6.045

Automata, Computability, and Complexity

csail.mit.edu/~rrw/6.045-2020

INSTRUCTORS & TAs

Ryan Williams

Brynmor Chapman

Dylan McKay

Recitations and Office Hours

Recitations on Fridays

Dylan: 11am-noon (**66-144**)

Brynmor: 1pm-2pm (**4-153**)

You're not required to attend recitations…

But it is *strongly* recommended

Attending lectures is also strongly recommended!

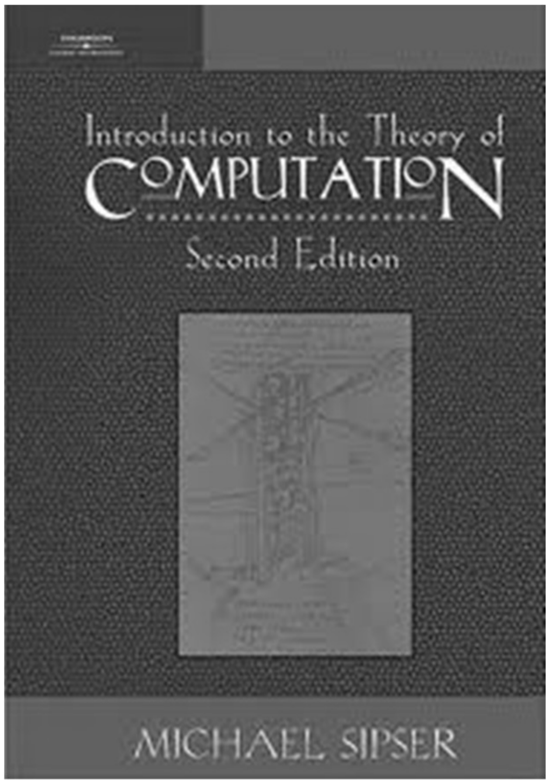
Office Hours (tentative):

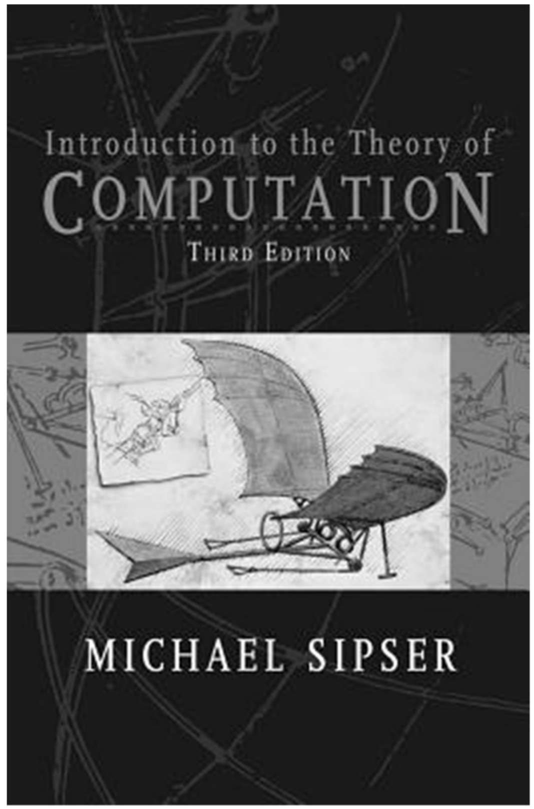
Brynmor: Monday 4pm-6pm (**Stata, G5 Lounge**)

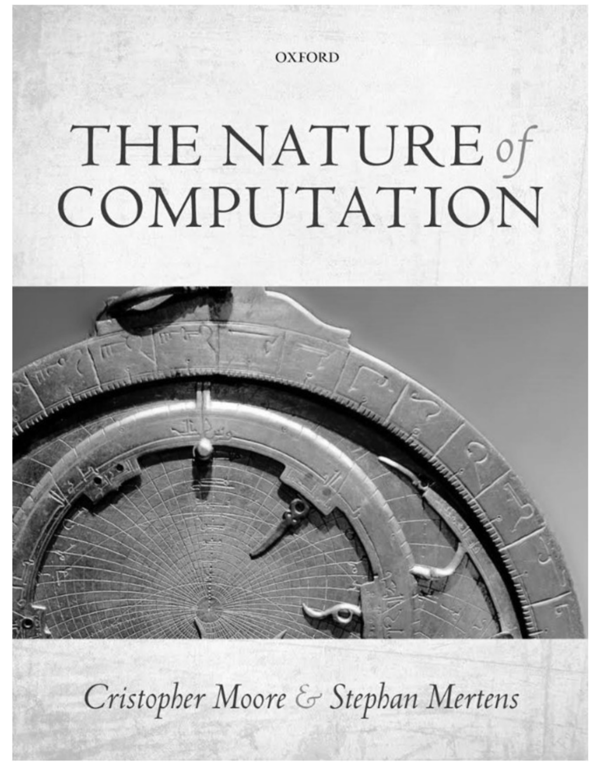
Dylan: Tuesday 4pm-6pm (**Stata, G5 Lounge**)

