CS 154

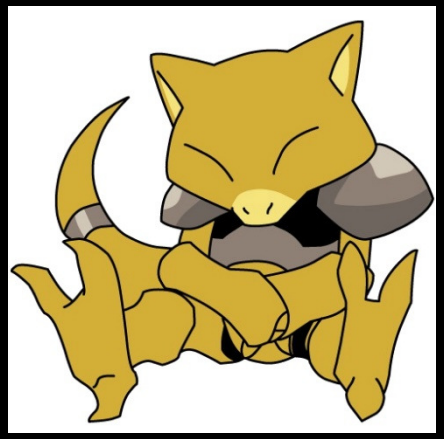
Introduction to Automata and Complexity Theory

cs154.stanford.edu

INSTRUCTORS & TAs

Ryan Williams

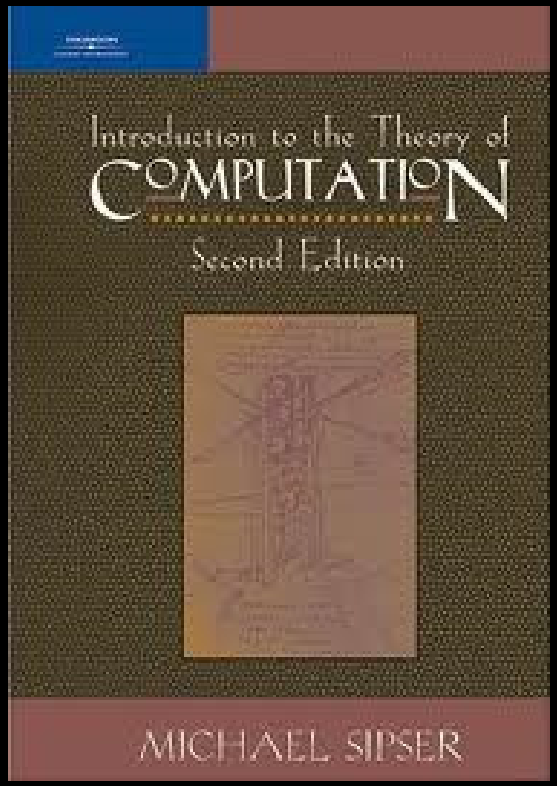
Brynmor Chapman

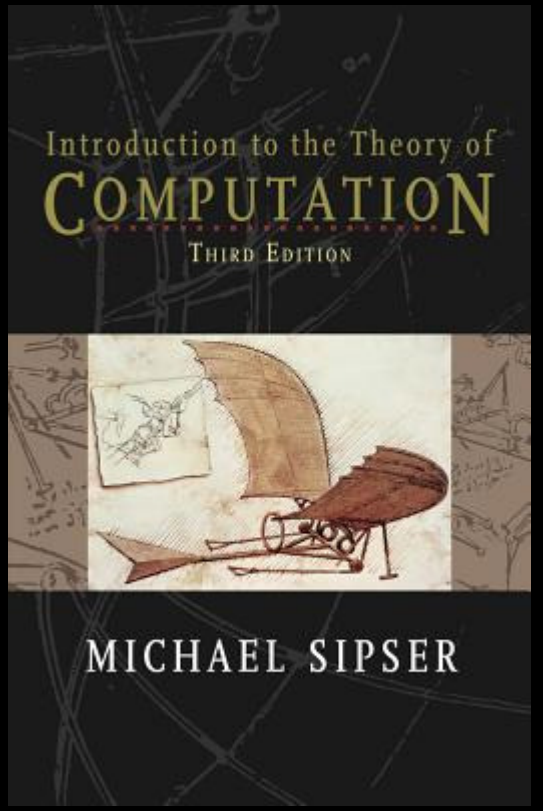


Mikaela Grace

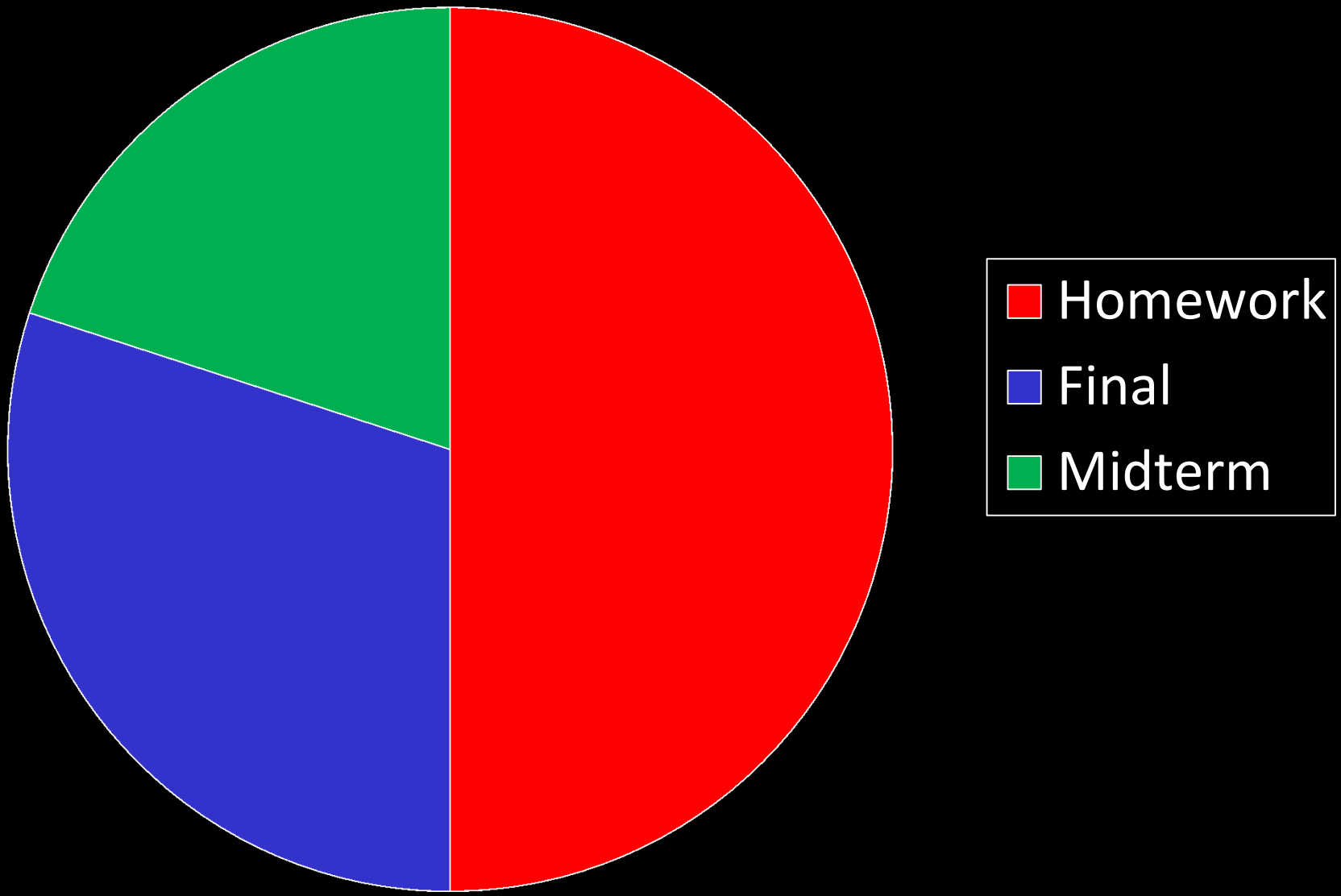
Kevin Lewi

Textbook


Introduction to the Theory of
COMPUTATION
Second Edition
MICHAEL SIPSER


Introduction to the Theory of
COMPUTATION
THIRD EDITION
MICHAEL SIPSER

Grades



Homework / Problem Sets

Homework will be assigned every Tuesday (except for the week before the midterm) and will be due one week later at the beginning of class

You may collaborate with others, but you must:

- Try to solve all the problems by yourself *first*
- List your collaborators on each problem
- ***Write your own solutions***, and write the solution to each problem on a separate sheet of paper – see website
- If you receive a significant idea from somewhere, you must acknowledge that source in your solution.

