Automata Theory

Lecture on Discussion Course of CS120

Runzhe Yang @ SJTU ACM CLASS

This Lecture is about

Mathematical Models of Computation.

Why Should **I Care**?

- Ways of **thinking**.
- Theory can drive practice.
- Don't be an **Instrumentalist**.

Mathematical Models of Computation

(predated computers as we know them)

1 **Automata and Languages:** (1940's)

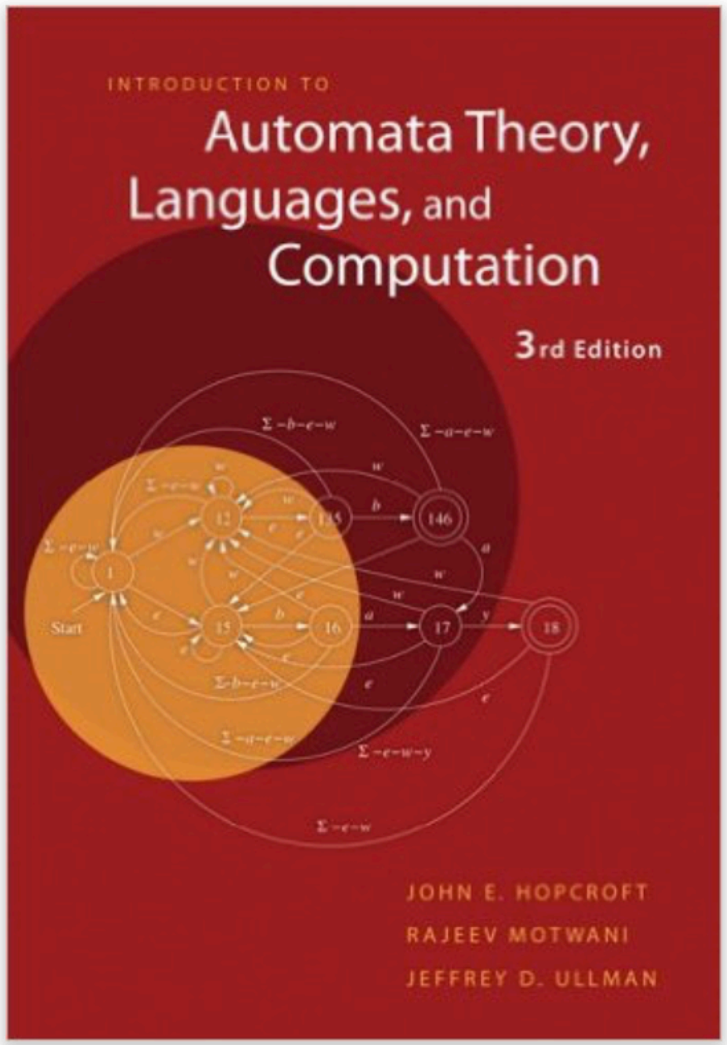
finite automata, regular languages, pushdown automata, context-free languages, pumping lemmas.

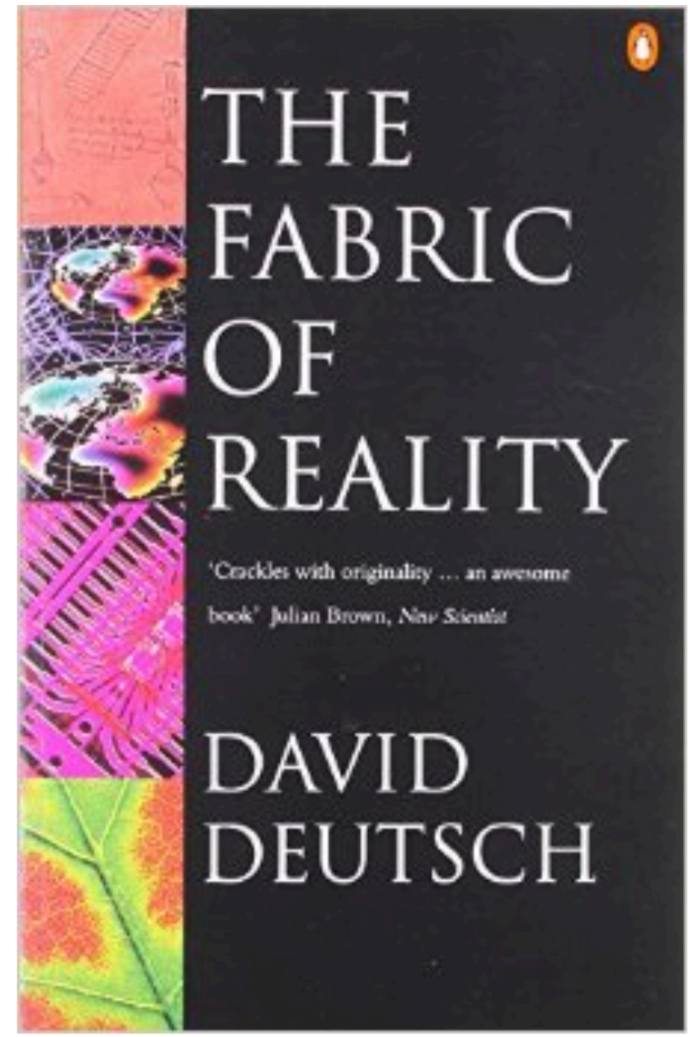
2 **Computability Theory:** (1930-40's)

Turing Machines, decidability, reducibility, the arithmetic hierarchy, the recursion theorem, the Post correspondence problem.

3 **Complexity Theory and Applications:** (1960-70's)

time complexity, classes P and NP, NP-completeness, space complexity PSPACE, PSPACE-completeness, the polynomial hierarchy, randomized complexity, classes RP and BPP.


INTRODUCTION TO
Automata Theory, Languages, and Computation
3rd Edition
JOHN E. HOPCROFT
RAJEEV MOTWANI
JEFFREY D. ULLMAN


THE FABRIC OF REALITY
'Crackles with originality … an awesome book' Julian Brown, New Scientist
DAVID DEUTSCH

Let me emphasize **Proofs** a bit more.

A good proof:

1. Correct
2. Easy to understand

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

✓ True or False:

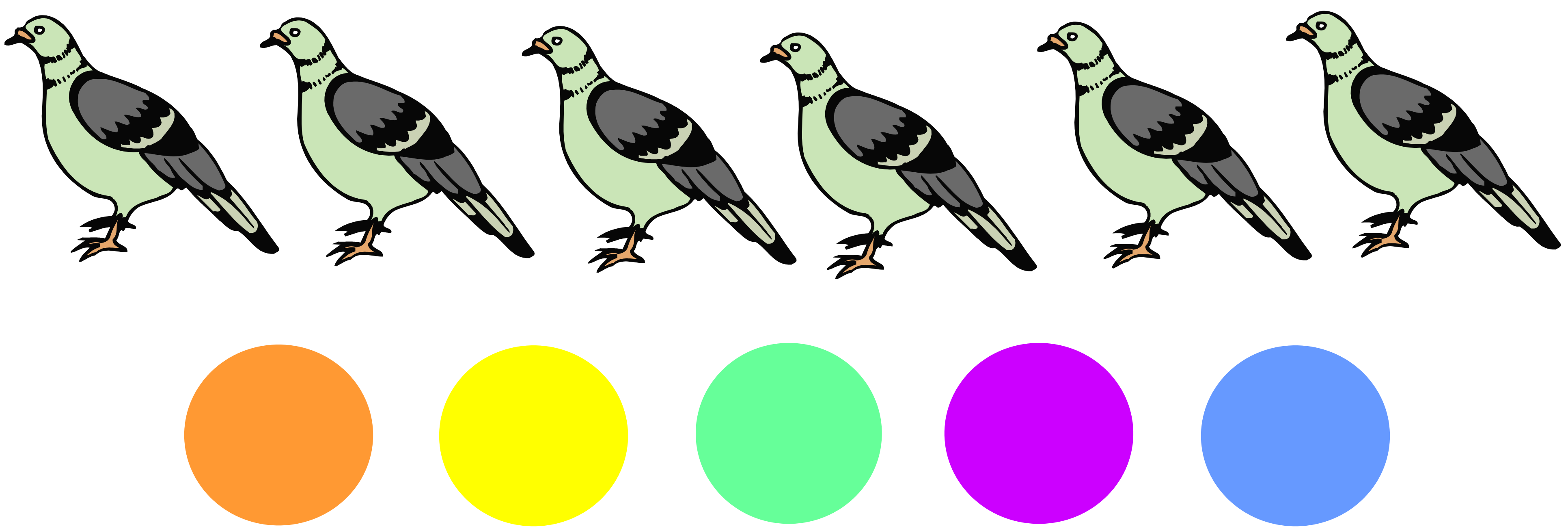
There are always two numbers in A such that one divides the other.

LEVEL 1

HINT 1:

THE PIGEONHOLE PRINCIPLE

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

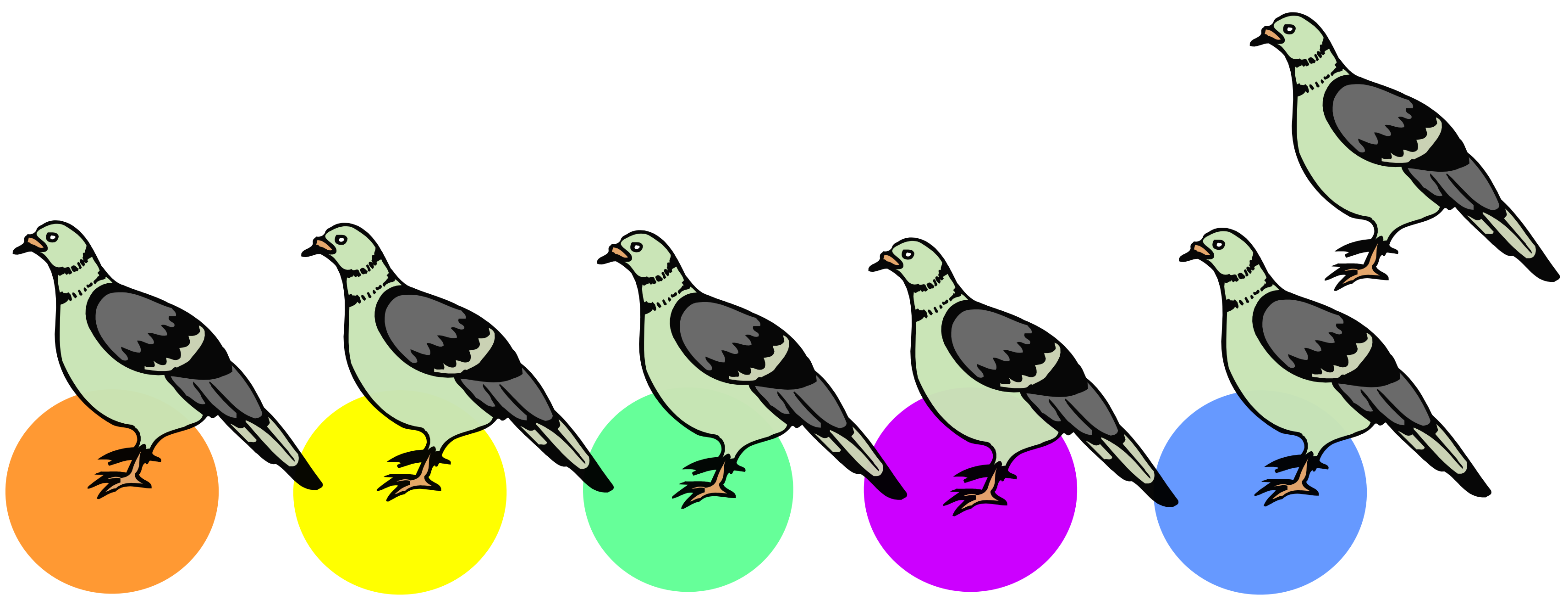


LEVEL 1

HINT 1:

THE PIGEONHOLE PRINCIPLE

If you put 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 put 6 pigeons in 5 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

LEVEL 2

Proof Idea:

Given: $A \subseteq \{1, 2, \ldots, 2n\}$ with $|A| = n + 1$.