Ryan: Wednesday 10:30am-12:30, **32-G638**

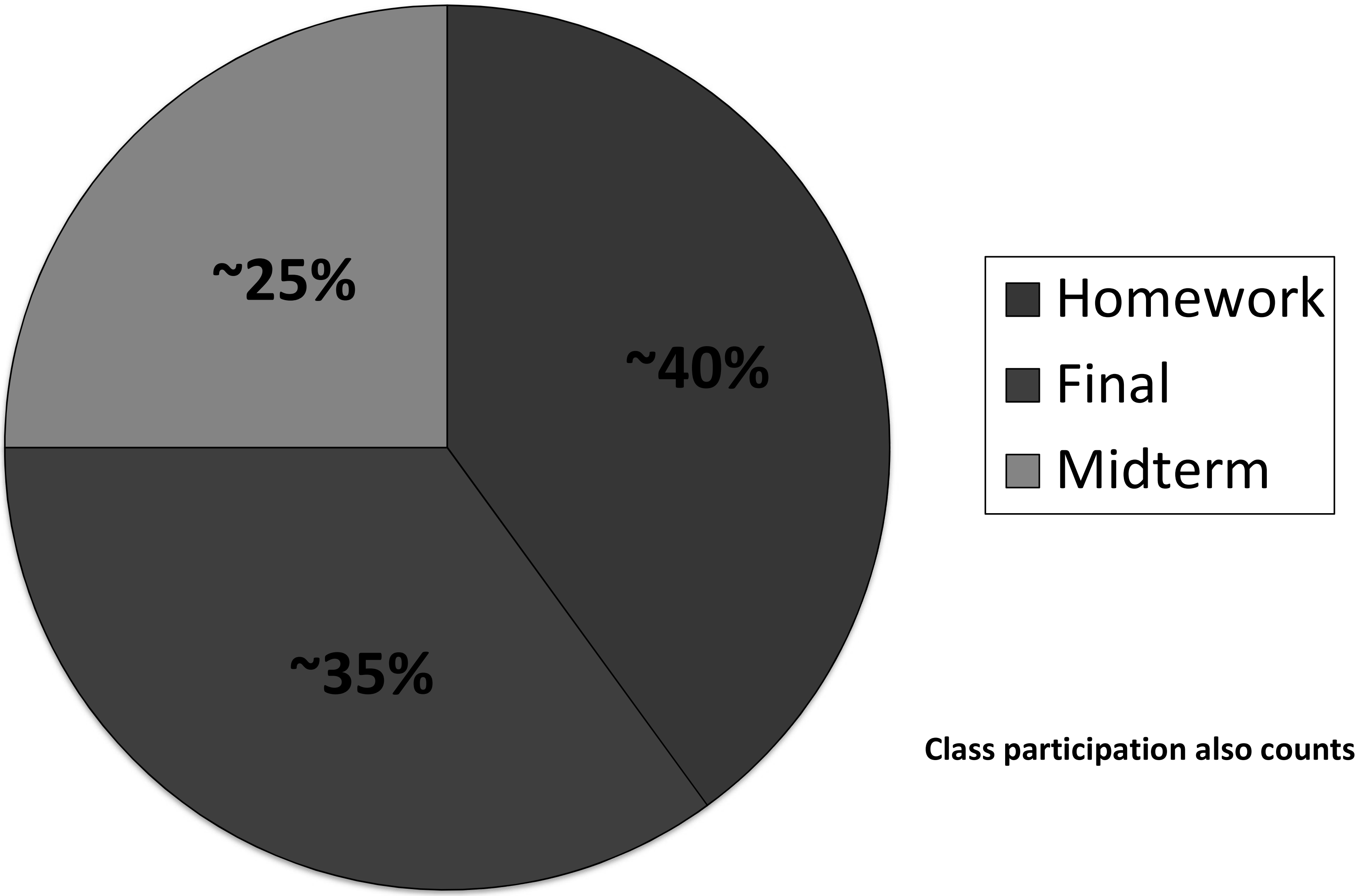
Textbook(s)


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


~25%
~40%
~35%
Homework
Final
Midterm
Class participation also counts

Homework / Problem Sets / Psets / Pests

Homework will be assigned on most Wednesdays and will be due one week later, 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)

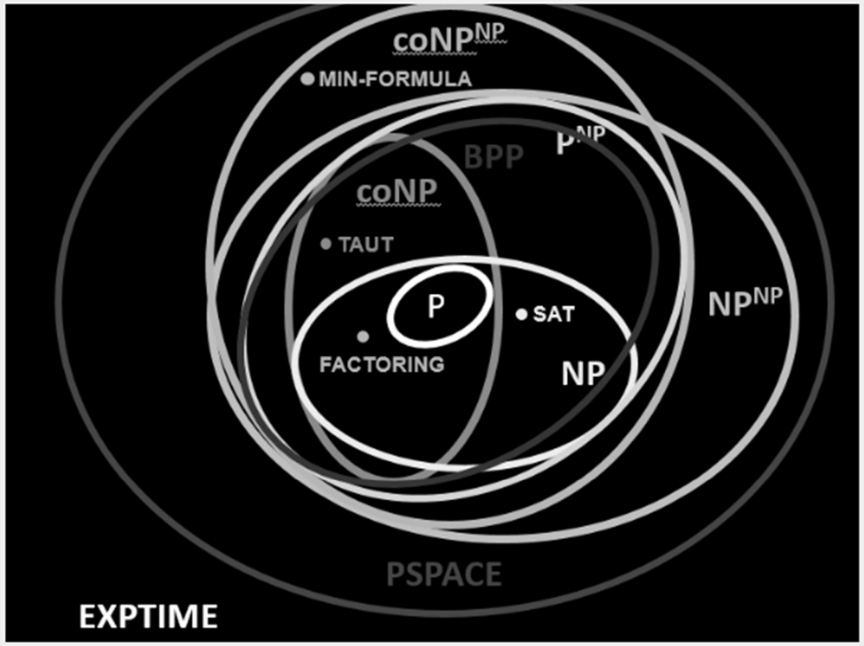
We will provide LaTeX source code for every homework assignment – fill it in with your answers!

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.

6.045 - Automata, Computability, and Complexity Theory - Spring 2020



[General Info] [Lectures] [Homeworks] [Exams]

Announcements on Piazza

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!

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 azza ×

piazza.com/mit/spring2020/6045/home

6.045 15 Q & A **Resources** Statistics Manage Class

Massachusetts Institute of Technology (MIT) - Spring 2020

6.045: Automata, Computability, & Complexity

+ 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

Webpage

https://people.csail.mit.edu/rrw/6.045-2020/index.html

Announcements

+ Add

Add an Announcement

Click the Add button to add an announcement.

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

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 under an hour 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

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.

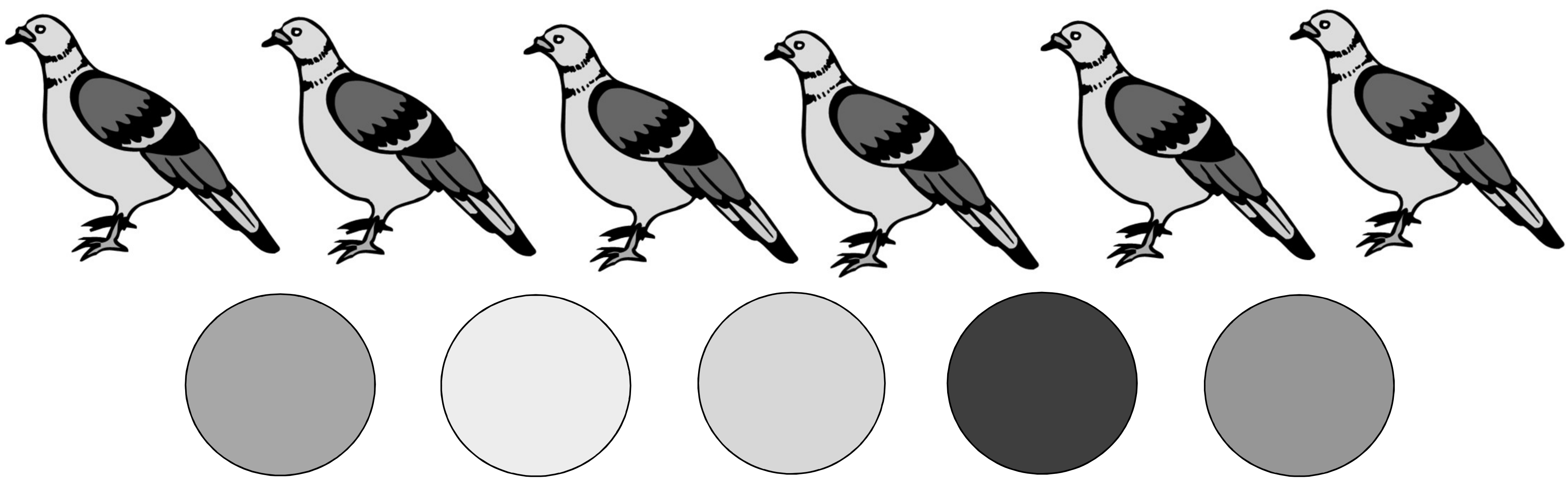
If 1 isn’t in A, then A = {2,3,4}, and 2 divides 4

