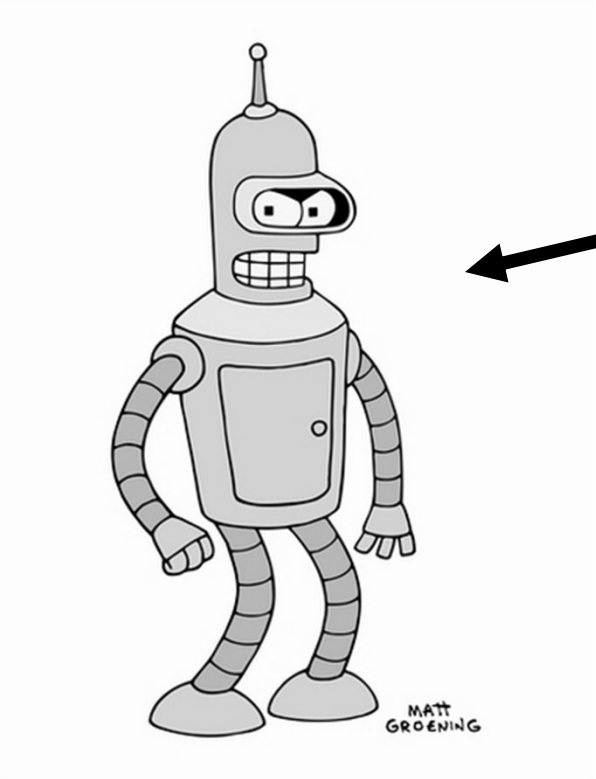
During lectures, my proofs will generally contain the first two levels, but only part of the third (TAs will guide you through some "third levels")

Think about how to fill in the details!
You aren't required to do this (except on certain problems in homework/exams) but it can really help you learn.

In this course, it's often the case that the big ideas and concepts are more important than gritty details!

Come by office hours or ask (privately) on piazza if you worry about your level of detail in a proof!