Show: There is an integer m and $a_1 \neq a_2 \in A$ such that $a_1 = 2^i m$ and $a_2 = 2^j m$.

LEVEL 3

Proof:

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

Write every number in A as $a = 2^k m$, where m is an odd number.

How many odd numbers in $\{1,\ldots,2n-1\}$? n

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

Say a_1 and a_2 have the same odd part m.
Then $a_1 = 2^i m$ and $a_2 = 2^j m$, so one must divide the other.

We expect your proofs to have three levels:

The first level should be a one-word or one-phrase "HINT" of the proof

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

The second level should be a short one-paragraph description or "KEY IDEA"

The third level should be the FULL PROOF

Double standards :-)

HINT + KEY IDEA (+ FULL PROOF)

Deterministic Finite Automata (DFA)

NOTATION

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

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

Σ^* denotes the set of finite length sequences of elements of Σ

For x a string, $|x|$ is the length of x

The unique string of length 0 will be denoted by ε and will be called the empty or null string

A language over Σ is a set of strings over Σ, ie, a subset of Σ^*

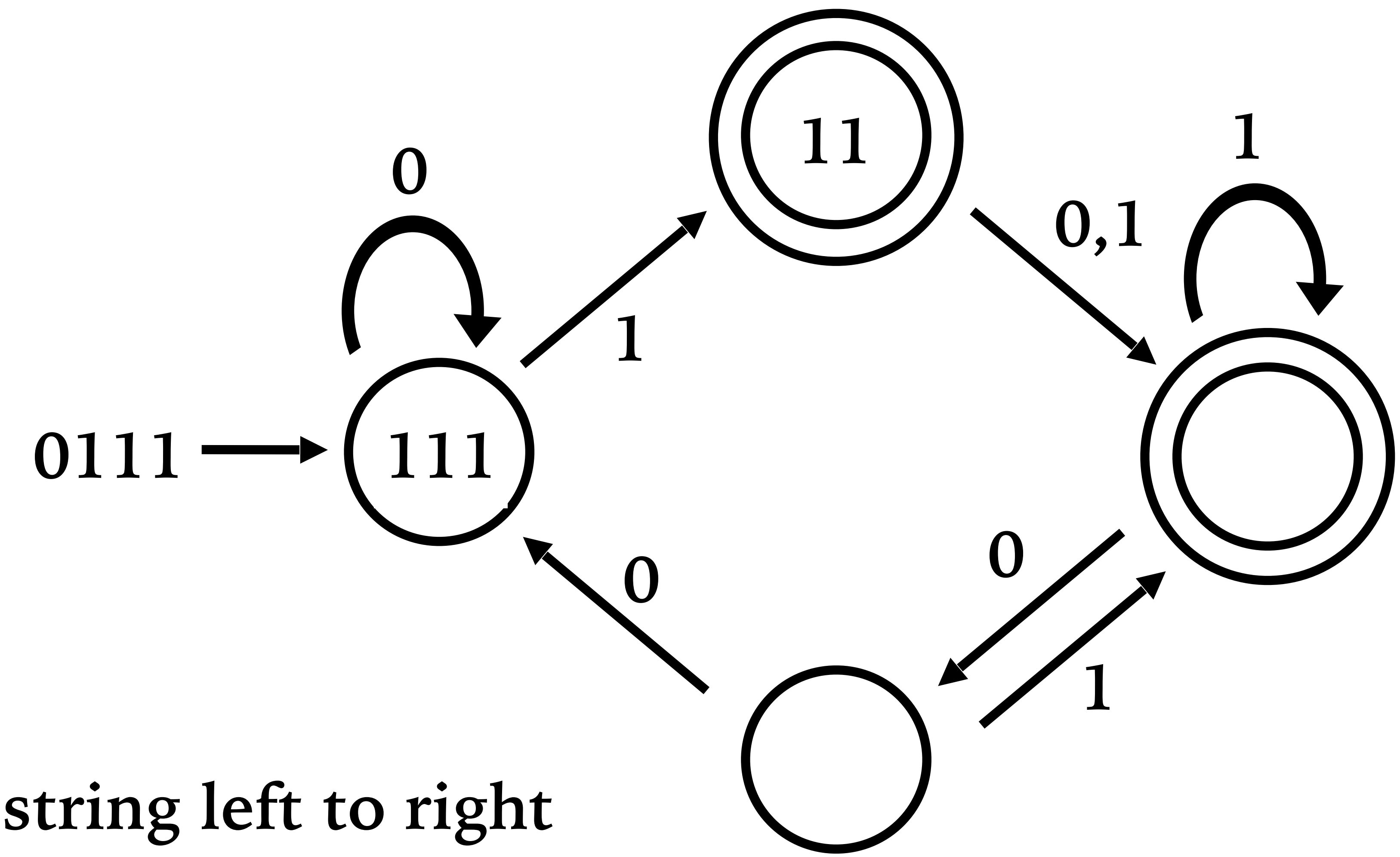
Finite Automata

finite # of
internal states

Input (String) -> -> Output (Yes/No)