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

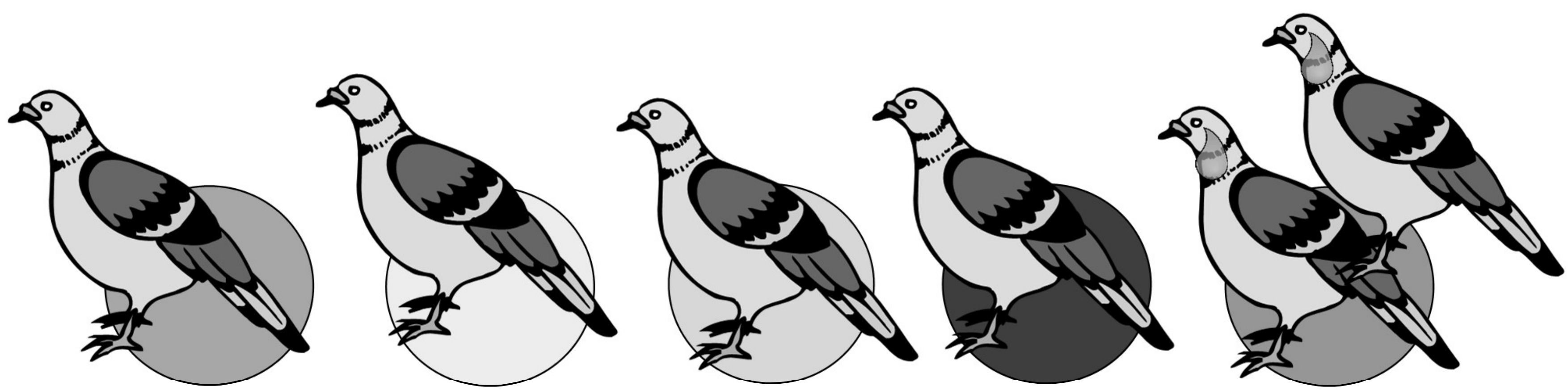


LEVEL 1

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 $a = 2^k m$, 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 $a_1 = 2^i m$ and $a_2 = 2^j m$
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 $a = 2^k m$ where m is an odd number in $\{1, \ldots, 2n\}$

Note: There are n odd numbers in $\{1, \ldots, 2n\}$

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 $a_1 = 2^i m$ and $a_2 = 2^j m$ where $i < j$, so a_1 divides a_2. QED

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

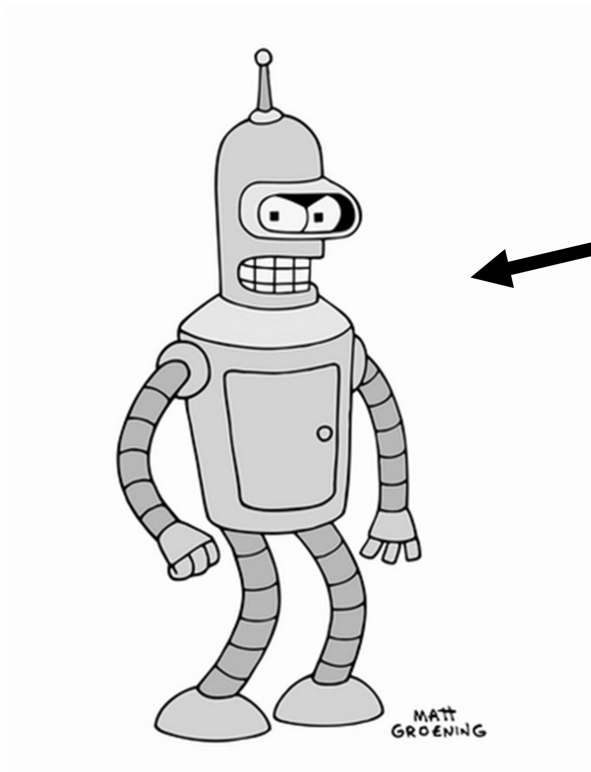
During lectures, my proofs will generally contain the first two levels, but only part of the third (TAs will guide you through some “third levels”)

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!

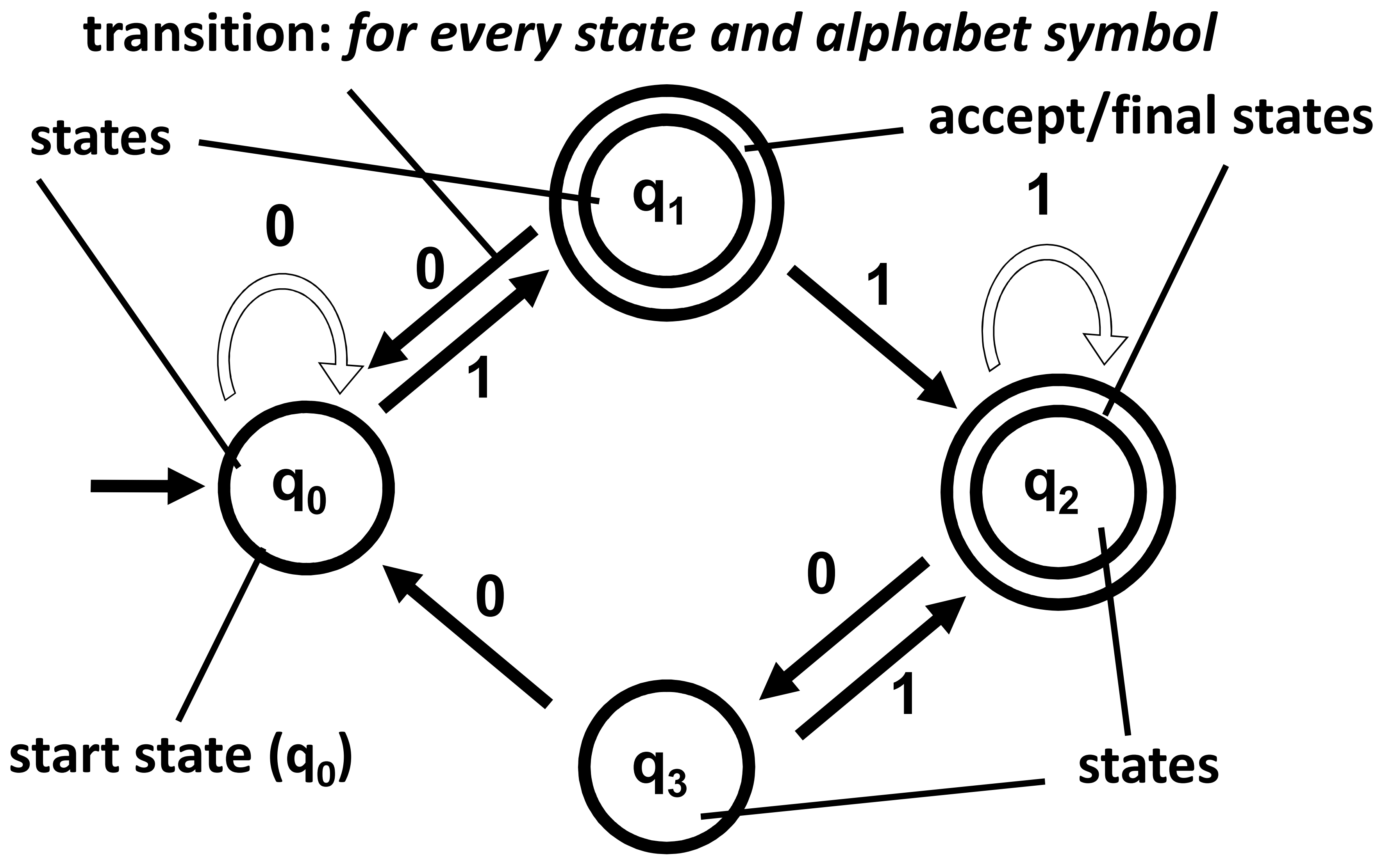
Come by office hours or ask (privately) on piazza if you worry about your level of detail in a proof!