We will drop your lowest homework grade!

web.stanford.edu/~rrwill/cs154-2015/

CS154: Automata and Complexity Theory, Winter 2015

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

Announcements

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 trends of technology, such as the Church-Turing Thesis and the P versus NP problem. **Prerequisites: CS 103 or 103B.**

General Information

Instructor:

- Ryan Williams, Gates 464, 650 723 6690, *rrw at cs dot stanford dot edu*

TAs:

- Brynmor Chapman, *chapmanb (at stanford dot edu)*
- Mikaela Grace, *mgrace (at stanford dot edu)*
- Kevin Lewi, *klewi (at cs dot stanford dot edu)*

Class: Tuesday-Thursday 12:50-2:05, Gates B3

Office hours:

https://piazza.com/stanford/winter2015/cs154/home

CS 154 ▾ | Q & A | Resources | Statistics | Manage Class

Stanford University - Winter 2015

CS 154: Introduction to Automata 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: CS 103 or 103B.**

General Information

Edit

Course Website

http://cs154.stanford.edu

Announcements

Add an Announcement

Click the Add button to add an announcement.

Thinking of taking another course at the same time?

If the other course:

- Allows you to take our midterm in-class (Ours is February 17)
- Does *not* have its own final exam (Ours is March 19, 7-10pm)

Then we'll allow it.

"Students must not register for classes with conflicting end quarter exams."

This class is about
formal models of computation

What is computation?

What can and cannot be computed?

What can be *efficiently* computed?

Why should you care?

new ways of thinking about computing

theory often drives practice

mathematical models of computation predated computers

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

math is good for you!

timelessness

Course Outline

PART 1

Finite Automata: Very Simple Models

DFAs, NFAs, regular languages, regular expressions, pumping lemma, Myhill-Nerode, computing the *minimum* DFA, streaming algorithms

PART 2

Computability Theory: Very Powerful Models

Turing Machines, decidability, recognizability, reducibility, Rice's theorem, the recursion theorem, oracles, Kolmogorov Complexity

PART 3

Complexity Theory: The Modern Models

time complexity, classes P and NP, NP-completeness, space complexity, PSPACE, PSPACE-completeness, polynomial time with oracles

Course Outline

PART 1

Finite Automata: 1940's – 50's

DFAs, NFAs, regular languages, regular expressions, pumping lemma, Myhill-Nerode, computing the *minimum* DFA, streaming algorithms

PART 2

Computability Theory: 1930's – 50's

Turing Machines, decidability, recognizability, reducibility, Rice's theorem, the recursion theorem, oracles, Kolmogorov Complexity

PART 3

Complexity Theory: 1960's – present

time complexity, classes P and NP, NP-completeness, space complexity, PSPACE, PSPACE-completeness, polynomial time with oracles

CS103 vs CS154

PART 1
Finite Automata

PART 2
Computability Theory

PART 3
Complexity Theory

CS103 gave you a good 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 more depth than 103

This class will emphasize

MATHEMATICAL PROOFS

A good proof should be:

Clear -- easy to understand

Correct

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

- The first level: a short phrase/sentence giving a "hint" of the proof

(e.g. "Proof by contradiction," "Proof by induction," "Follows from the pigeonhole principle")

- The second level: a short, one paragraph description of the main ideas

- The third level: the full proof (and nothing but)

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

Here's an example.

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

TRUE or FALSE?

There are always two numbers in A such that one number divides the other number

TRUE

Example: $A \subseteq \{1, 2, 3, 4\}$

1 divides every number.

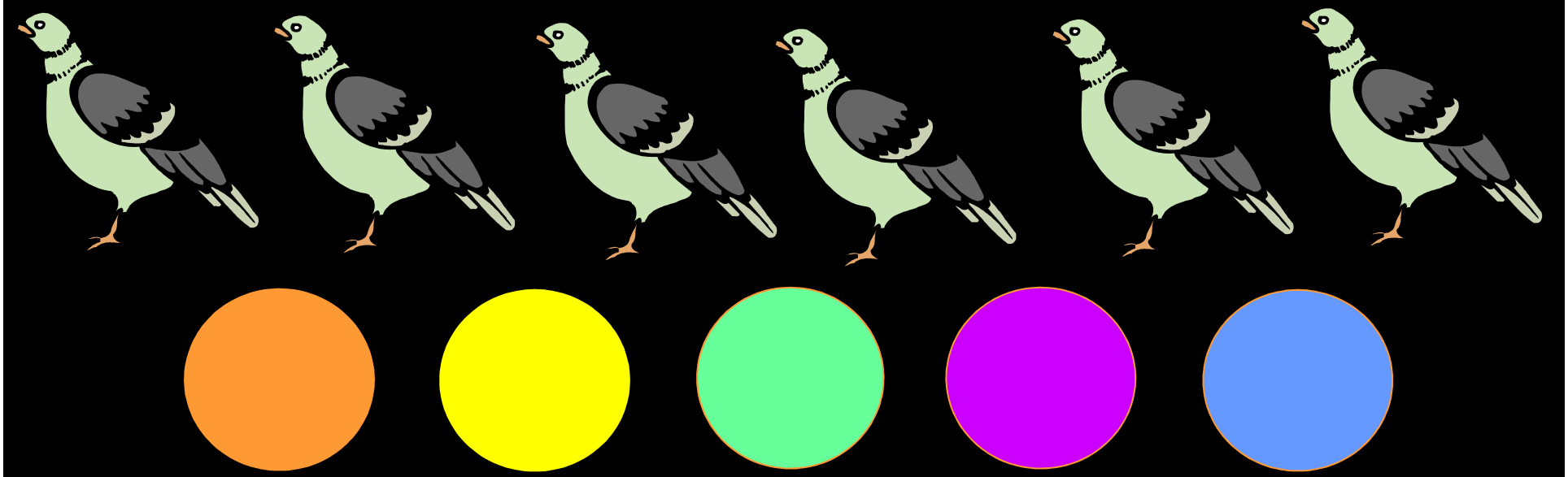
If 1 isn't in A then A = {2,3,4}, and 2 divides 4