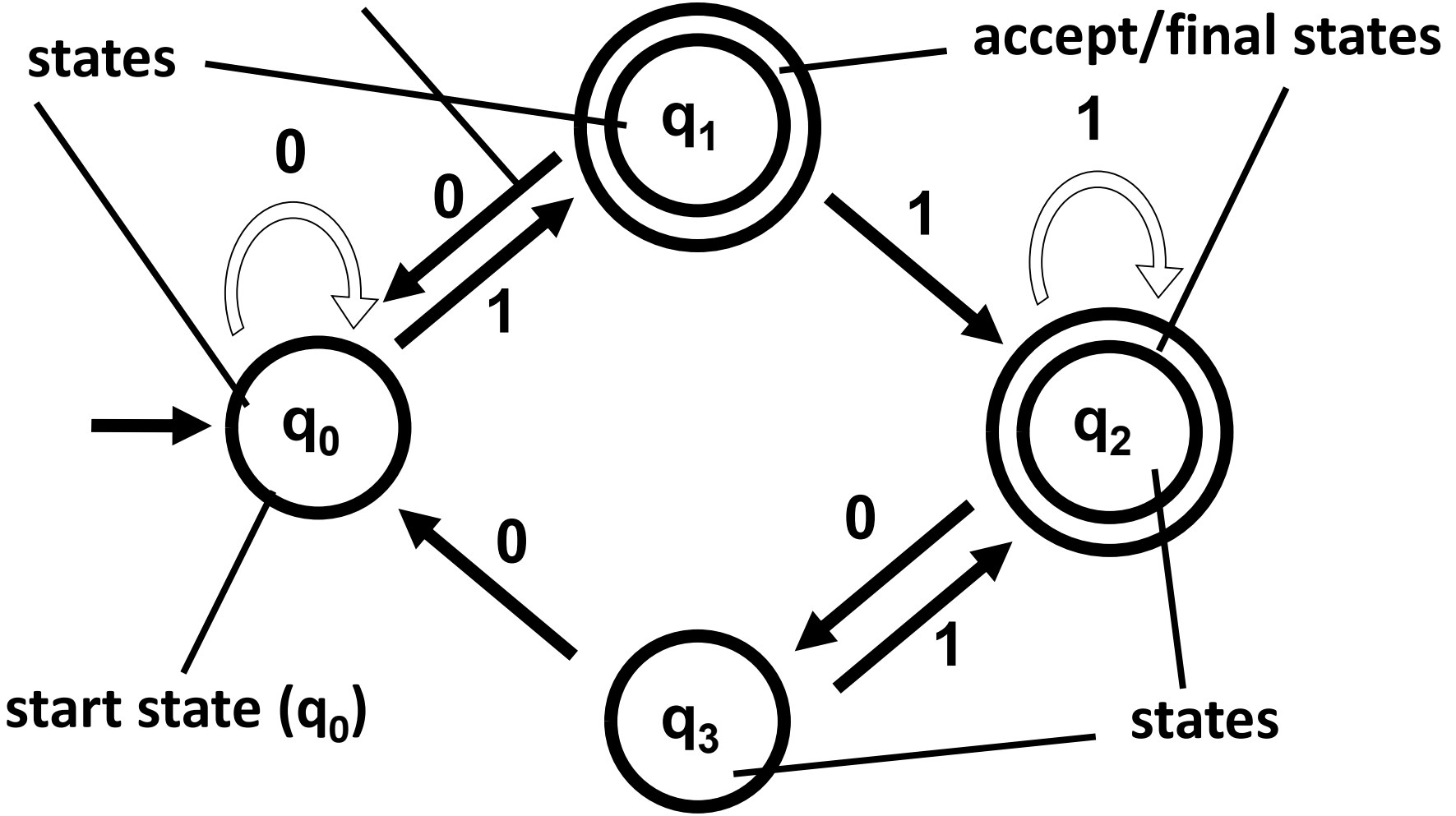
Deterministic **F**inite **A**utomata



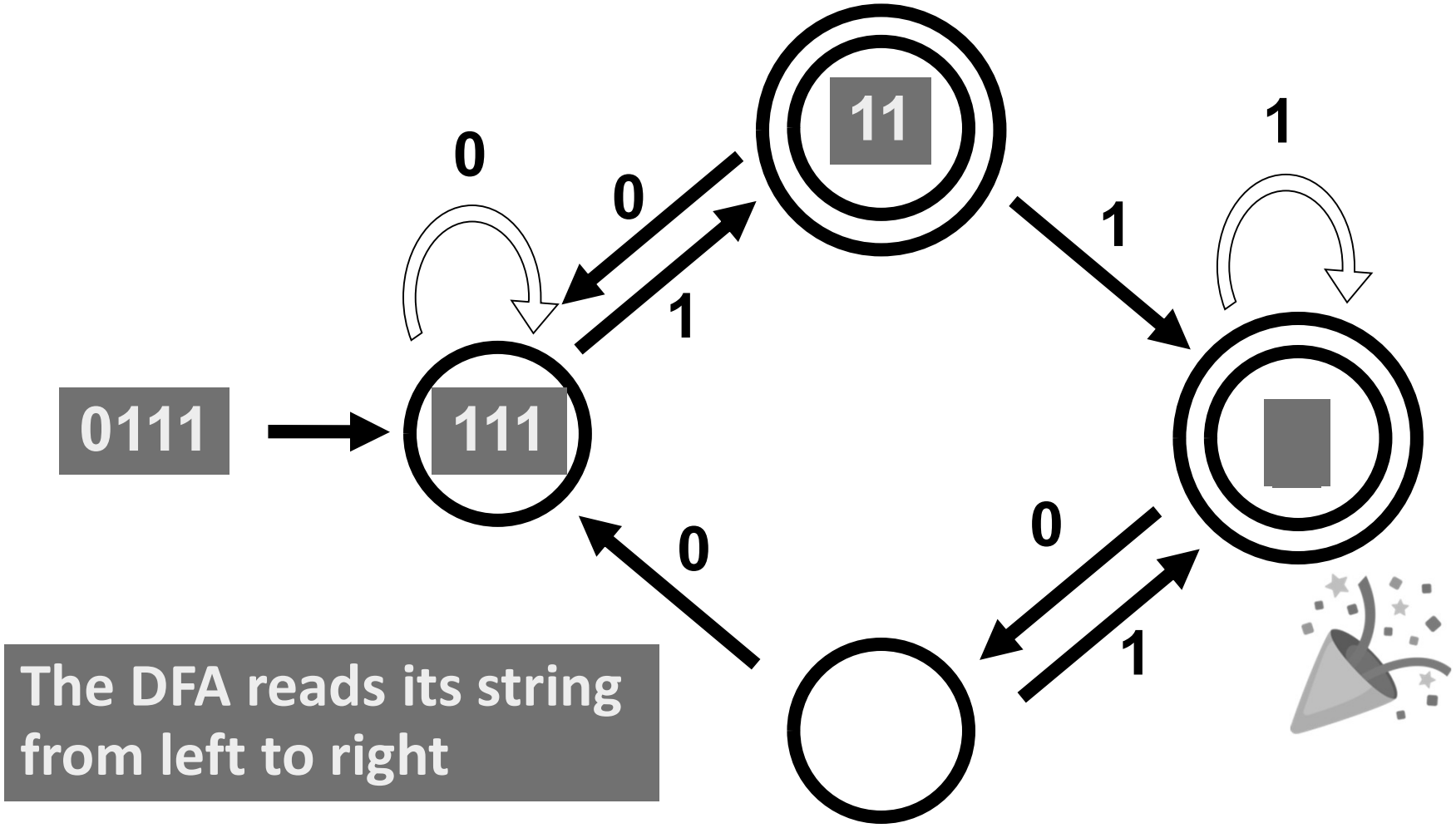
(not a DFA)