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

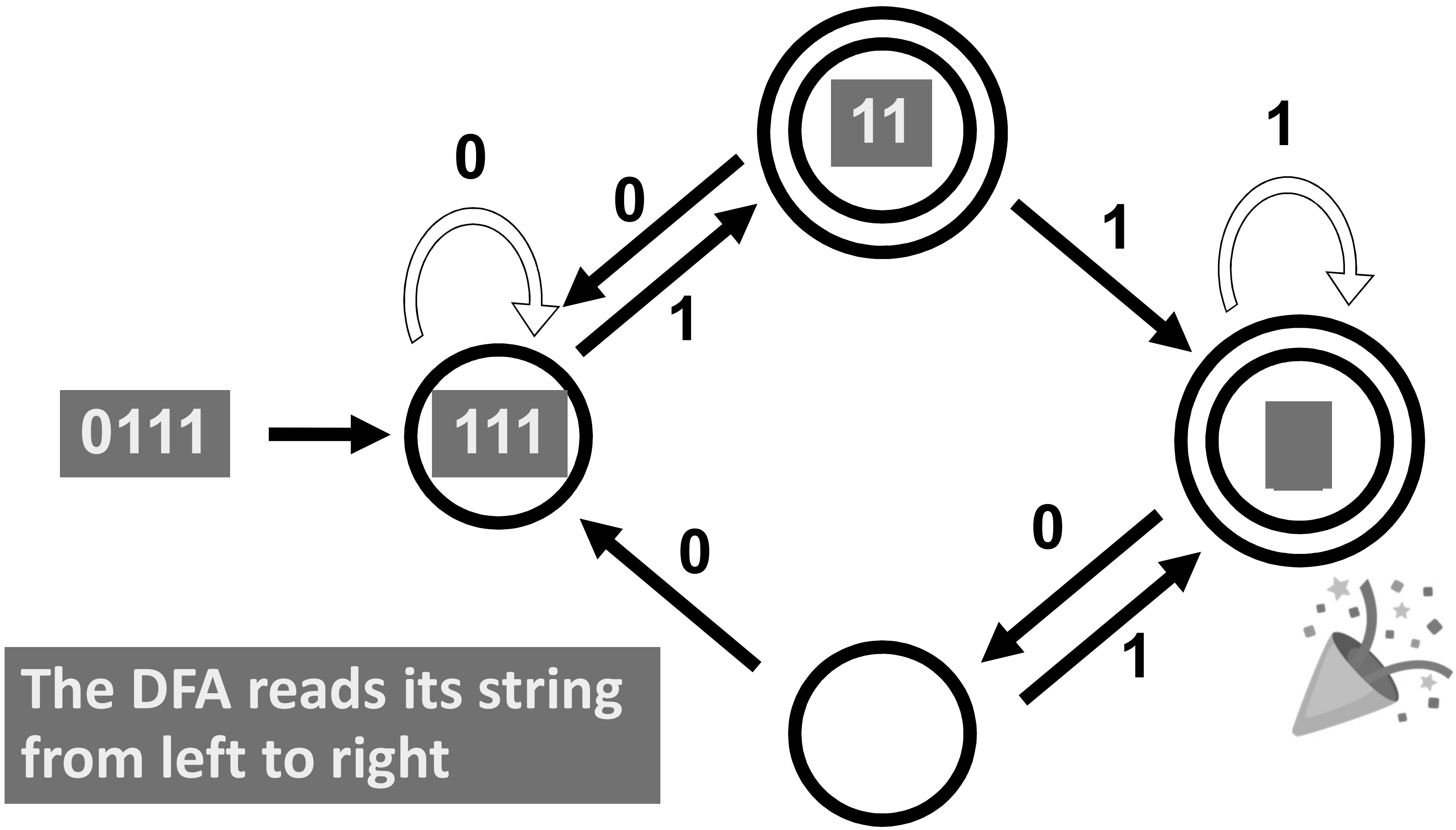


(not a DFA)