Divice

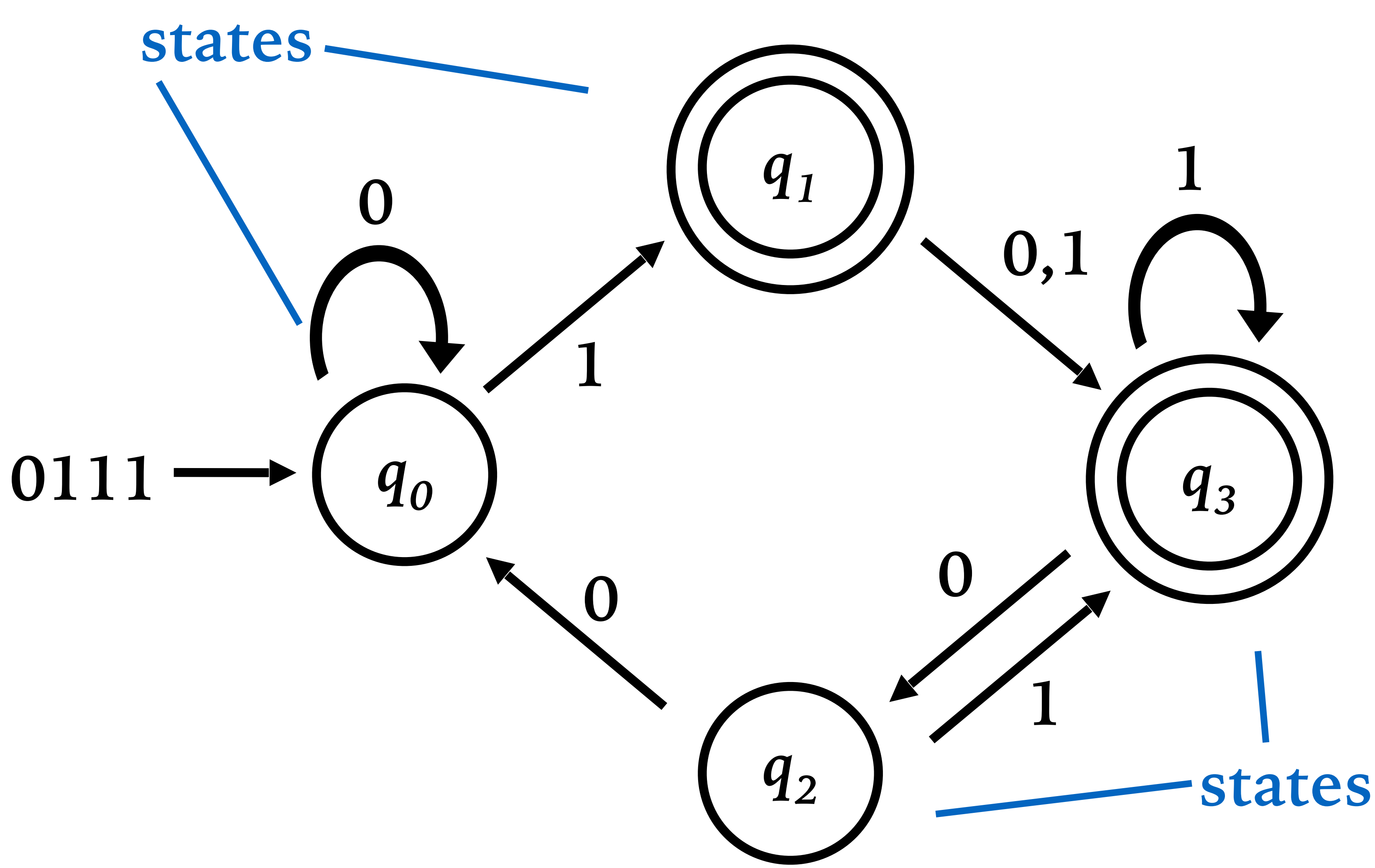
A Deterministic Finite Automata



Read string left to right

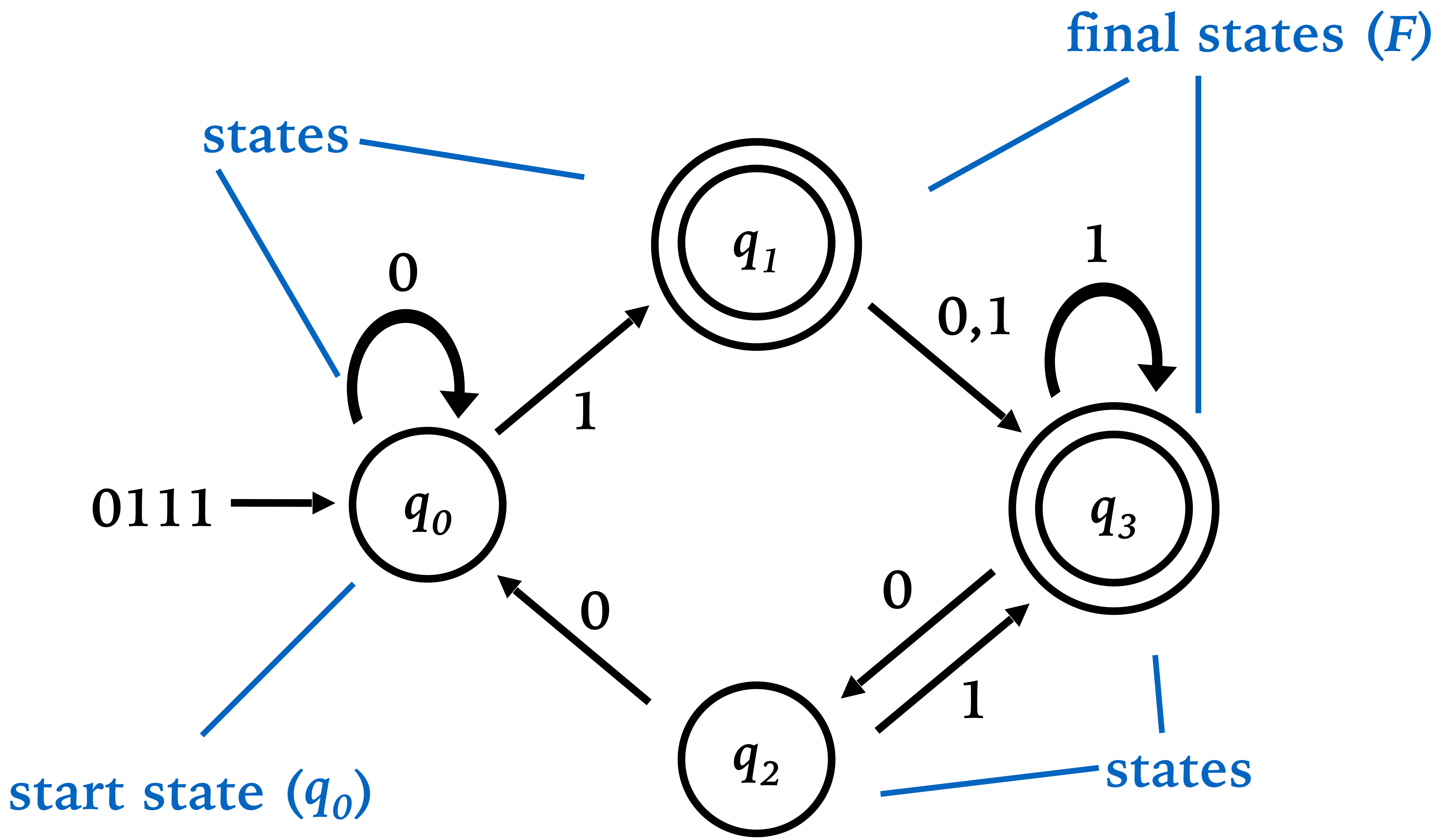
The machine **accepts** a string if the process ends in a **double circle**.

A Deterministic Finite Automata



The machine accepts a string if the process ends in a double circle.

A Deterministic Finite Automata



The machine accepts a string if the process ends in a double circle.

A Deterministic Finite Automata

is represented by 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 states

Let $w_1, \ldots, w_n \in \Sigma$ and $w = w_1 \ldots w_n \in \Sigma^*$
Then 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 $i = 0, \ldots, n-1$, and
- $r_n \in F$

A Deterministic Finite Automata

is represented by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$:

Q is the set of states (finite)

Σ is the alphabet (finite)

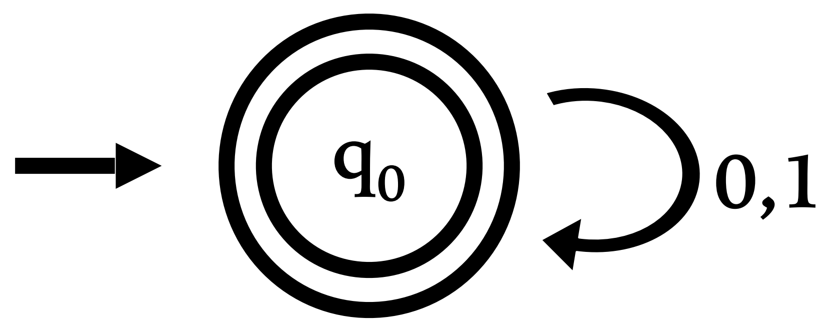
$\delta : Q \times \Sigma \to Q$ is the transition function

$q_0 \in Q$ is the start state

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

$L(M)$ = the language of machine M
= set of all strings machine M accepts

Warming-up



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

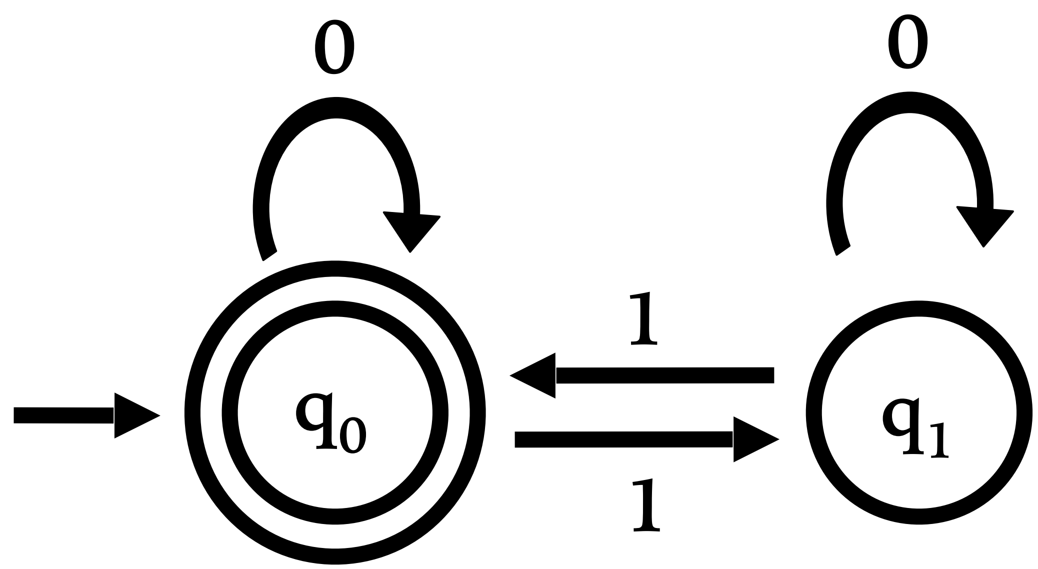
Warming-up

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

Warming-up

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

Warming-up



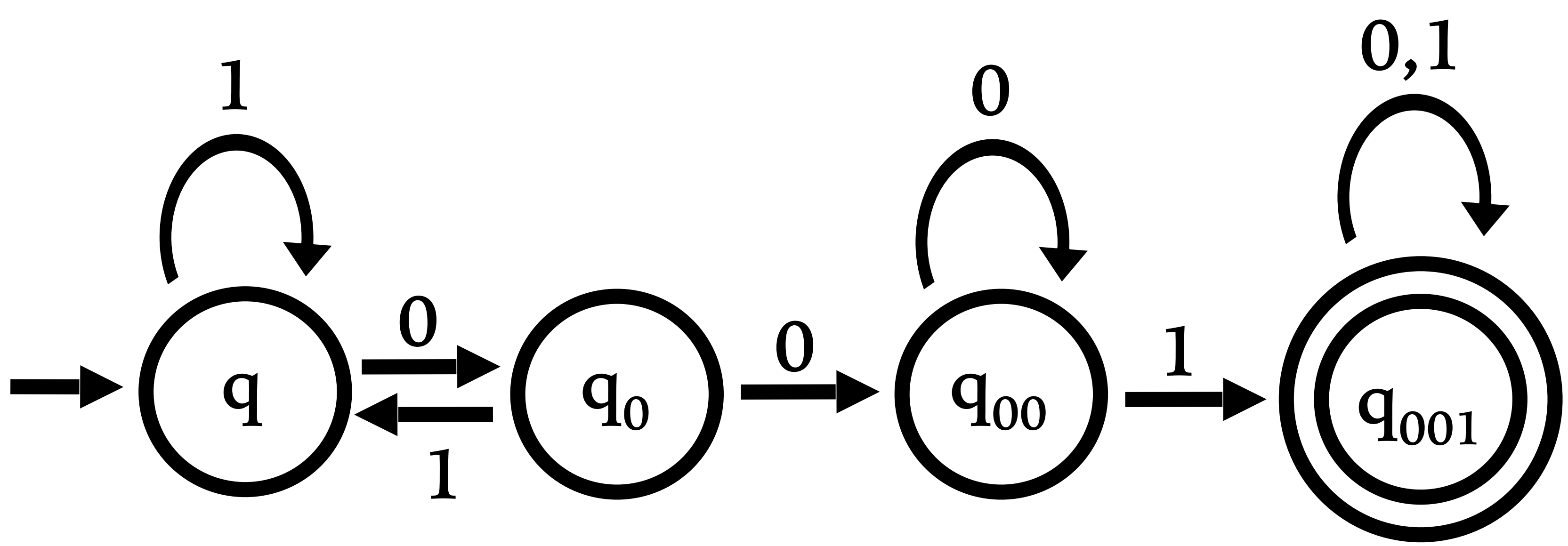
L(M) = { w | w has an even number of 1s}

Warming-up

Build an automaton that accepts all and only those strings that contain 001

Warming-up

Build an automaton that accepts all and only those strings that contain 001



Regular Language

A language L is regular if it is recognized by a deterministic finite automaton (DFA), i.e. if there is a DFA M such that $L = L(M)$.