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

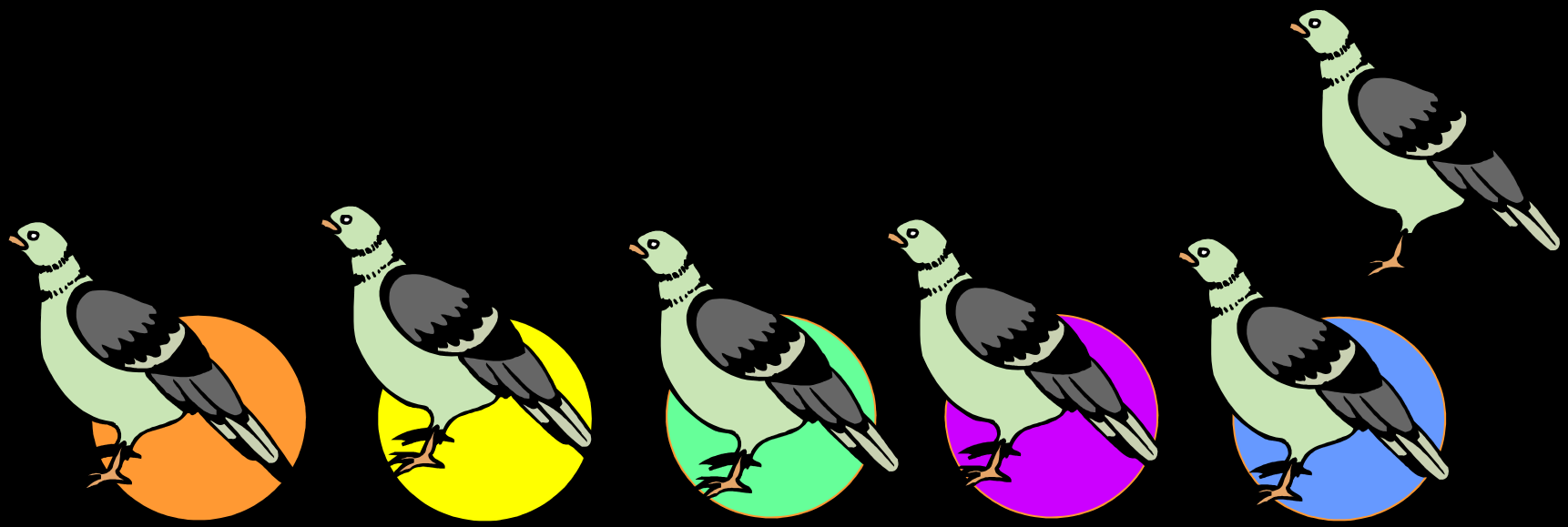


LEVEL 1

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

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

LEVEL 2

Proof Idea:

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

Using the pigeonhole principle,
we'll show there are elements $a_1 \neq a_2$ of A

such that $a_1 = 2^i m$ and $a_2 = 2^k m$
for some odd m and integers i and k

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,..., 2n}

Observe there are n odd numbers in {1, ..., 2n}

Since $|A| = n+1$, there must be two distinct numbers in A with the same odd part

Let a_1 and a_2 have the same odd part m.
Then $a_1 = 2^i m$ and $a_2 = 2^k m$, so one must divide the other (e.g., if k > i then a_1 divides a_2)

WARNING WARNING WARNING WARNING

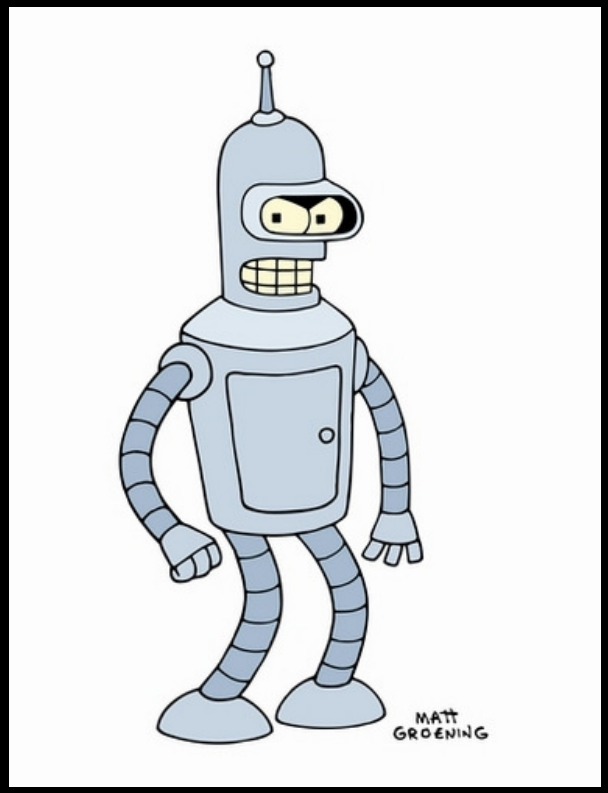
… about my proofs

During lectures, my proofs will generally contain the first two levels, but only part of the third

You should think about how to fill in the details! You aren't required to do this (except on some assignments) but it can help you learn.

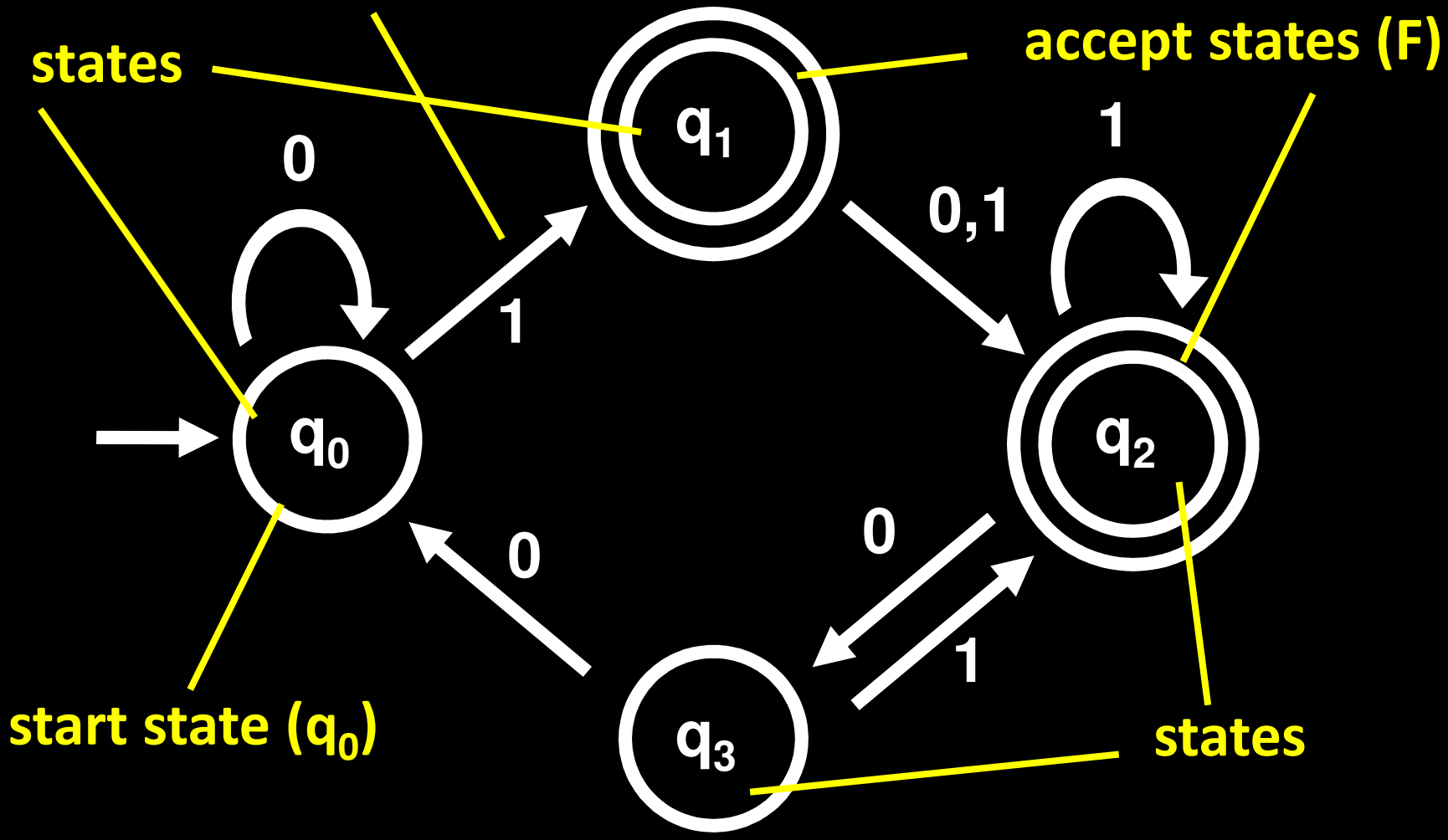
In this course, it's often the case that the second level of detail is more important than the third!