Anatomy of Deterministic Finite Automata



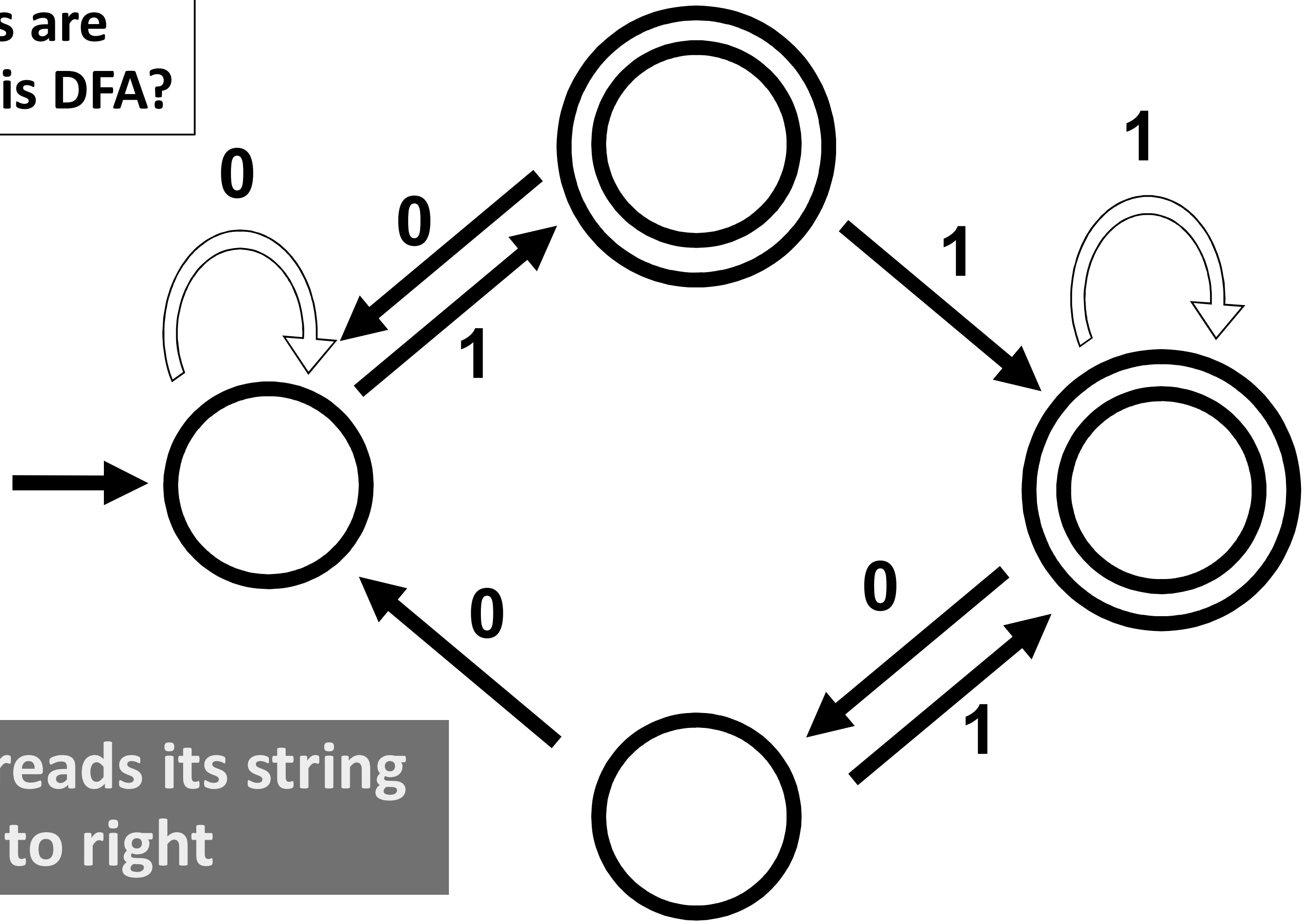
directed graph, possibly with self-loops



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?



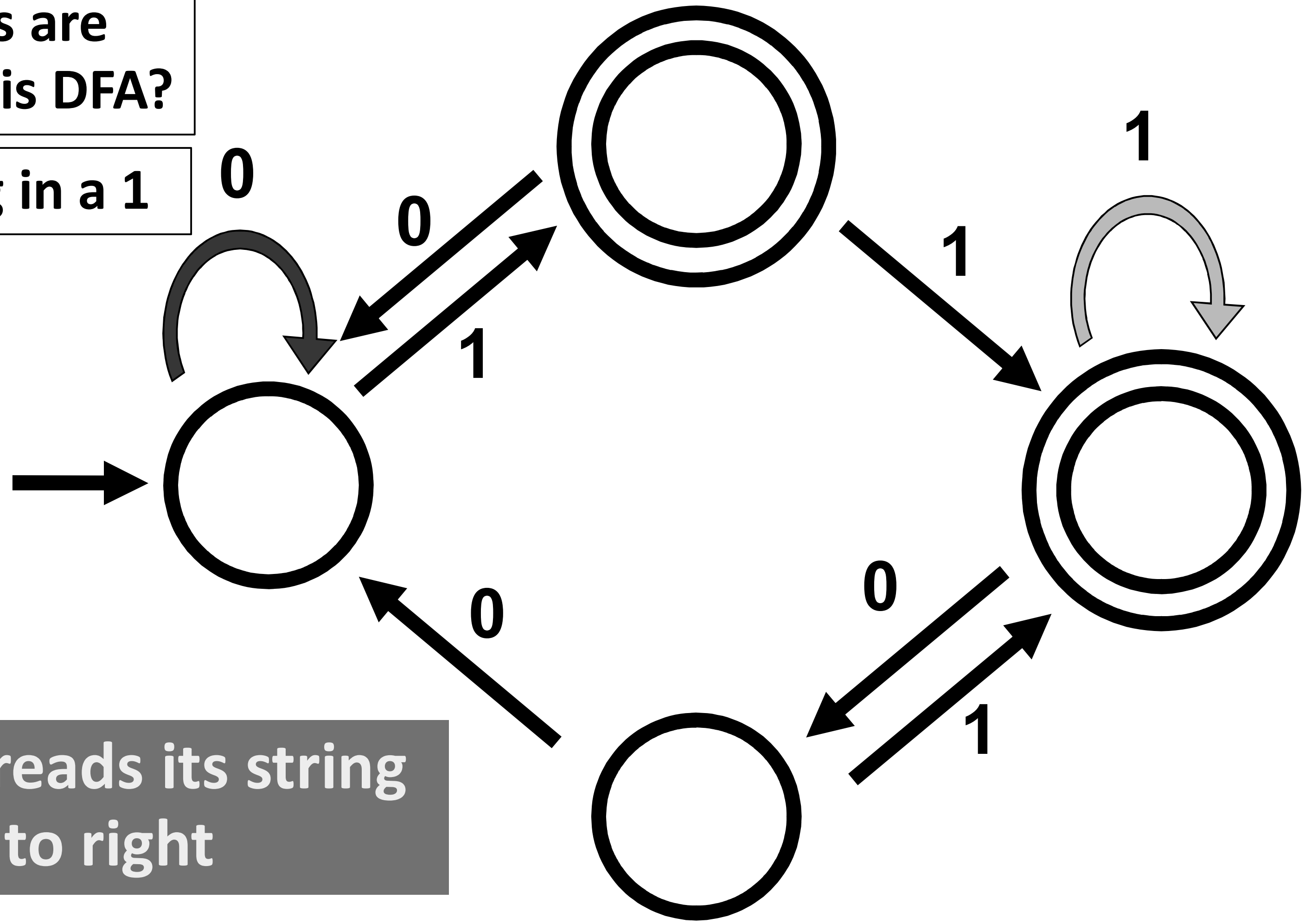
The DFA reads its string from left to right

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

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 letters 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** $f : \Sigma^* \to \{0,1\}$.