Anatomy of Deterministic Finite Automata

transition: ***for every state and alphabet symbol***



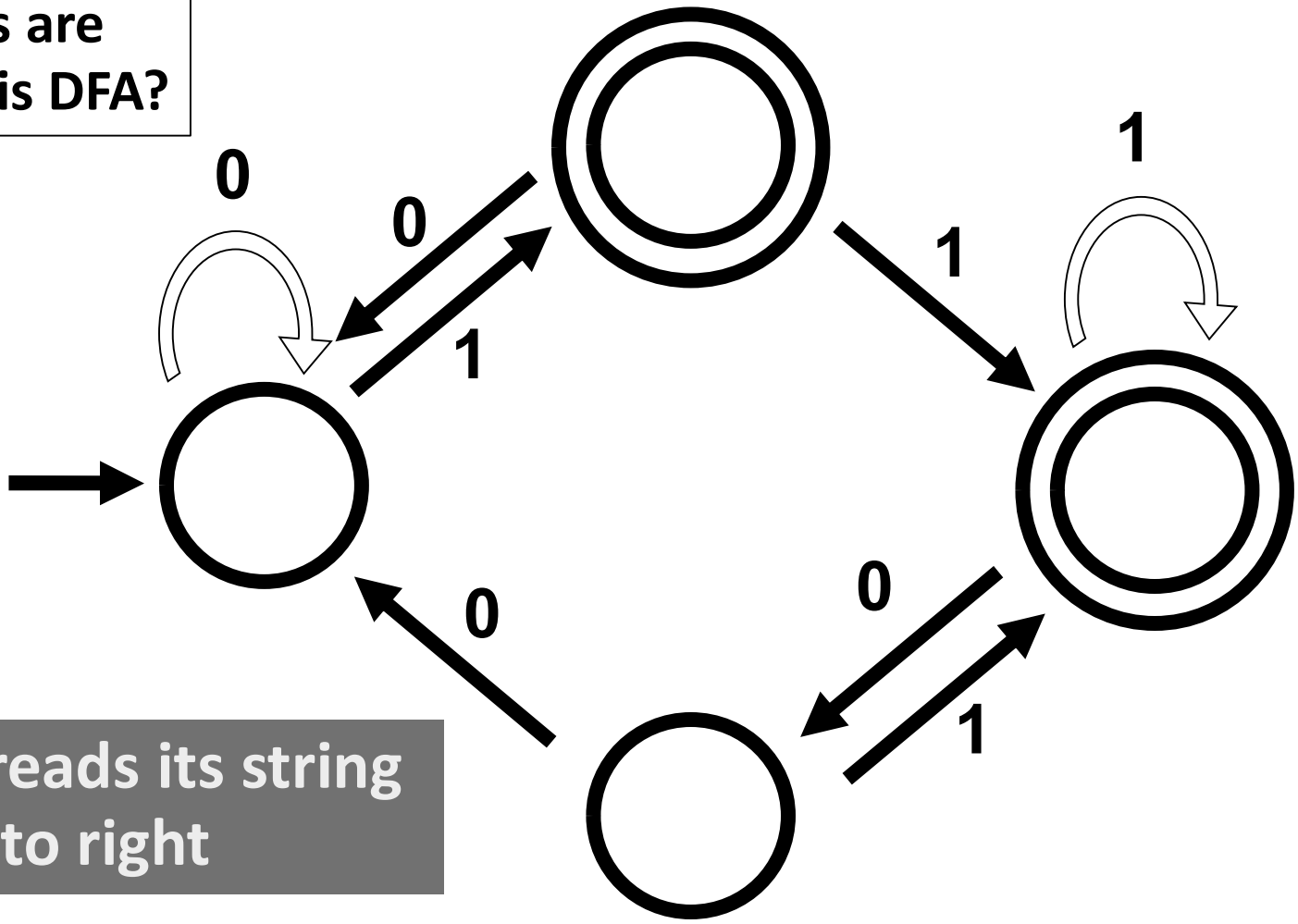
directed graph, possibly with self-loops



The automaton accepts the input string if this process ends in a double-circle state

Otherwise, the automaton rejects the string

What strings are accepted by this DFA?