L = { w | w contains 001} is regular

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

UNION THEOREM

Given two languages, L_1 and L_2, define the union of L_1 and L_2 as

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

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 finite automaton for L_1

and

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

We want to construct a finite automaton
$M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $L = L_1 \cup L_2$

Theorem:

The union of two regular languages is also a regular language.

Idea: Run both M_1 and M_2 at the same time!

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

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

$= Q_1 \times Q_2$

$q_0 = (q_0, q_0)$

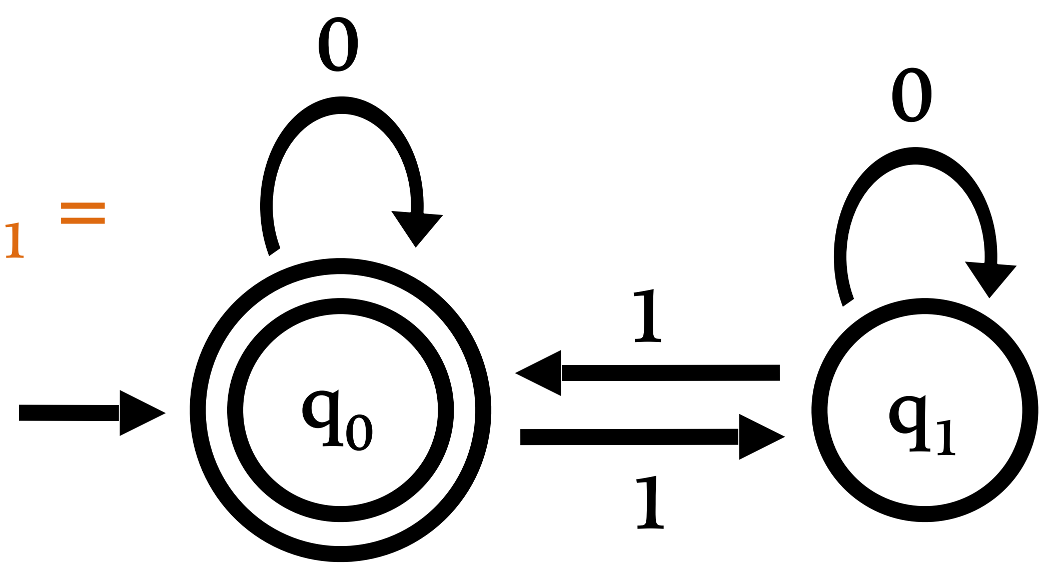
$F = \{ (q_1, q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2 \}$

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

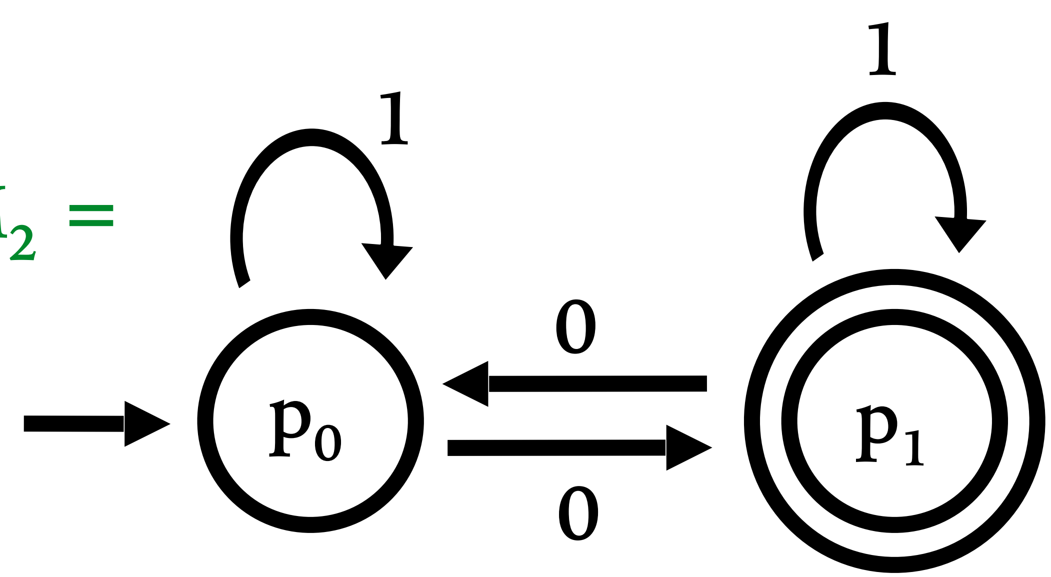
Theorem:

The union of two regular languages is also a regular language.

$M_1 =$



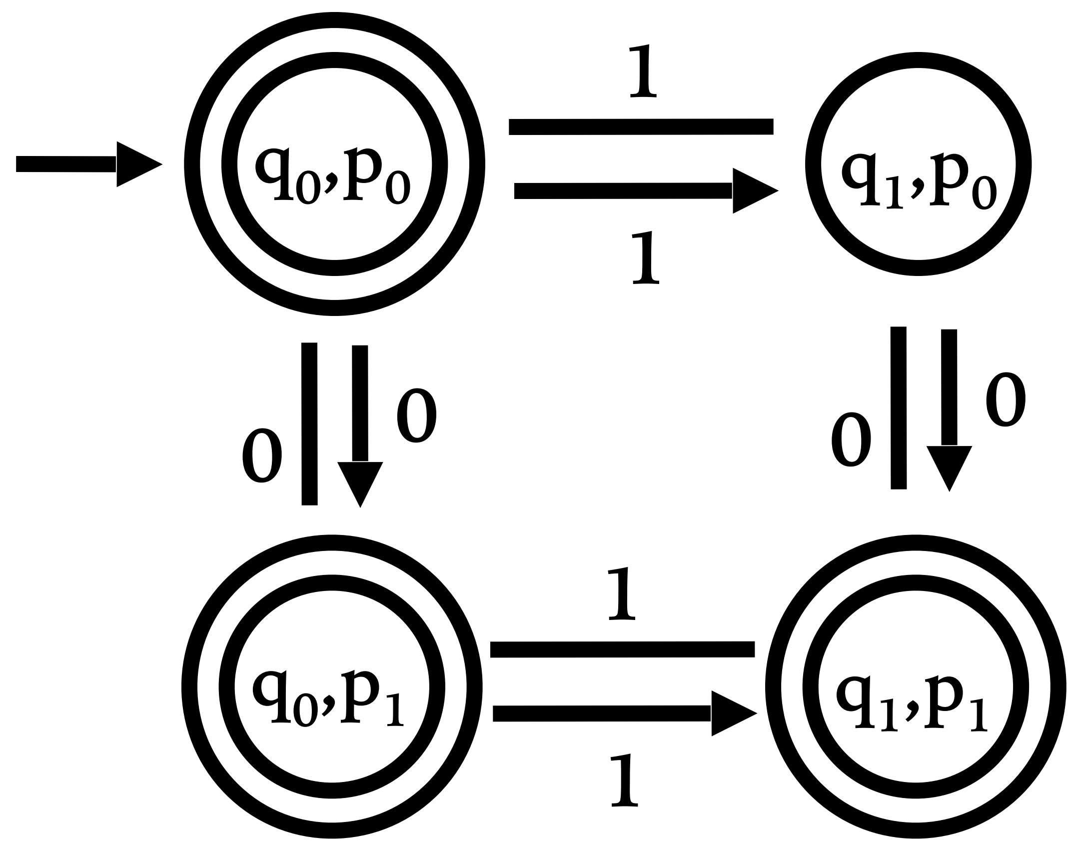
$M_2 =$



Theorem:

The union of two regular languages is also a regular language.

$M =$



INTERSECTION THEOREM

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.

COMPLEMENT THEOREM

Given language L, define the complement of L as

$\neg L = \{ w \in \Sigma^* \mid w \notin L \}$

Theorem:

The complement of a regular languages is also a regular language.

THE REGULAR OPERATIONS

- **Union: $L_1 \cup L_2$ = { w | w $\in L_1$ or w $\in L_2$ }**
- **Intersection: $L_1 \cap L_2$ = { w | w $\in L_1$ and w $\in L_2$ }**
- **Negation: $\neg$ L = { w $\in \Sigma^*$ | w $\notin$ L }**

Reverse: L^R = { $w_1 \ldots w_k$ | $w_k \ldots w_1 \in$ L }

Concatenation: $L_1 \cdot L_2$ = { vw | v $\in L_1$ and w $\in L_2$ }

Star: L^* = { $w_1 \ldots w_k$ | k $\geq$ 0 and each $w_i \in$ L }

REVERSE THEOREM

Given language L, define the

reverse of L^R as

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

Theorem:

The reverse of a regular languages is also a regular language.

Theorem:

The reverse of a regular languages is also a regular language.

Proof: Let

$M = (Q, \Sigma, \delta, q_0, F)$ that recognizes L.

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

We want to construct a finite automaton M^R as M with the arrows reversed.

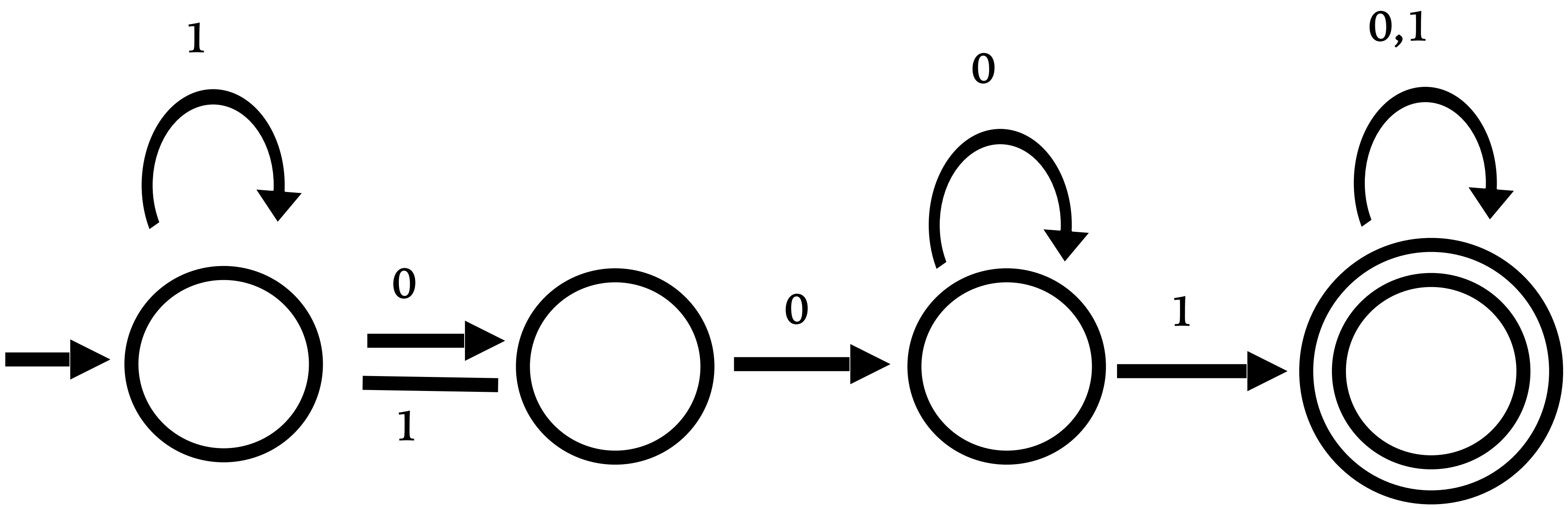
…

M^R IS NOT ALWAYS A DFA!