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

$$f(x) = 1 \text{ if } x \in L$$
$$= 0 \text{ otherwise}$$

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* : Σ* → {0,1}.**

Proof Idea: Given *f*, define **L = {x | *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 $w_1, \ldots, w_n \in \mathbf{\Sigma}$ and $\mathbf{w} = w_1 \cdots w_n \in \mathbf{\Sigma^*}$

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

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

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

M rejects w iff M does not accept w

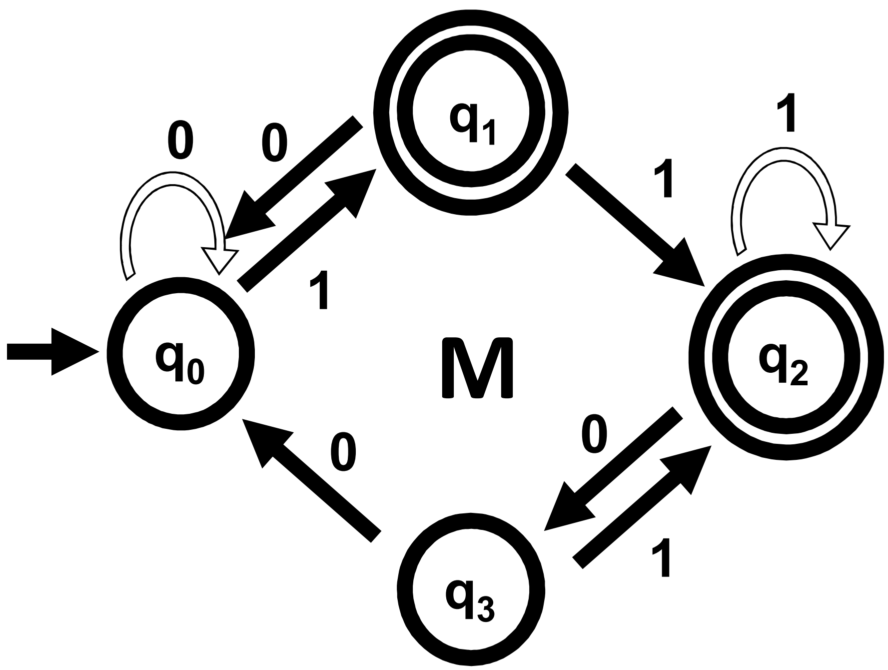
$M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$

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

$\delta : Q \times \Sigma \rightarrow Q$ 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)

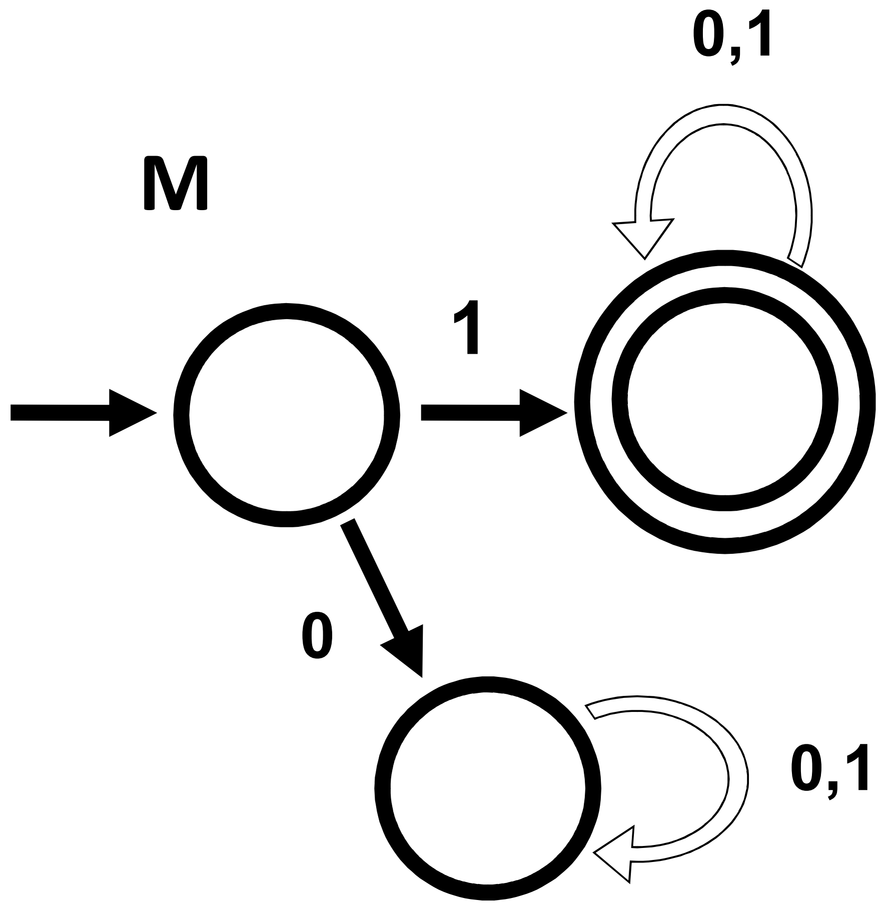
$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

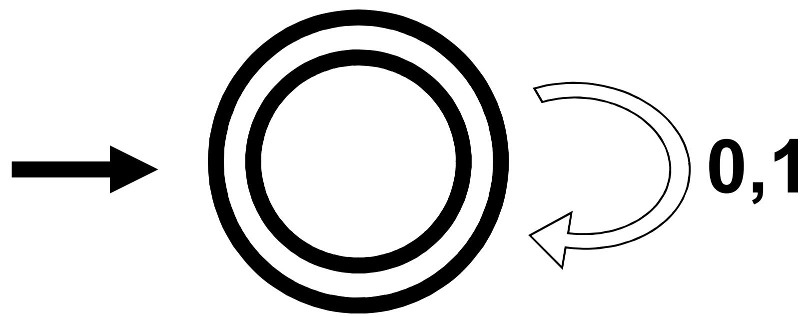
$F \subseteq Q$ is the set of accept/final states

The problem "solved" by the DFA M is:

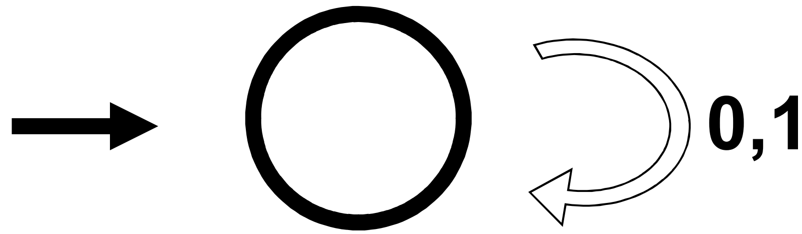
L(M)	= set of all strings that M accepts
	= "the language recognized by M"
	$\equiv$ the *function computed by* M