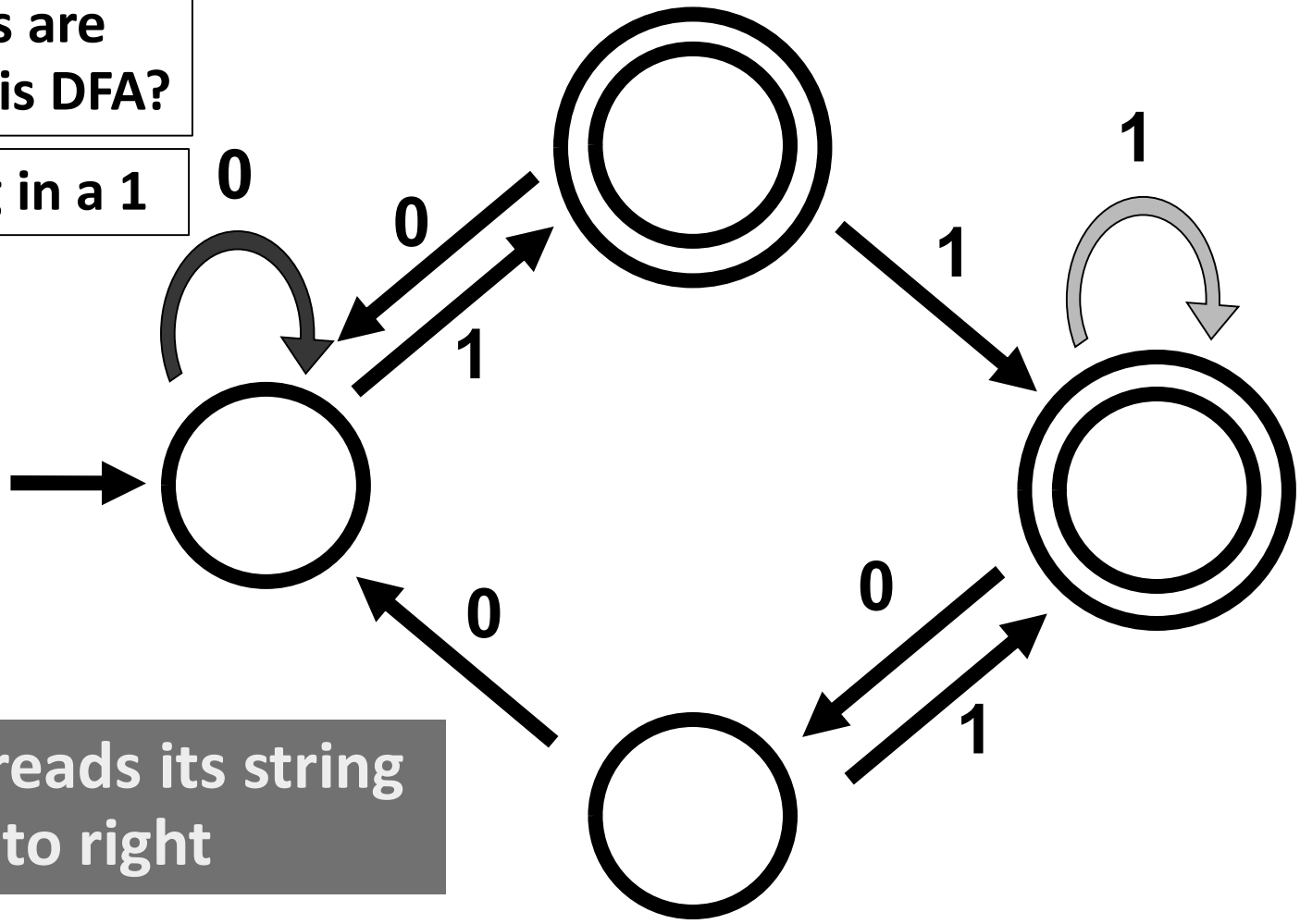
The DFA reads its string from left to right

The automaton accepts the input string if this process ends in a double-circle state

Otherwise, the automaton rejects the string

What strings are accepted by this DFA?

Strings ending in a 1



The DFA reads its string from left to right

The automaton accepts the input string if this process ends in a double-circle state

Otherwise, the automaton rejects the string

Let's make this more formal…

An alphabet Σ is a finite set (e.g., Σ = {0,1})

A string over Σ is a finite sequence of elements of Σ

Σ* = the set of all strings over Σ

For a string x, |x| is the length of x
(number of symbols in x)

The unique string of length 0 is denoted by ε
and is called the empty string

A language over Σ is a set of strings over Σ
In other words: **a language is a subset of Σ***

Languages = Problems

A language over Σ is a set of strings over Σ
In other words: **a language is a subset of Σ***

Problem: Given a string x, is x in the language?

Languages $\equiv$ ***Functions that take a string as input, and output a single bit***

Thm: Every language L over **Σ** uniquely corresponds to a **function $f : \Sigma^* \rightarrow \{0,1\}$.**

Proof Idea: Given L, define *f* such that:

$f(x) = 1$ **if** $x \in L$

$= 0$ **otherwise**

Languages = Problems

A language over Σ is a set of strings over Σ
In other words: **a language is a subset of Σ***

Problem: Given a string x, is x in the language?