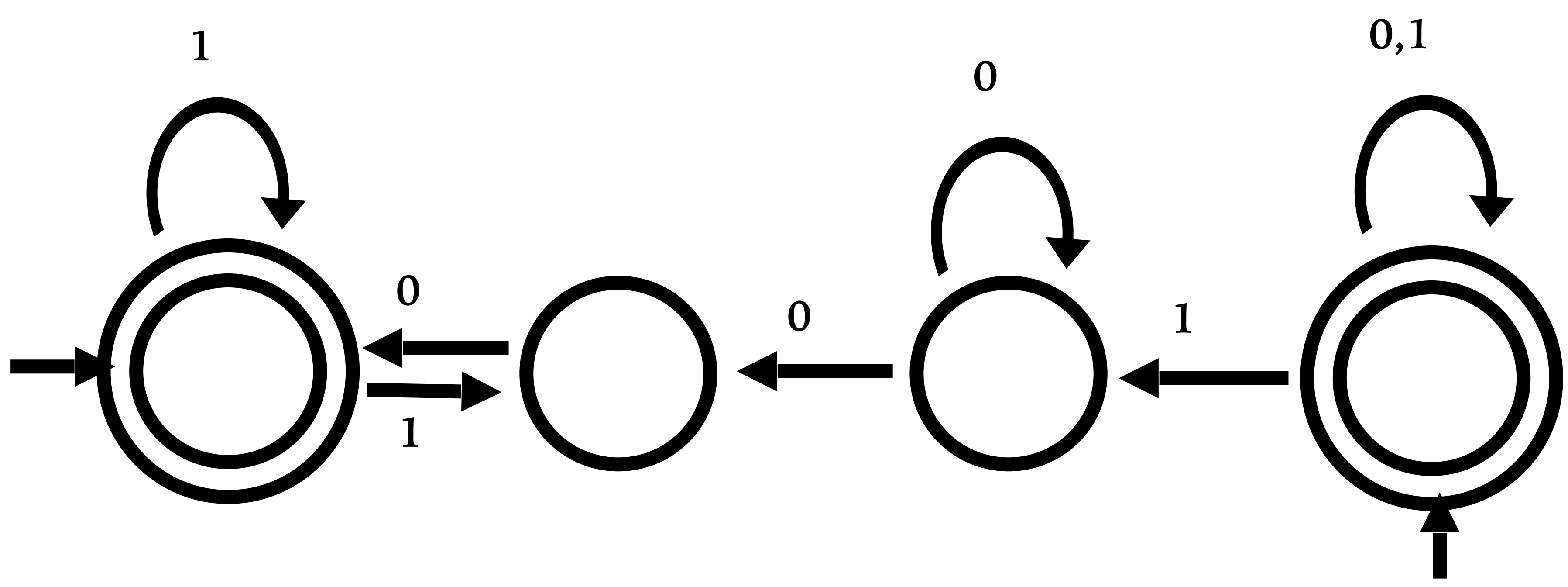
- It may have many start states

- Some states may have too many outgoing edges, or none

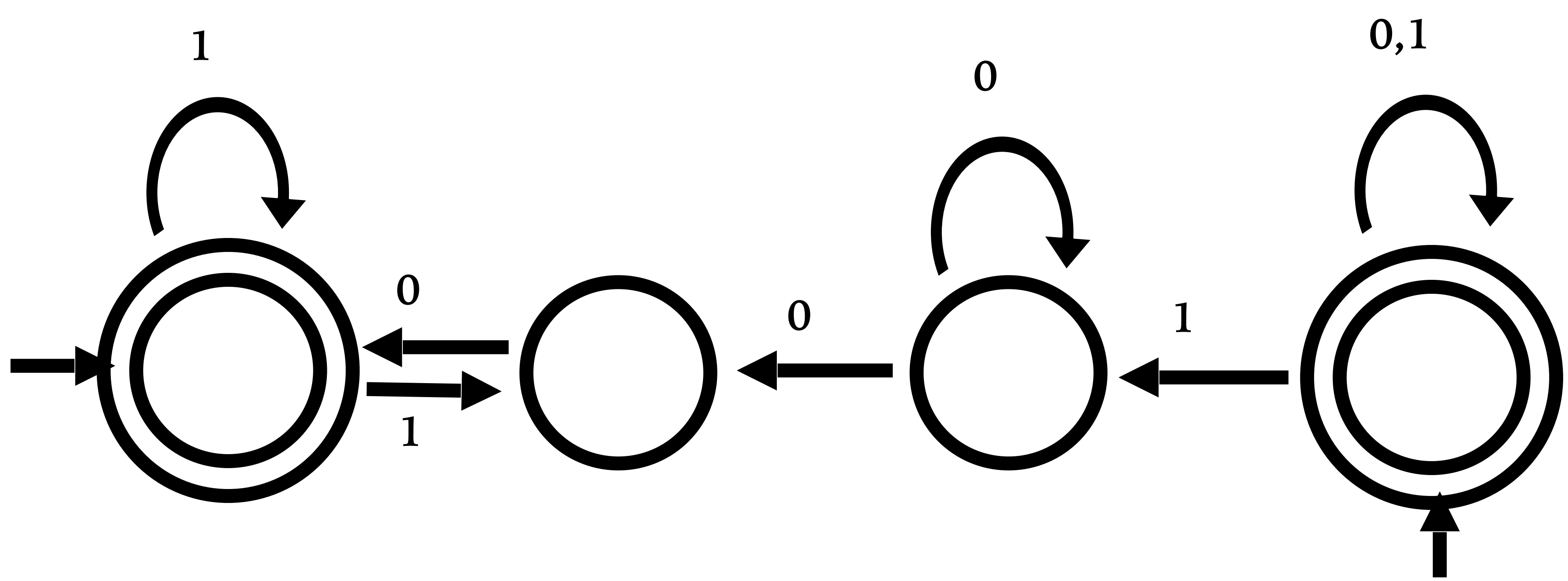
M^R IS NOT ALWAYS A DFA!



M^R IS NOT ALWAYS A DFA!