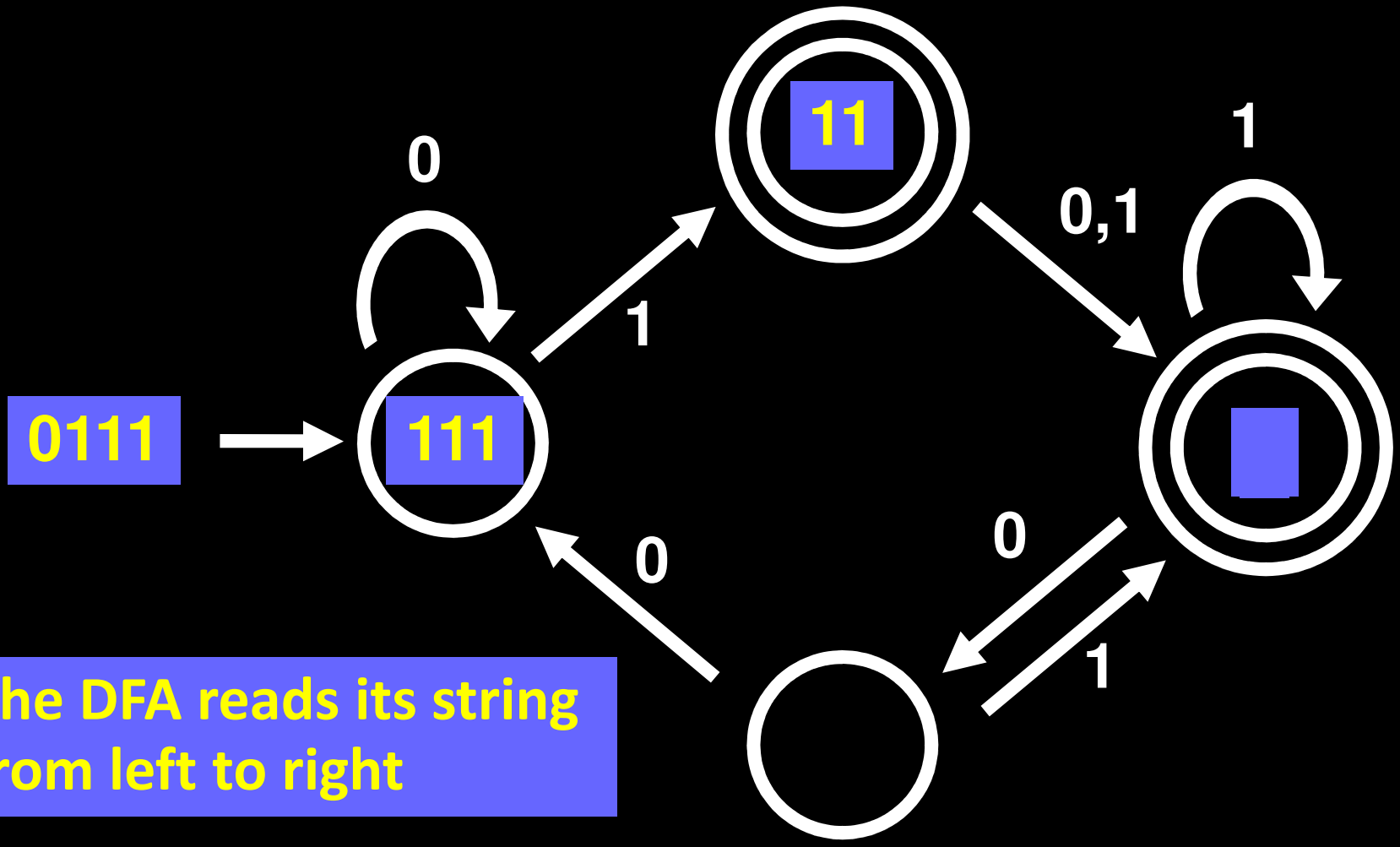
Deterministic **F**inite **A**utomata



Anatomy of Deterministic Finite Automata

transition: *for every state and alphabet symbol*





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

Otherwise, the automaton **rejects** the string

Some Vocabulary

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

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

Σ* = the set of all strings over Σ

For string **x**, **|x|** is the **length** of **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***

Why "Languages"?

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

Languages = functions that take strings as input and output a single bit

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

Define f such that $f(x) = 1$ if $x \in L$
$= 0$ otherwise

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 $w = w_1 \ldots w_n \in \Sigma^*$
M accepts w if there are $r_0, r_1, \ldots, r_n \in Q$, s.t.

- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$ for all i = 0, ..., n-1, and
- $r_n \in F$

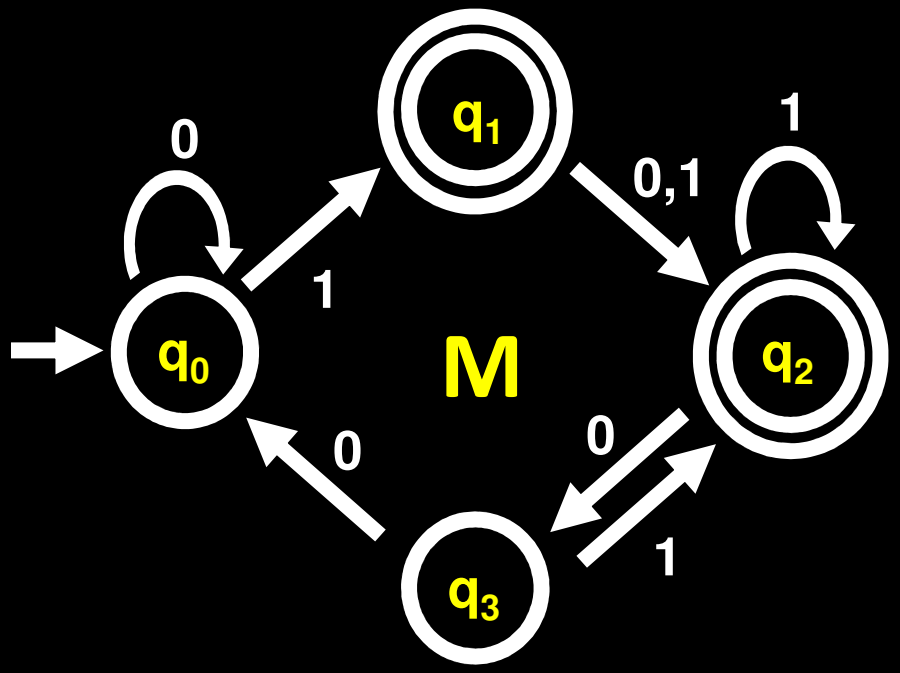
$M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$