Languages $\equiv$ ***Functions that take a string as input, and output a single bit***

Thm: Every language L over **Σ** uniquely corresponds to a **function $f : \Sigma^* \rightarrow \{0,1\}$.**

Proof Idea: Given *f*, define **$L = \{x \mid f(x) = 1\}$**

Definition. A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Q is the set of states (finite)

Σ is the alphabet (finite)

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

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept/final states

Let $w_1, \ldots, w_n \in \Sigma$ and $w = w_1 \cdots w_n \in \Sigma^*$

M accepts w if there are $r_0, r_1, \ldots, r_n \in Q$, s.t.

- $r_0 = q_0$
- $\delta(r_{i-1}, w_i) = r_i$ for all i = 1, ..., n, and
- $r_n \in F$

M rejects w iff M does not accept w

Definition. A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Q is the set of states (finite)

Σ is the alphabet (finite)

$\delta : Q \times \Sigma \to Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept/final states

Let $w_1, \ldots, w_n \in \Sigma$ and $w = w_1 \cdots w_n \in \Sigma^*$
M accepts w if the (unique) path starting from q_0 with edge labels $w_1, \ldots, w_n$ ends in a state in **F**.

M rejects w iff M does not accept w

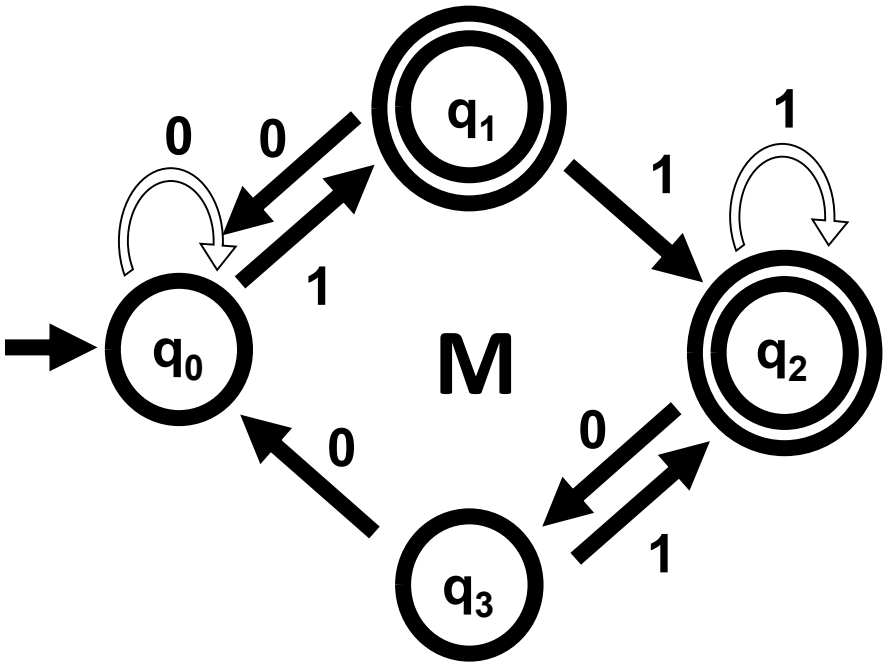
$M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$

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

$\delta : Q \times \Sigma \rightarrow Q$ transition function*

$q_0 \in Q$ is start state

$F = \{q_1, q_2\}$



*

δ	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_3	q_2
q_3	q_0	q_2

A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Q is the set of states (finite)

Σ is the alphabet (finite)