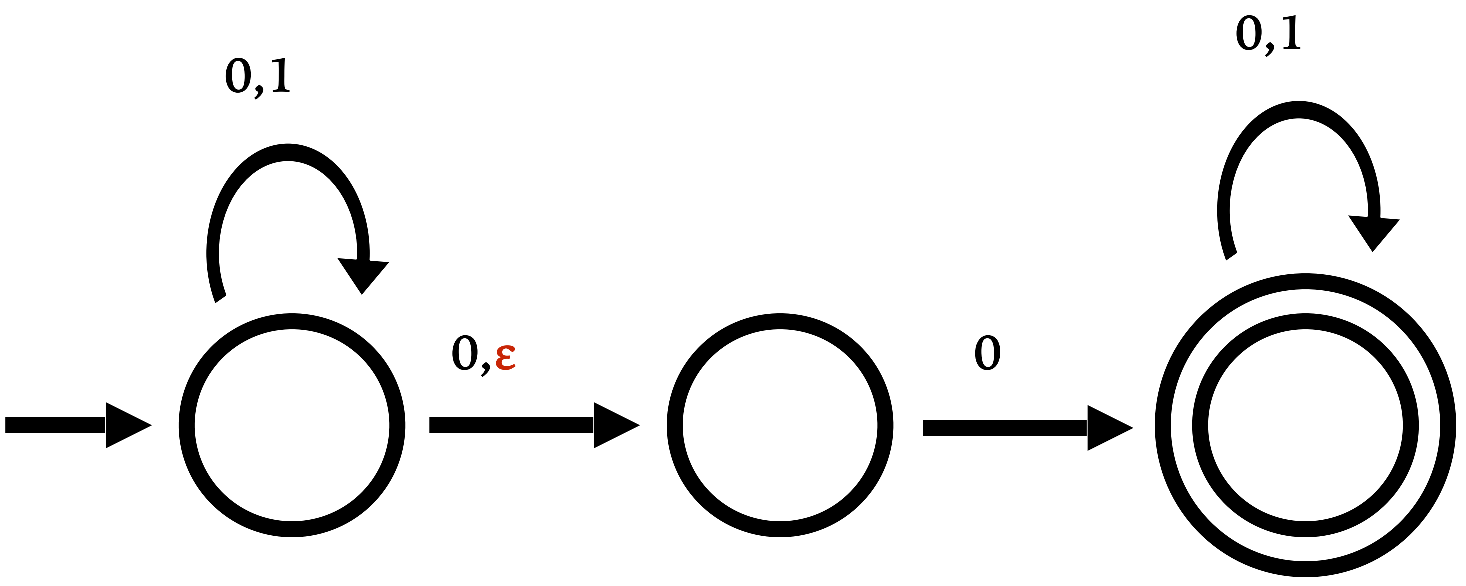


Non-Determinism



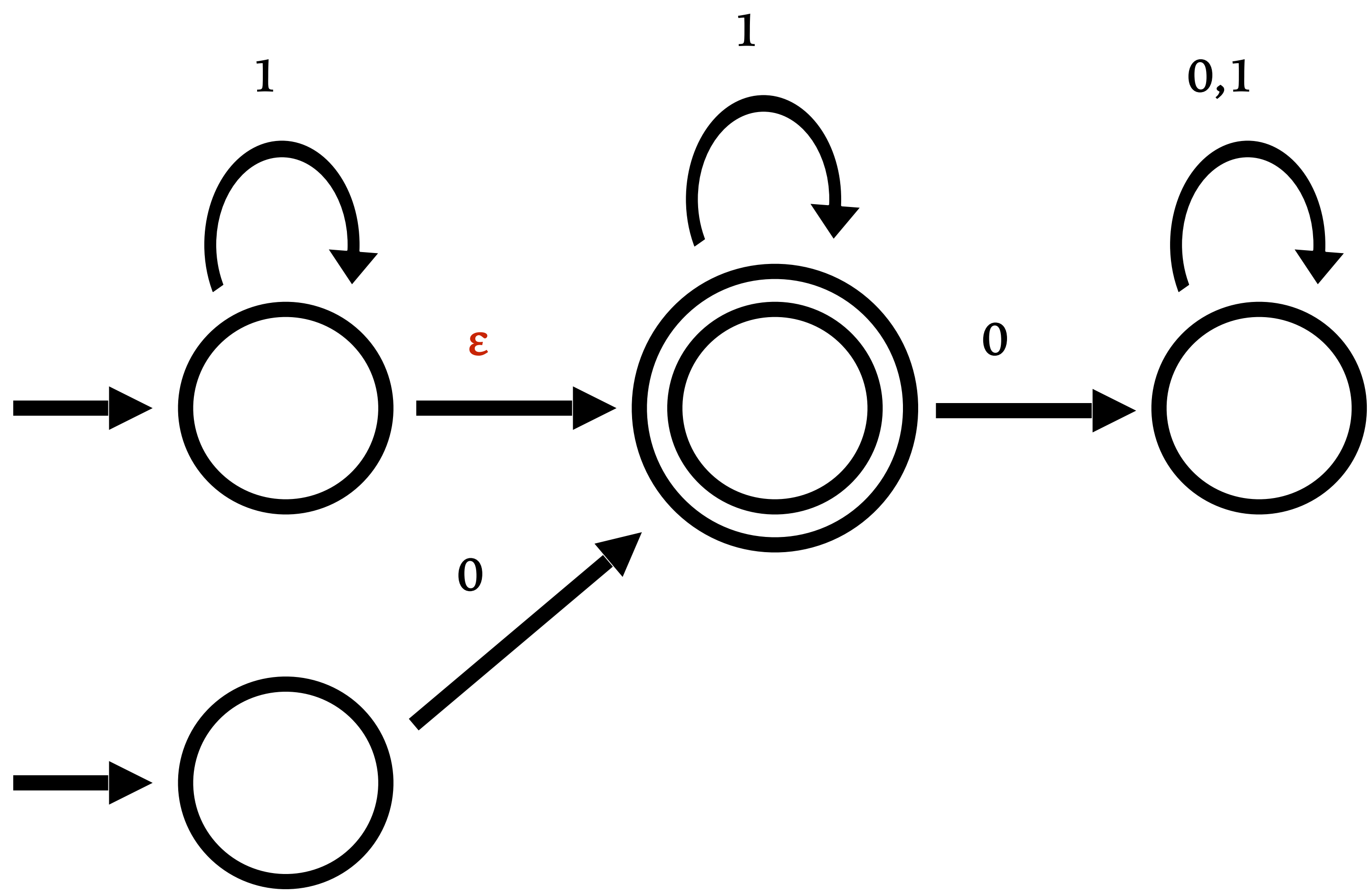
We will say that the machine accepts if there is some way to make it reach an accept state

Example



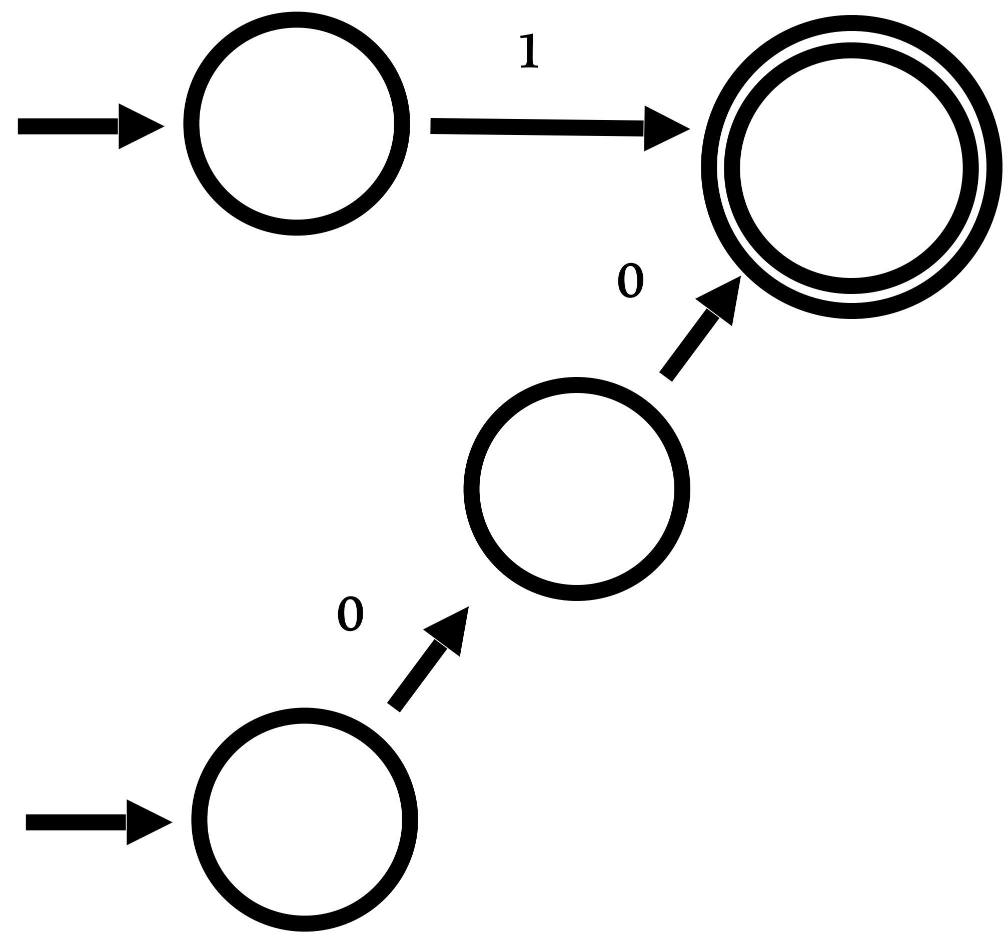
At each state, possibly zero, one or many out arrows for each $\sigma \in \Sigma$ or with label ε

Example



Possibly many start states

Example



$L(M) = \{1, 00\}$

A Non-Deterministic Finite Automata (NFA)

is represented by a 5-tuple $N = (Q, \Sigma, \delta, Q_0, F)$:

Q is the set of states (finite)

Σ is the alphabet (finite)

$\delta : Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$ is the transition function

$Q_0 \subseteq Q$ is the set of start state

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

Let $w_1, \dots, w_n \in \Sigma \cup \{\varepsilon\}$ and $w = w_1 \dots w_n \in \Sigma_\varepsilon^*$
Then N accepts w if there are $r_0, r_1, \dots, r_n \in Q$, s.t.

- $r_0 = Q_0$
- $r_{i+1} \in \delta(r_i, w_{i+1})$, for $i = 0, \dots, n-1$, and
- $r_n \in F$

A Non-Deterministic Finite Automata (NFA)

is represented by a 5-tuple $N = (Q, \Sigma, \delta, Q_0, F)$:

Q is the set of states (finite)

Σ is the alphabet (finite)

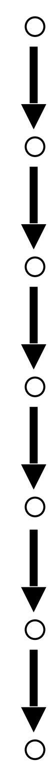
→ $\delta : Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$ is the transition function

→ $Q_0 \subseteq Q$ is the set of start state

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

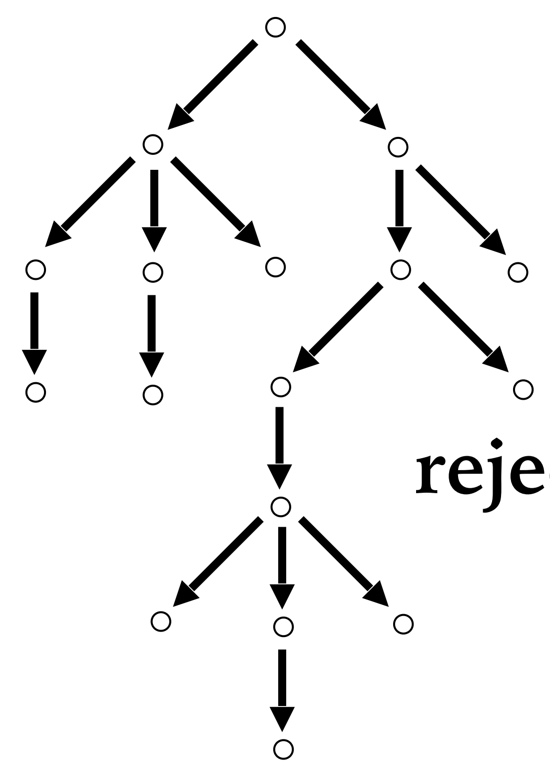
$L(N)$ = the language of machine N
= set of all strings machine N accepts