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

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

$q_0 \in Q$ is start state

$F = \{q_1, q_2\} \subseteq Q$ accept states



*

δ	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

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

Q is the set of states (finite)

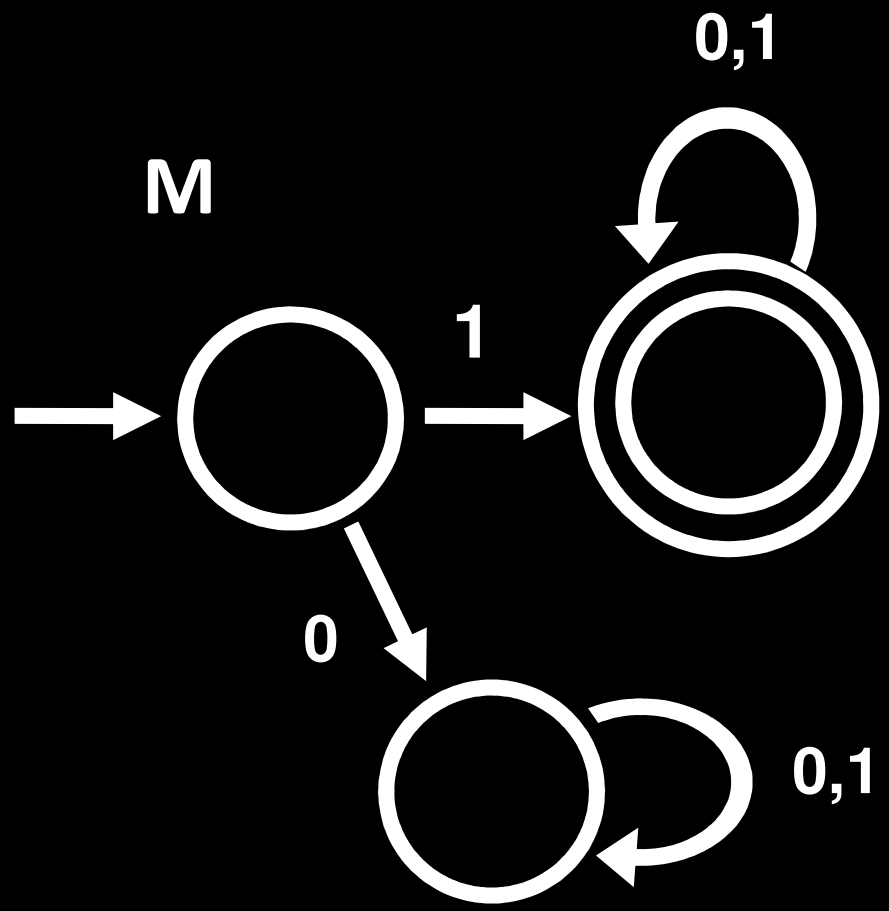
Σ is the alphabet (finite)

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

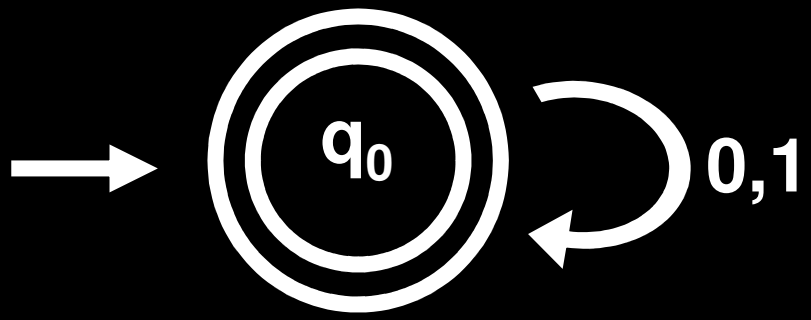
$q_0 \in Q$ is the start state

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

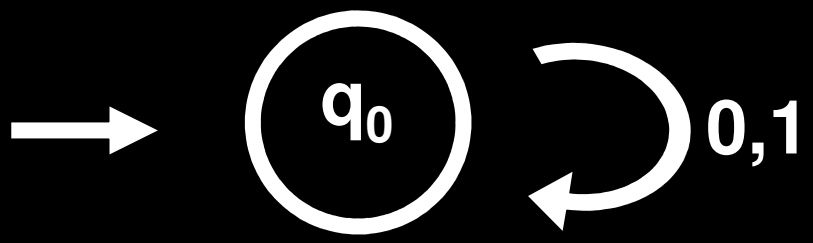
L(M) = set of all strings that **M** accepts
= "the language recognized by **M**"
= the function computed by **M**



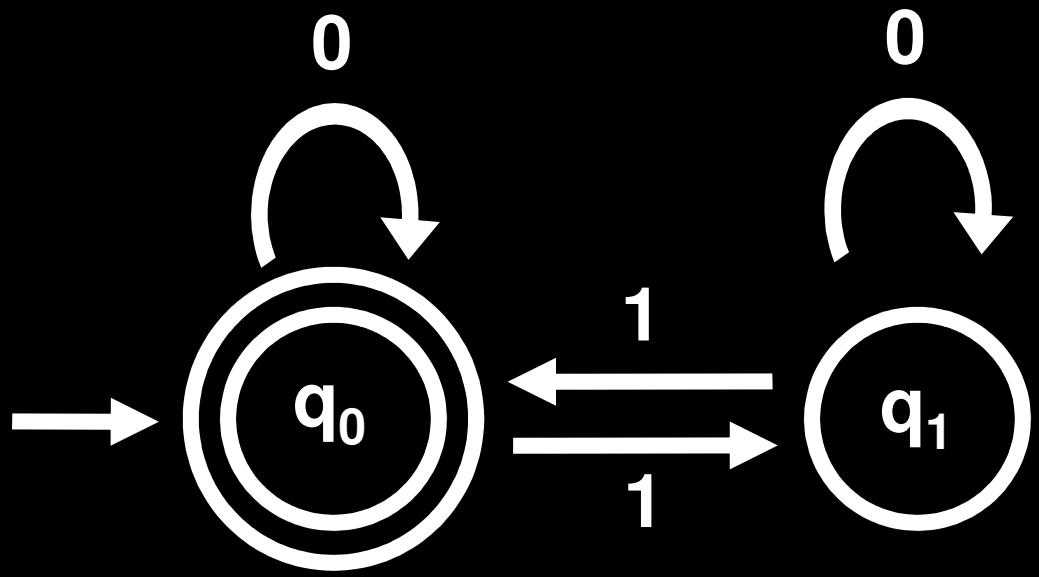
L(M) = { w | w begins with 1}



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

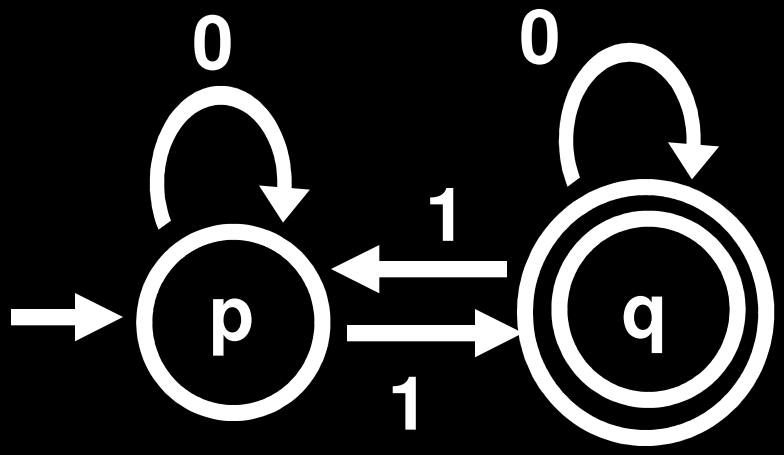


$L(M) = \emptyset$



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

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



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: $\varepsilon \notin L$ and $\varepsilon \notin L(M)$