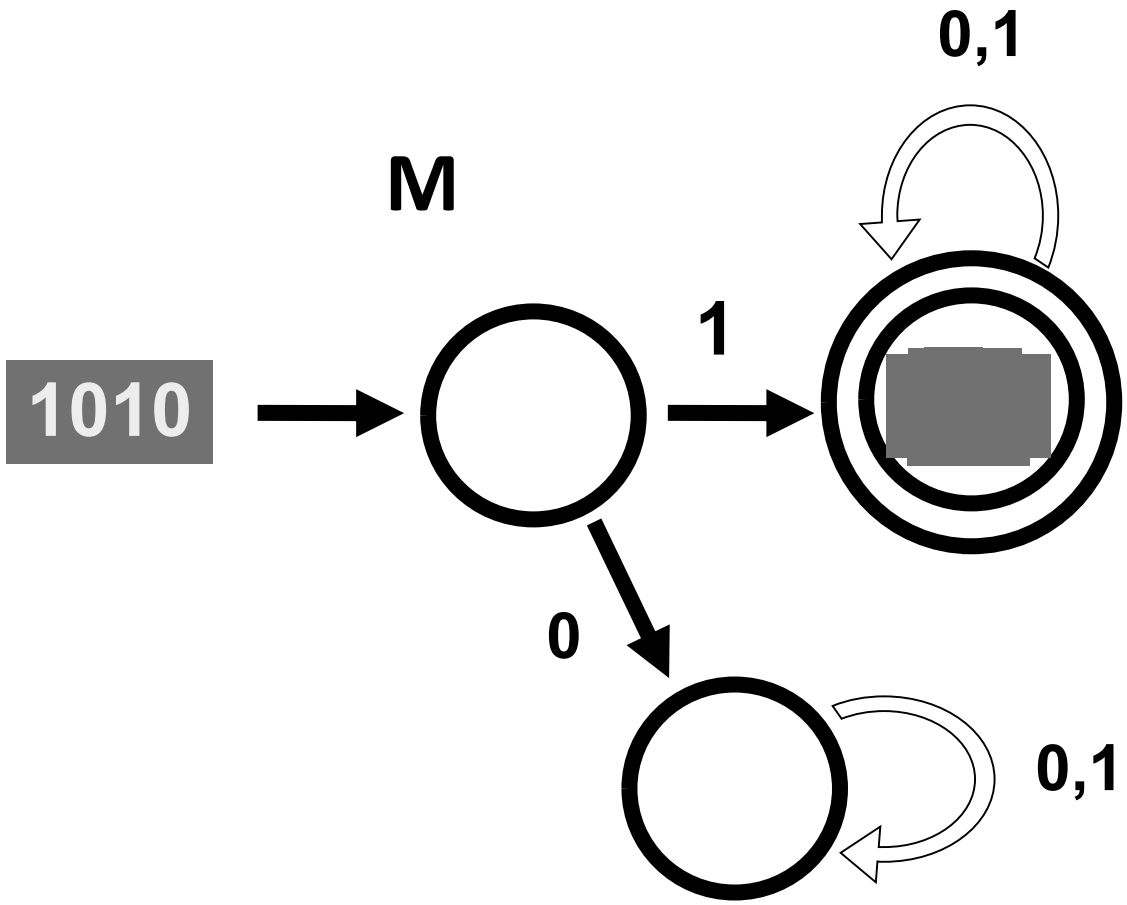
$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept/final states

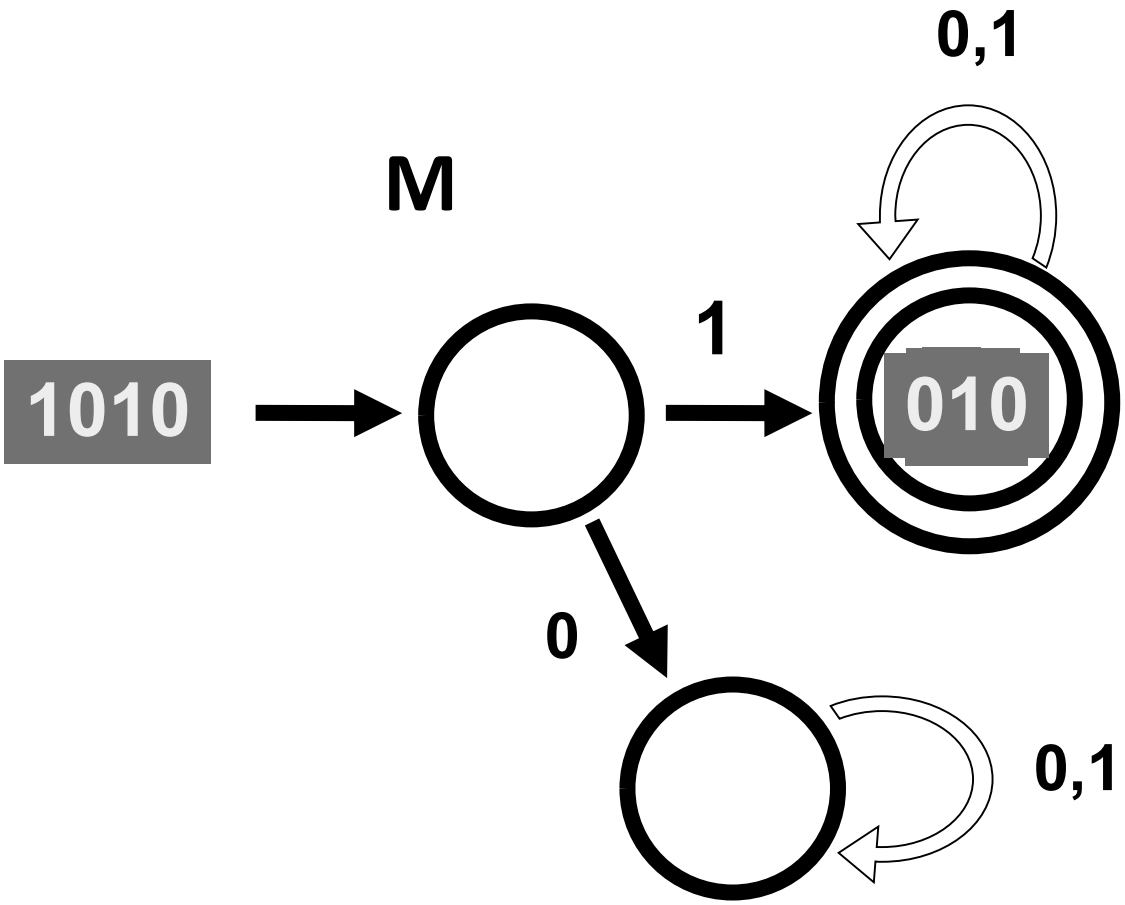
The problem "solved" by the DFA M is:

L(M) = set of all strings that M accepts
= "the language recognized by M"
$\equiv$ the *function computed by* M

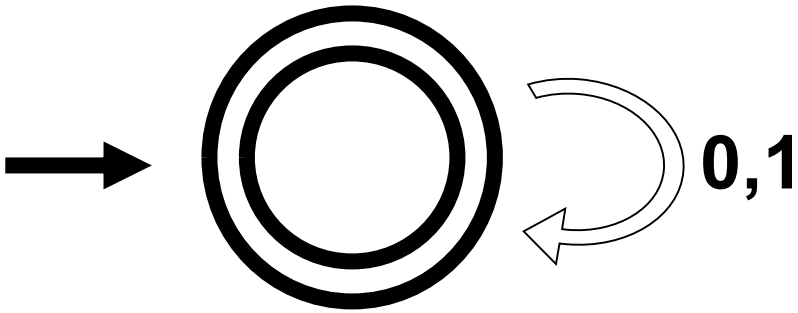


L(M) = { w | w begins with 1}

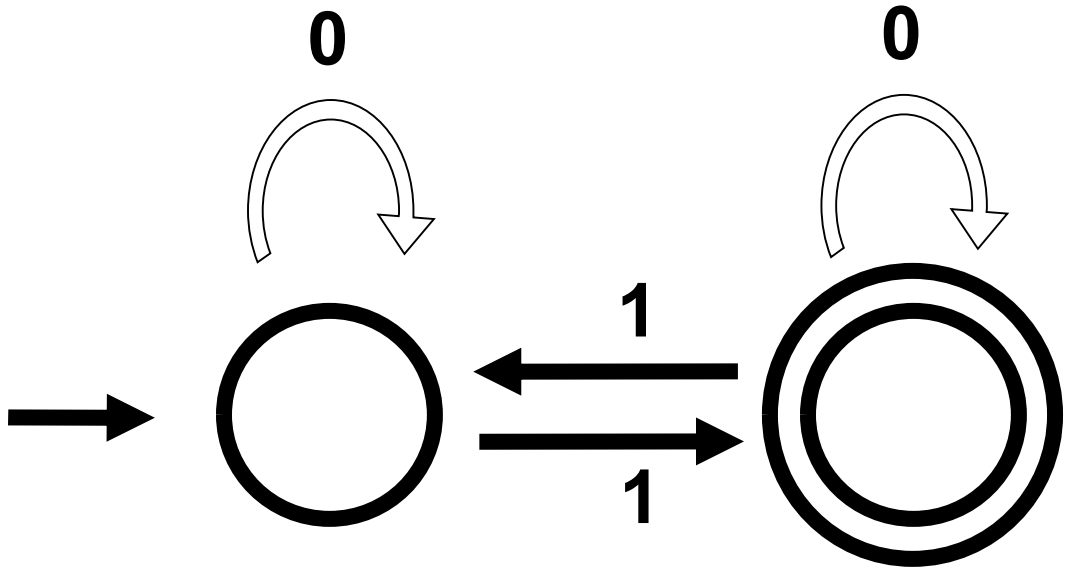
Suppose the above machine read strings from *right* to *left*…
What language would be recognized then?