Deterministic Computation



accept or reject

Non-Deterministic Computation

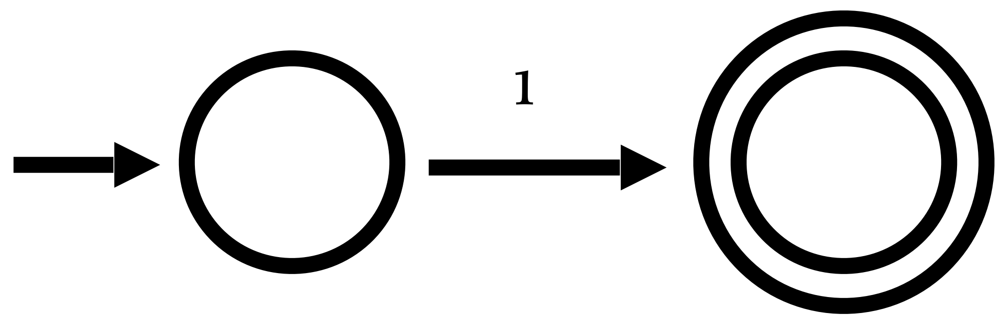


reject

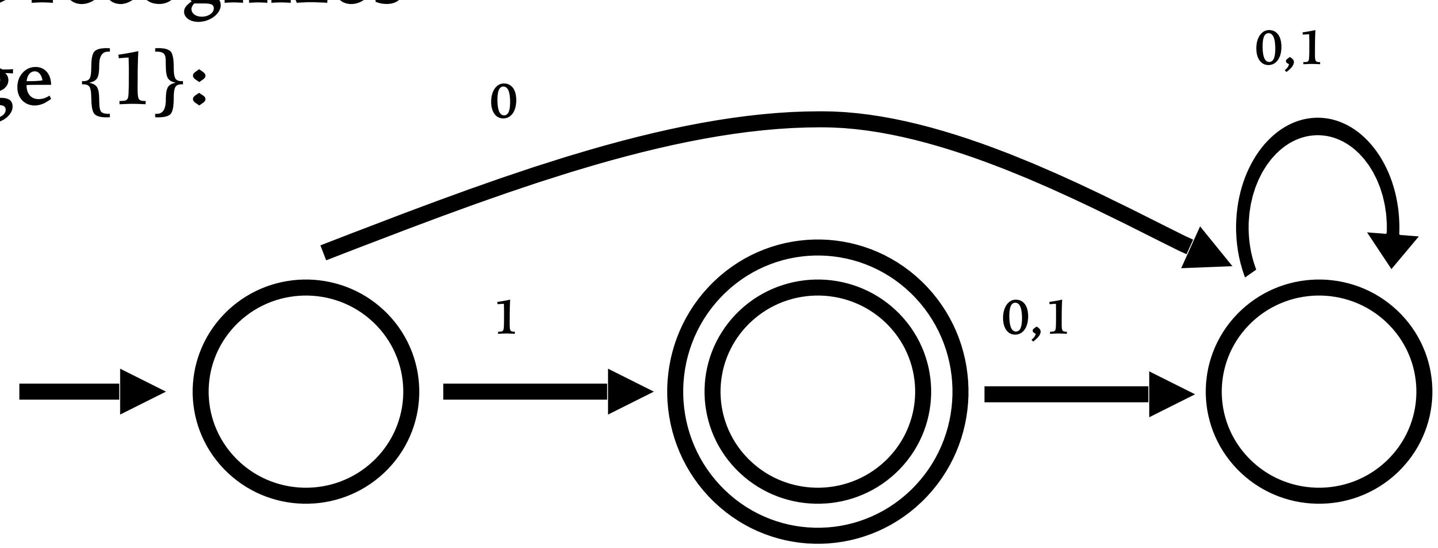
accept

NFAs ARE SIMPLER THAN DFAs

An NFA that recognizes the language {1}:



A DFA that recognizes the language {1}:



FROM NFA TO DFA

Theorem: Every NFA has an equivalent DFA

N is equivalent to M if L(N) = L (M)

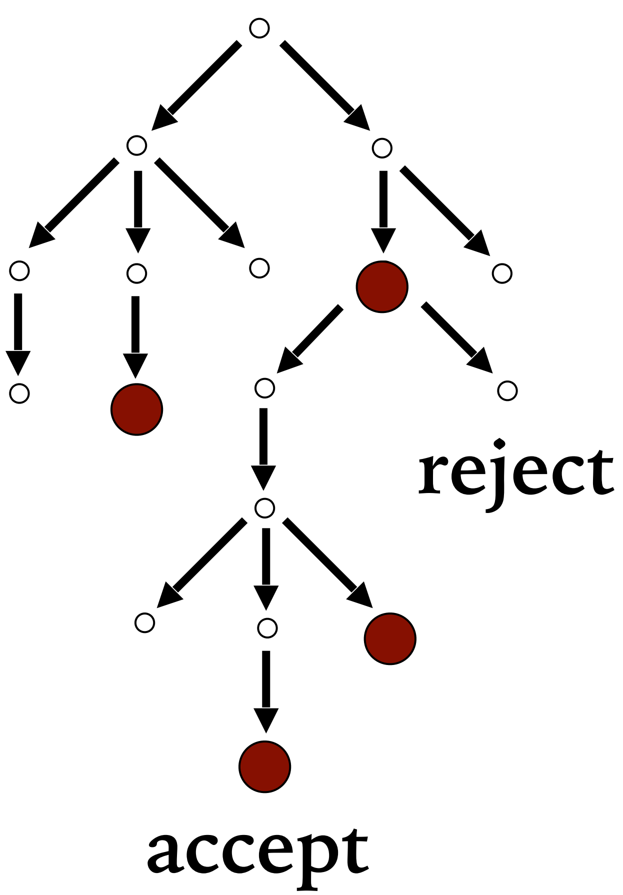
Corollary: A language is regular iff it is recognized by an NFA

Corollary: L is regular iff L^R is regular

FROM NFA TO DFA

Input: N = $(Q, \Sigma, \delta, Q_0, F)$

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



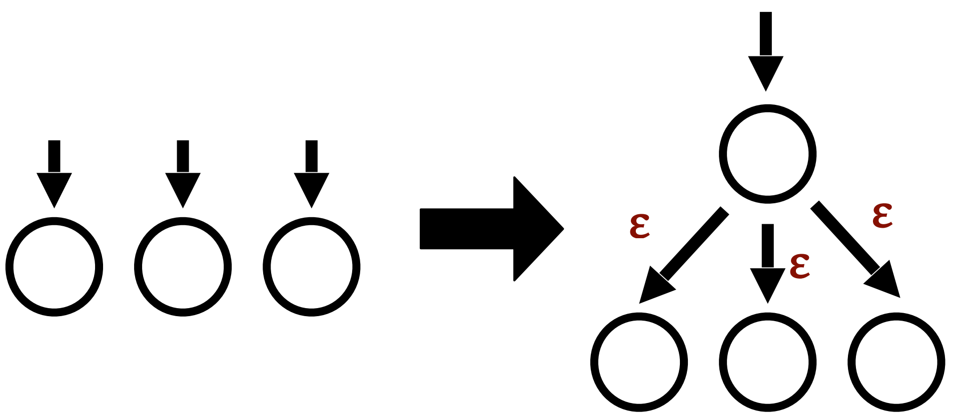
To learn if NFA accepts, we could do the computation in parallel, maintaining the set of states where all threads are.

Idea: $Q' = 2^Q$

FROM NFA TO DFA

Input: $N = (Q, \Sigma, \delta, Q_0, F)$

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



$\mathbf{Q'} = \mathbf{2^Q}$

$\delta' : \mathbf{Q'} \times \boldsymbol{\Sigma} \to \mathbf{Q'}$

$\delta'(\mathbf{R},\sigma) = \cup\, \boldsymbol{\varepsilon}(\, \delta(\mathbf{r},\sigma)\,), \mathbf{r} \in \mathbf{R}$

$q_0' = \boldsymbol{\varepsilon}(\mathbf{Q_0})$

$\mathbf{F'} = \{\, \mathbf{R} \in \mathbf{Q'} \mid \mathbf{f} \in \mathbf{R} \text{ for some } \mathbf{f} \in \mathbf{F}\,\}$

For **R** ⊆ **Q**, the **ε-closure of R**, **ε(R)** = {**q** that can be reached from some r ∈ **R** by traveling along zero or more **ε** arrows},

FROM NFA TO DFA

$$(01)^{+} \cup (010)^{+}$$

Regular Languages Closure Under Concatenation

$\boxed{\text{DFA1 -> DFA2}}$

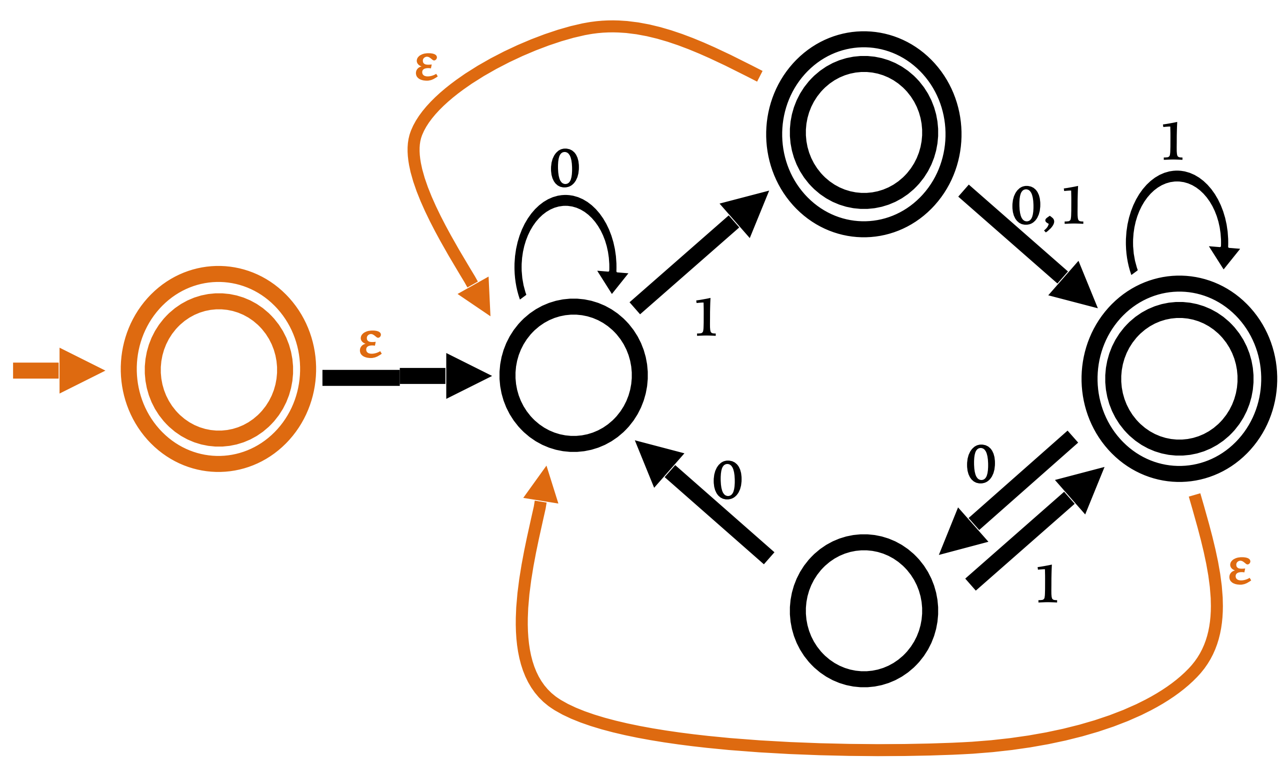
NFA

Given DFAs M_1 and M_2, construct NFA by connecting all accept states in M_1 to start states in M_2 .

Regular Languages Closure Under Star

Let **L** be a regular language and **M** be a DFA for **L**

We construct an NFA **N** that recognizes **L***



THE REGULAR OPERATIONS

- **Union:** $L_1 \cup L_2 = \{ w \mid w \in L_1 \text{ or } w \in L_2 \}$
- **Intersection:** $L_1 \cap L_2 = \{ w \mid w \in L_1 \text{ and } w \in L_2 \}$
- **Negation:** $\neg L = \{ w \in \Sigma^* \mid w \notin L \}$
- **Reverse:** $L^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in L \}$
- **Concatenation:** $L_1 \cdot L_2 = \{ vw \mid v \in L_1 \text{ and } w \in L_2 \}$
- **Star:** $L^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in L \}$

Are all languages regular?