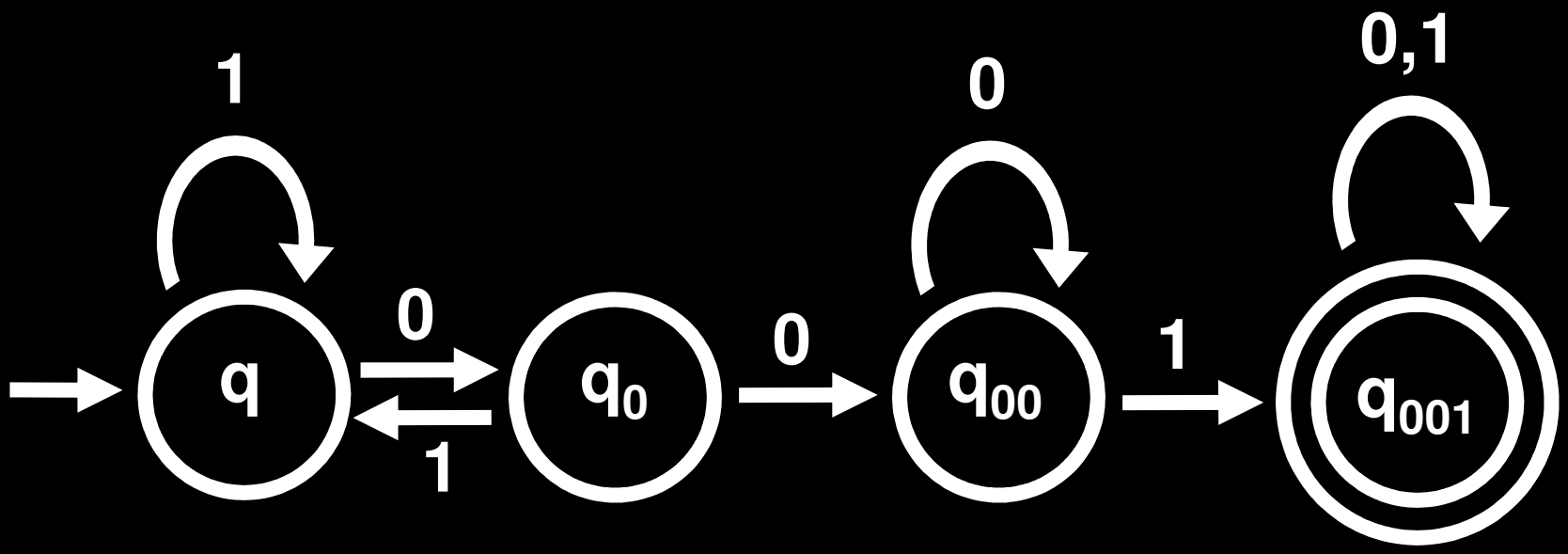
Induction Step Suppose for all $w \in \Sigma^*$, $|w| = n$,
M accepts w ⇔ **w** has odd number of 1s

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

w has odd # of 1's iff M is in state **q** after reading **w**

1. If our string of length n+1 is w0, and w has even # of 1's, then M is in state **p** after reading **w0** → M rejects

Build a DFA that accepts exactly the strings containing 001



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 has an even 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"

ne construction of $\mathfrak{N}$ 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
her

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

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-
l definitions needed for two-way
ne main point is that, as with the
omaton, the table of moves of a
tomaton 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

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

Proof: Let

$M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ be a finite automaton for L_1

and

$M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ be a finite automaton for L_2