L(M) = { w | w begins with 1}



$L(M) = \{0,1\}^*$



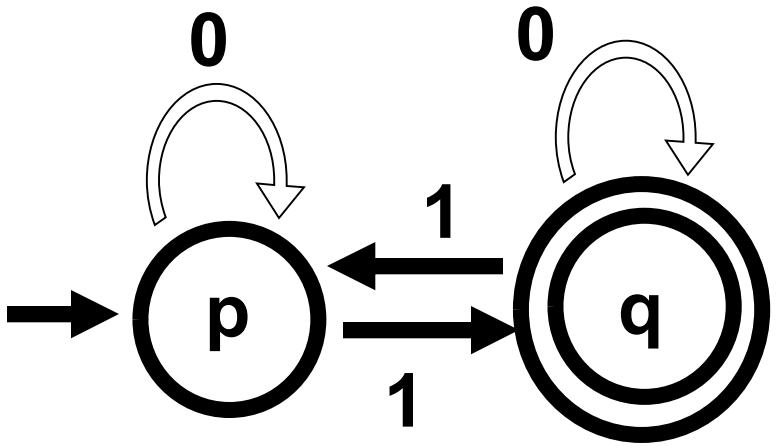
L(M) = { w | w has an odd number of 1s}

How would you *prove* this?

$$M = (\{p,q\}, \{0,1\}, \delta, p, \{q\})$$

(Q, Σ, q_0, F)

δ	0	1
p	p	q
q	q	p



L = {w | w has odd number of 1s }

Theorem: L(M) = L

Proof: By induction on **n**, the length of a string.

Base Case n=0: $\varepsilon \notin L$ and $\varepsilon \notin L(M)$

Induction Hypothesis: Suppose for all $\mathbf{w} \in \Sigma^*$, $|\mathbf{w}| = \mathbf{n}$,

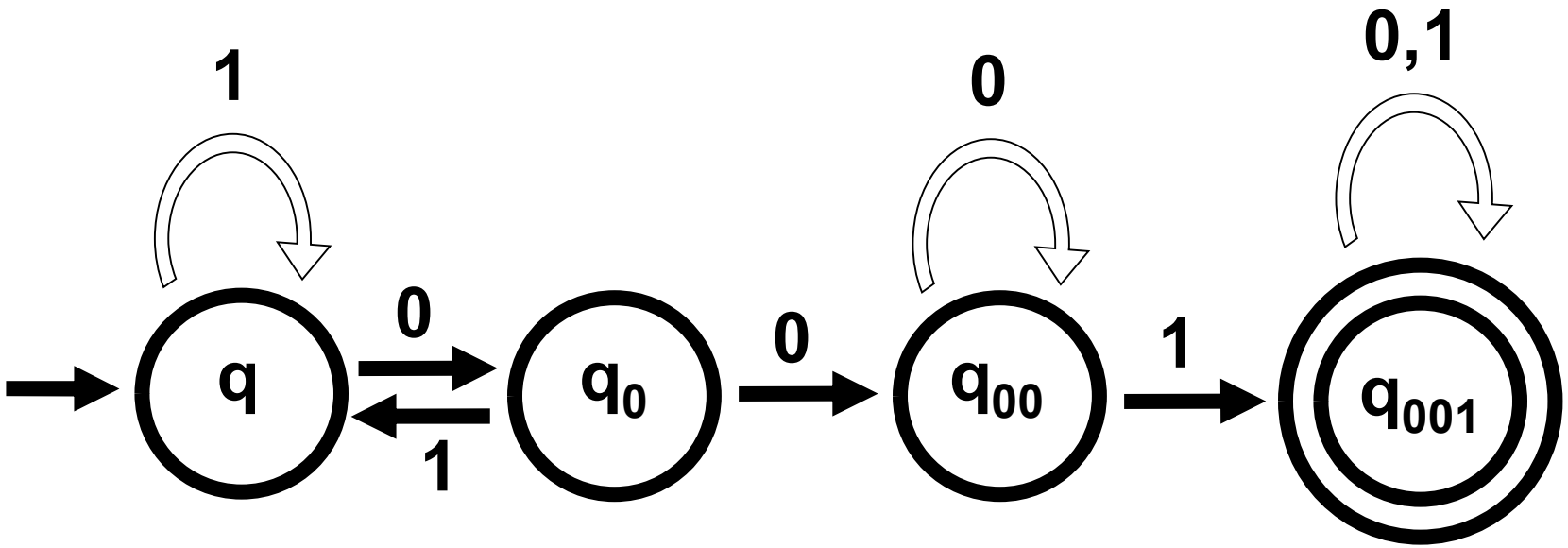
M accepts w $\Leftrightarrow$ **w** has odd number of 1s

Every string of length **n+1** has the form **w0** or **w1, |w|=n**

Show that after reading w0 or w1, M correctly accepts/rejects. Use Induction Hypothesis!

<your case analysis goes here…>

Build a DFA that accepts exactly the strings containing 001



Can we use fewer states? **No! But why…?**

The Problems Solved by DFAs

Definition: A language **L'** is ***regular*** if
L' is recognized by a DFA;
that is, there is a DFA M where **L'** = L(M).

L' = { w | w contains 001} is regular

L' = { w | w begins with a 1} is regular

L' = { w | w has an odd number of 1s} is regular