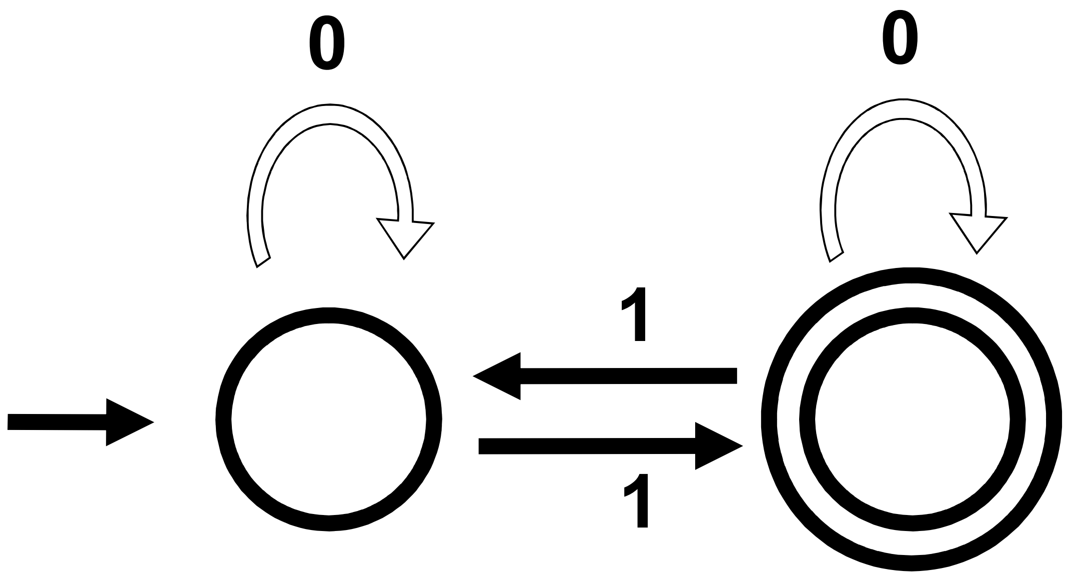
L(M) = { w | w begins with 1}



L(M) = {0,1}*



L(M) = ∅

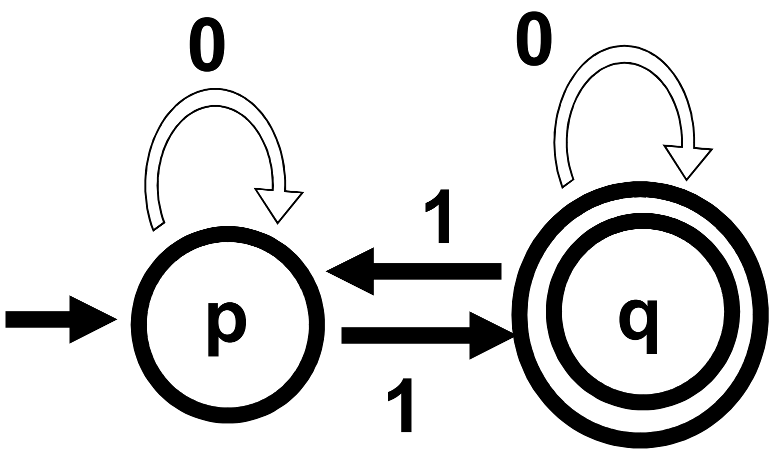


L(M) = { w | w has an odd number of 1s}

How would you *prove* this?

$$\overset{Q}{M = (\{p,q\}},\ \overset{\Sigma}{\{0,1\}},\ \delta,\ \overset{q_0}{p},\ \overset{F}{\{q\}})$$

δ	0	1
p	p	q
q	q	p



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

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

Base Case n=0: ε ∉ L and ε ∉ L(M)

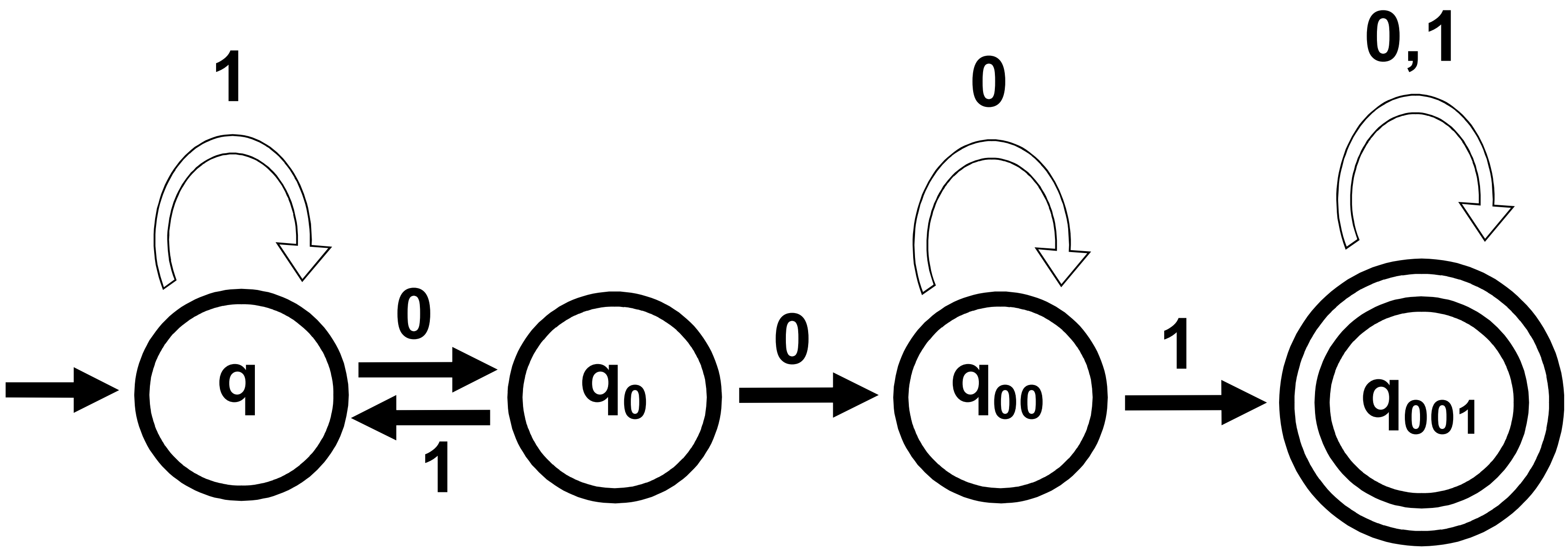
Induction Hypothesis: Suppose for all **w** ∈ **Σ***, **|w| = n**,
M accepts w ⇔ **w** has odd number of 1s

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

***IBM JOURNAL* APRIL 1959**

Turing Award **winning paper**

M. O. Rabin*

D. Scott†

Finite Automata and Their Decision Problems‡

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang[1] and by E. F. Moore,[4] and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape"

ıe construction of $\mathfrak{A}$ and we shall
:tail.

ces of words $S_1=(a_1,a_2, \ldots, a_n)$
b_n) then $P(a_1,a_2, \ldots, a_n) \cap P(b_1,$
d only if the Post correspondence
$_2$ has a solution. Since the corre-
not effectively solvable it follows
ther

$T_2(\mathfrak{A}(b_1, \ldots, b_n)) \neq \phi$

ble.

tape automata

-way, two-tape automata we find
constructive decision processes is
sible to decide, by a constructive
icable to all automata, whether a
chine accepts any tapes. To prove
ourse, necessary to give the explicit
y machine. We shall not give the
are long and not very much dif-
ıl definitions needed for two-way
ıe main point is that, as with the
omaton, the table of moves of a
ıtomaton sometimes requires the
om the scanned square. However,
should clarify the method.
that there is no constructive deci-

Theorem 19. *There is no effective method of deciding whether the set of tapes definable by a two-tape, two-way automaton is empty or not.*

An argument similar to the above one will show that the class of sets of pairs of tapes definable by two-way, two-tape automata is closed under Boolean operations. In view of Theorem 17, this implies that there are sets definable by two-way automata which are not definable by any one-way automaton; thus no analogue to Theorem 15 holds.