We want to construct a finite automaton

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

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

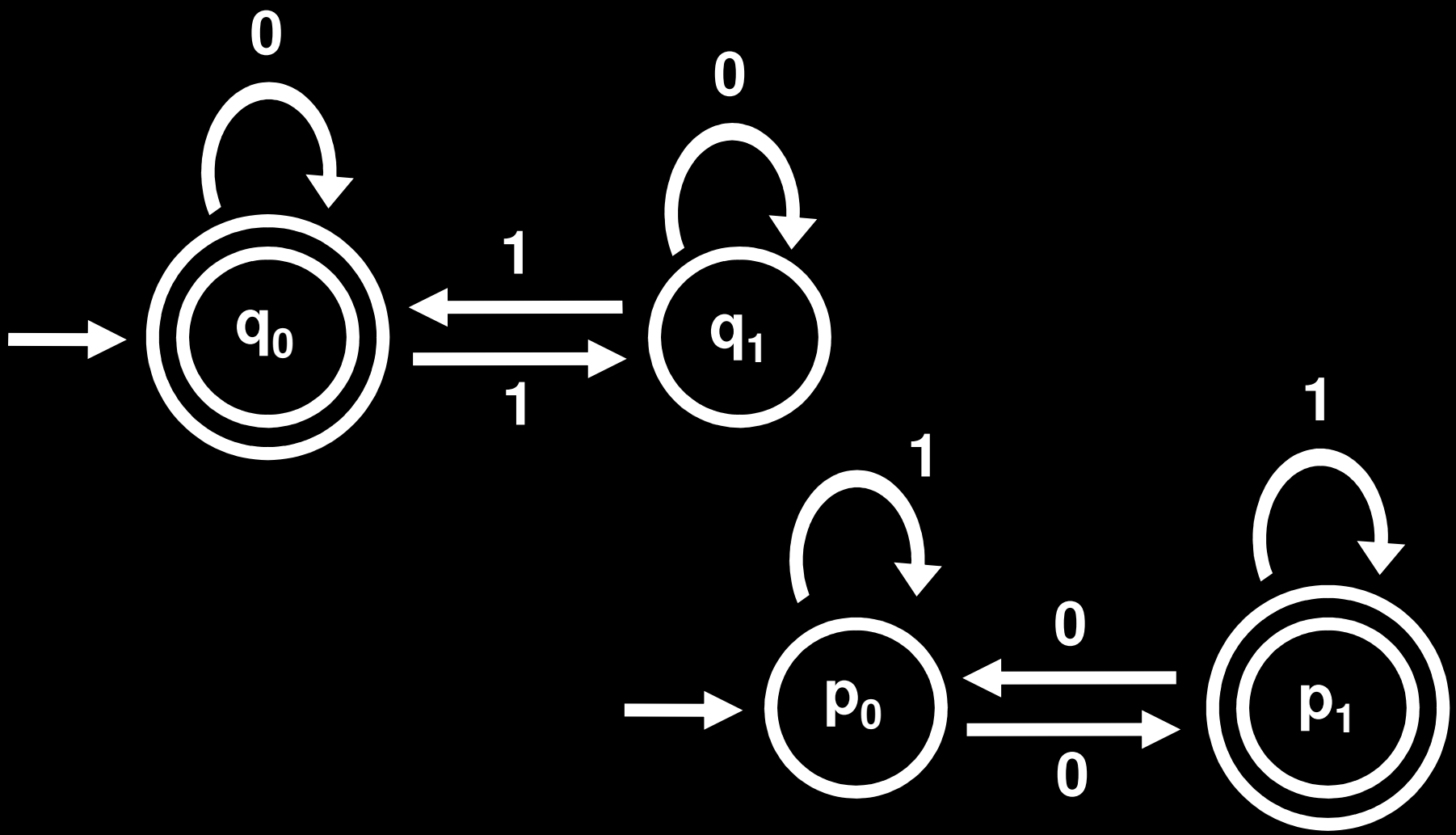
$= Q_1 \times Q_2$

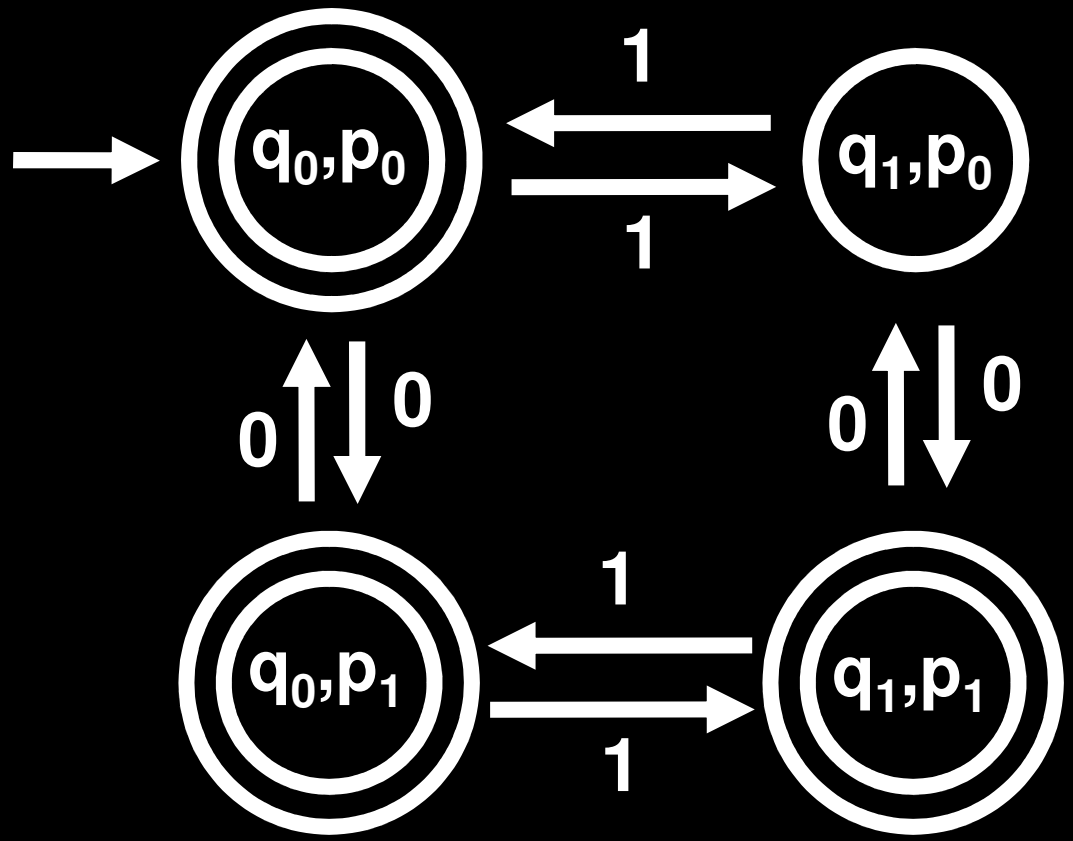
$q_0 = (q_0^1, q_0^2)$

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

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

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





What about the INTERSECTION of two languages?

Intersection Theorem for Regular Languages

Given two languages, L_1 and L_2, define the **intersection of L_1 and L_2** as

$$L_1 \cap L_2 = \{ w \mid w \in L_1 \text{ and } w \in L_2 \}$$

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

Proof Idea: Again, run both M_1 and M_2

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

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

$= Q_1 \times Q_2$

$q_0 = (q_0^1, q_0^2)$

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

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

Union Theorem for Regular Languages

The union of two regular languages is also a regular language

"Regular Languages Are Closed Under Union"

Intersection Theorem for Regular Languages

The intersection of two regular languages is also a regular language

Complement Theorem for Regular Languages

The complement of a regular language is also a regular language

In other words,

if L is regular than so is ¬L,

where ¬L= { w ∈ Σ* | w ∉ L }

Proof ?

Some Operations on Languages

- **Union:** $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$
- **Intersection:** $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$
- **Complement:** $\neg A = \{ w \in \Sigma^* \mid w \notin A \}$

Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A, w_i \in \Sigma \}$

Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Star: $A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \}$

The **Reverse** of a Language

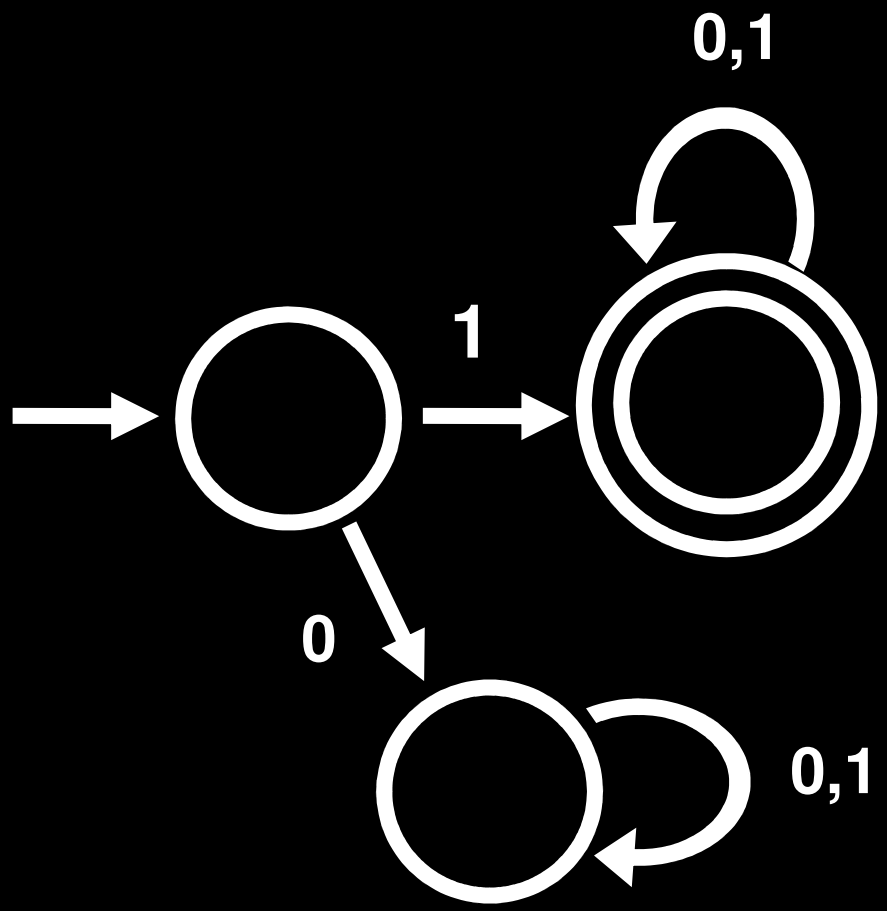
Reverse of L:

$L^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in L, w_i \in \Sigma \}$

If L is recognized by the usual kind of DFA,
Then L^R is recognized by a DFA that reads its strings from *right to left*!