References

1. A. W. Burks and Hao Wang, "The logic of automata," *Journal of the Association for Computing Machinery,* **4,** 193-218 and 279-297 (1957).
2. S. C. Kleene, "Representation of events in nerve nets and finite automata," *Automata Studies,* Princeton, pp. 3-41, (1956).
3. W. S. McCulloch and E. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bulletin of Mathematical Biophysics,* **5,** 115-133 (1943).
4. E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies,* Princeton, pp. 129-153 (1956).
5. A. Nerode, "Linear automaton transformations," *Proceedings of the American Mathematical Society,* **9,** 541-544 (1958).
6. E. Post, "A variant of a recursively unsolvable problem," *Bulletin of the American Mathematical Society,* **52,** 264-268 (1946).
7. J. C. Shepherdson, "The reduction of two-way automata to one-way automata," *IBM Journal,* **3,** 198-200 (1959).

Revised manuscript received August 8, 1958

Union Theorem for Regular Languages

Given two languages L_1 and L_2
recall that the union of L_1 and L_2 is

$$L_1 \cup L_2 = \{ w \mid w \in L_1 \text{ or } w \in L_2 \}$$

Theorem: The union of two regular languages is also a regular language

Given two DFAs M and N,
there is a DFA M' that accepts x
$\Leftrightarrow$ At least one of M or N accepts x

Given two languages that we know to be regular, how can we make new languages out of them?

Theorem: The union of two regular languages is also a regular language

Proof: Let

$M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ **be a finite automaton for** L_1

and

$M_2 = (Q_2, \Sigma, \delta_2, q'_0, F_2)$ **be a finite automaton for** L_2

We want to construct a finite automaton
$M = (Q, \Sigma, \delta, p_0, F)$ **that recognizes** $L = L_1 \cup L_2$

Proof Idea: Run both M_1 and M_2 "in parallel"!

$M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ recognizes L_1 and
$M_2 = (Q_2, \Sigma, \delta_2, q'_0, F_2)$ recognizes L_2

Q = pairs of states, one from M_1 and one from M_2

$= \{ (q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \}$

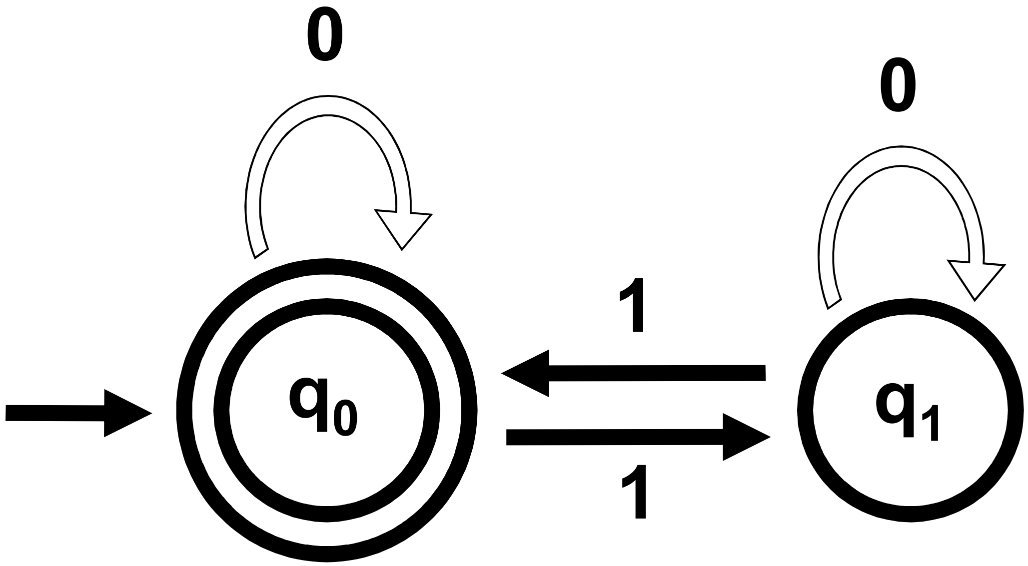
$= Q_1 \times Q_2$

$p_0 = (q_0, q'_0)$

$F = \{ (q_1, q_2) \mid q_1 \in F_1 \text{ OR } q_2 \in F_2 \}$

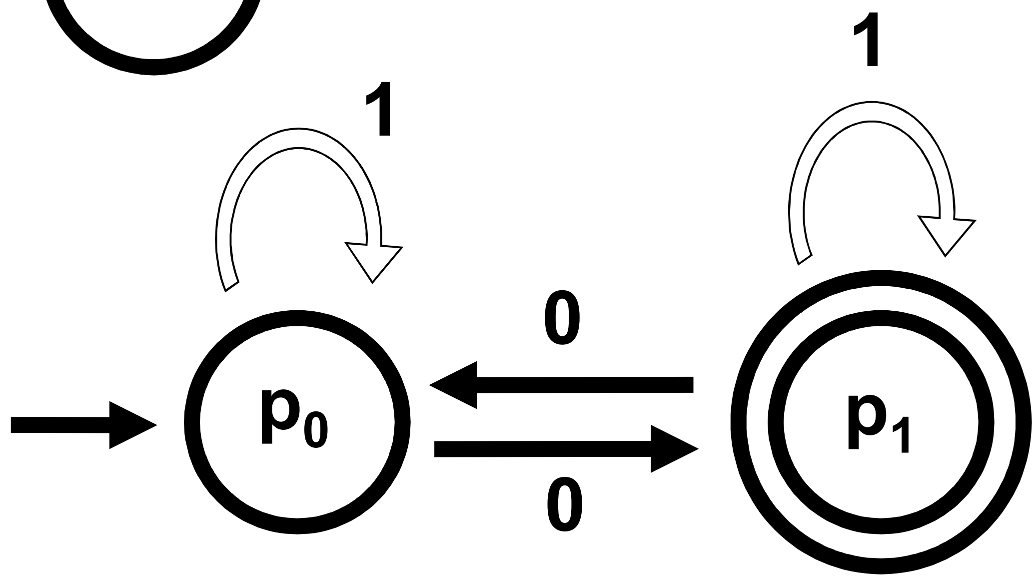
$\delta((q_1,q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$

Theorem: The union of two regular languages is also a regular language



Even number of 1's

Odd number of 0's



Even number of 1's OR Odd number of 0's