Question: If L is regular, then is L^R also regular?

Can every "Right-to-Left" DFA be replaced by a normal "Left-to-Right" DFA?



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

Suppose our machine reads strings from *right* to *left*…
Then **L(M) = {w | w ends with a 1}. Is this regular?**

Reverse Theorem for Regular Languages

The reverse of a regular language is also a regular language

"Regular Languages Are Closed Under Reverse"

If **a language can be recognized by a DFA that reads strings from *right* to *left*,**
then **there is an "normal" DFA that accepts the same language**

Counterintuitive! DFAs have finite memory…

Reversing DFAs

Assume L is a regular language.
Let M be a DFA that recognizes L

We want to build a machine M^R that accepts L^R

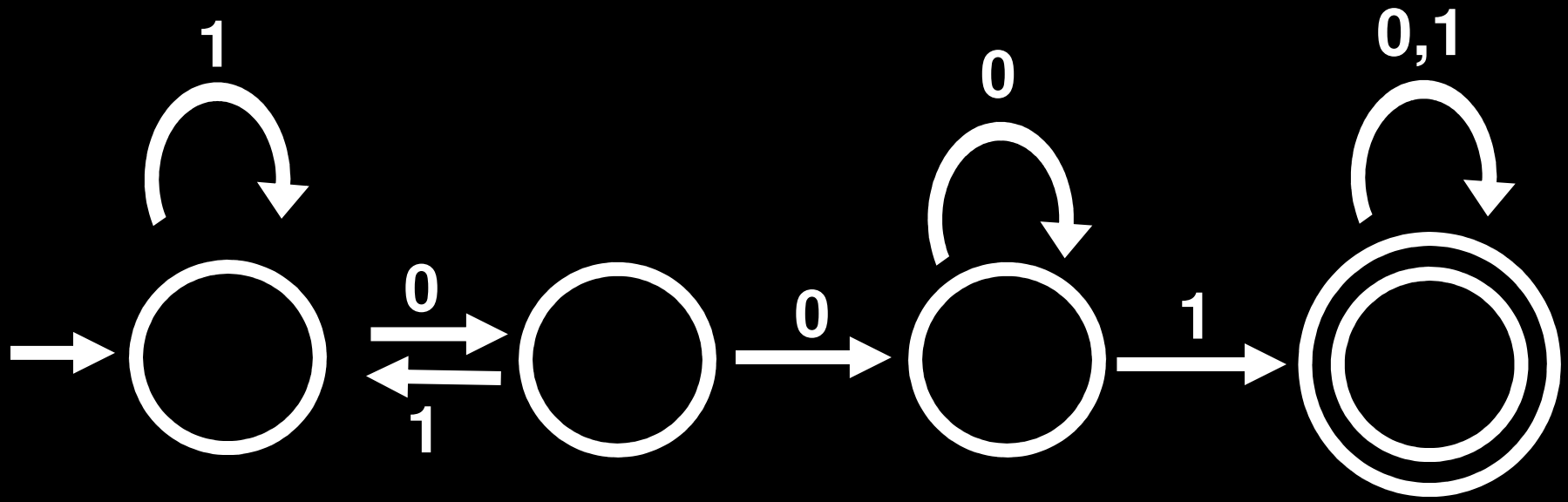
If M accepts w, then w describes a directed path in M from the start state to an accept state

First Attempt: Try to define M^R as M with the arrows reversed, turn start state into a final state, turn final states into starts

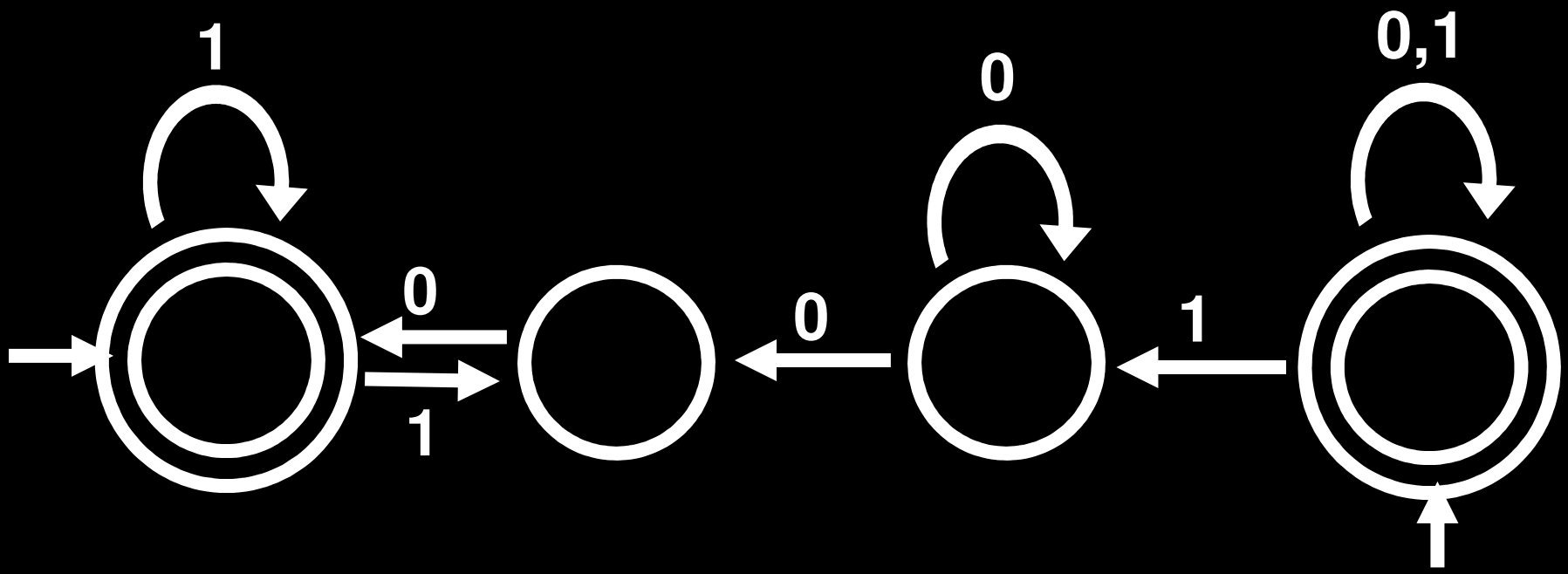
Problem: M^R IS NOT ALWAYS A DFA!

It could have many start states

Some states may have *more than one* outgoing edge, or none at all!


1
0
1
0
0
1
0,1

Non-deterministic Finite Automata (NFA)



What happens with 100?

We will say this new machine **accepts** a string if
there is some path that reaches
some accept state from *some* start state

end