Consider the language $L = \{ 0^n 1^n \mid n > 0 \}$

No finite automaton accepts this language.

Can you prove this?

Idea?

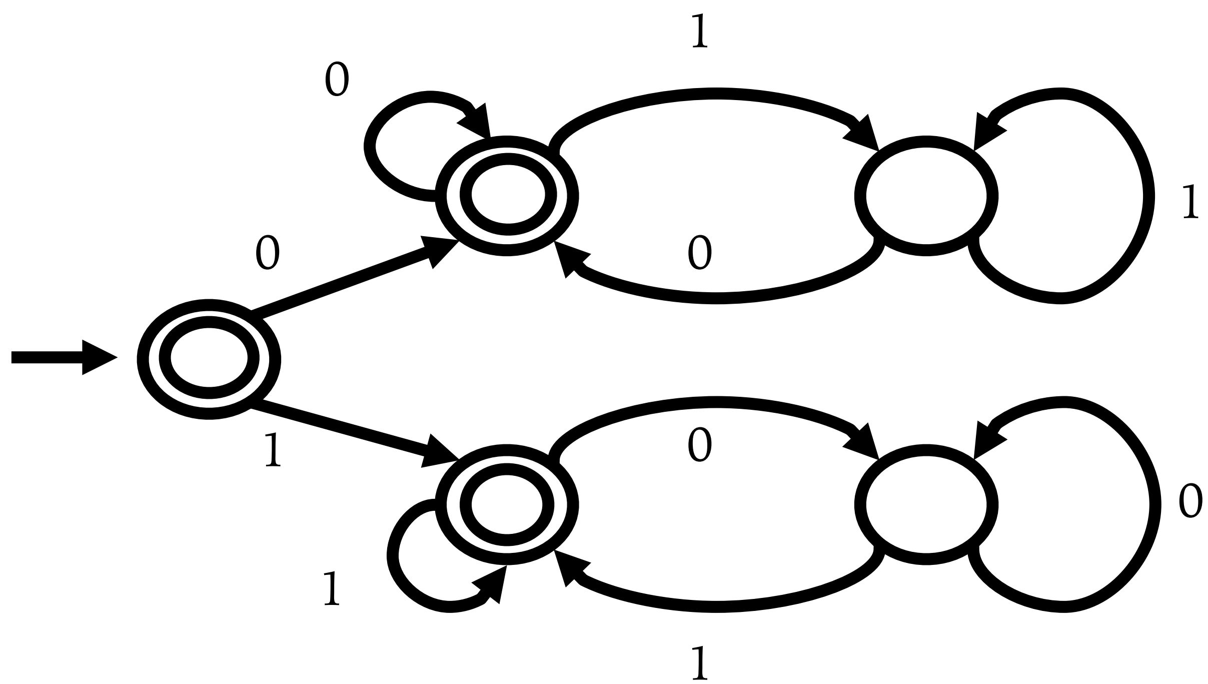
“$0^n 1^n$ is not regular. No machine has enough states to keep track of the number of 0’s it might encounter”

That is a fairly weak argument

Consider the following example…

L = { w | w has equal number of occurrences of 01 and 10}

No machine has enough states to keep track of the number of 01's it might encounter.



THE PUMPING LEMMA

Let L be a regular language with $|L| = \infty$

Then there exists a positive integer P such that

Any $x \in L, |x| \geq P$ can be written as

$$x = uvw$$

where:

1. $|v| > 0$
2. $|uv| \leq P$
3. $uv^iw \in L$ for any $i \geq 0$

THE PUMPING LEMMA

Assume $x \in L$ is such that $|x| \geq P$

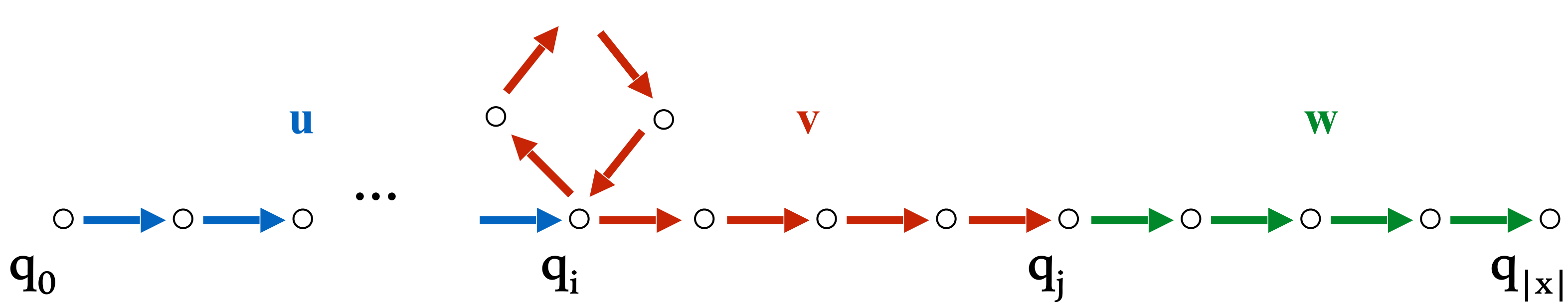
Let P be the number of states in M

We show $x = uvw$

where:

1. $|v| > 0$
2. $|uv| \leq P$
3. $uv^iw \in L$ for any $i \geq 0$

PIGEONHOLE: There must be $j > i$ such that $q_i = q_j$



THE PUMPING LEMMA

Assume $x \in L$ is such that $|x| \geq P$

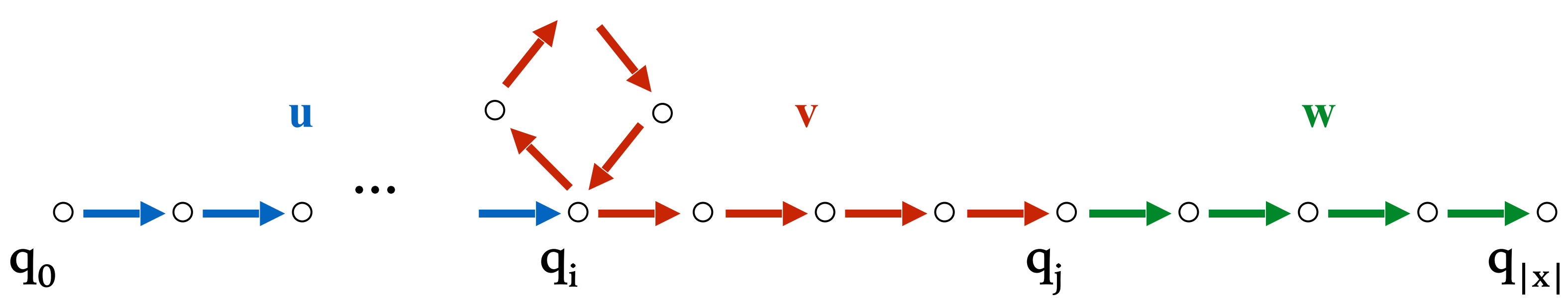
Let P be the number of states in M

We show $x = uvw$

where:

1. $|v| > 0$
2. $|uv| \leq P$
3. $uv^iw \in L$ for any $i \geq 0$

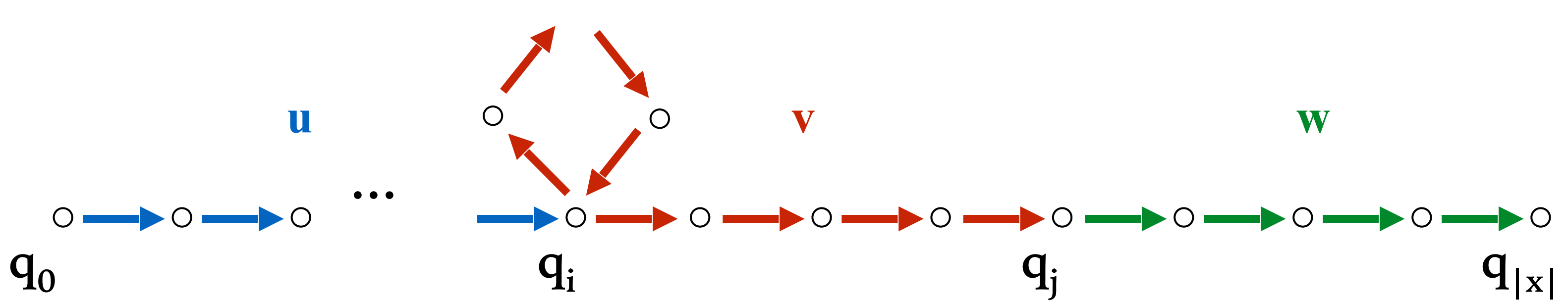
PIGEONHOLE: There must be $j > i$ such that $q_i = q_j$



THE PUMPING LEMMA

$$L = \{\, 0^n 1^n \mid n > 0 \,\}$$

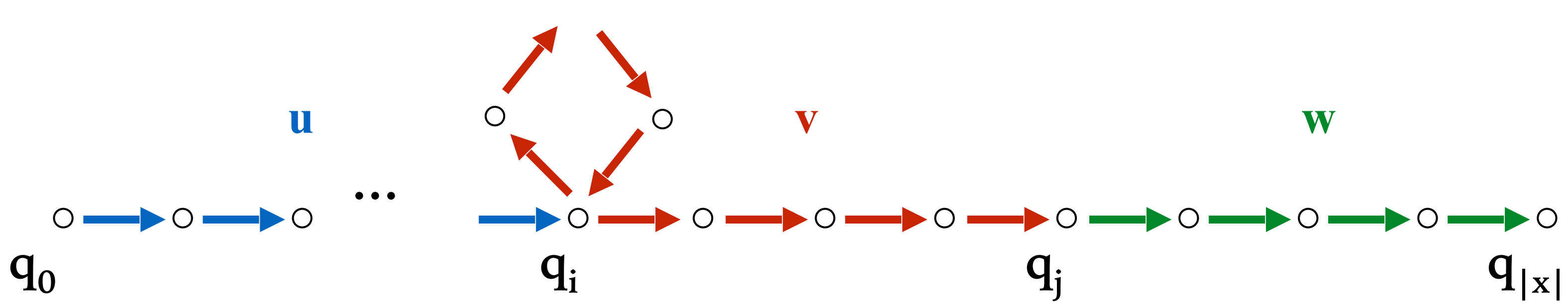
HINT: Assume L is regular, and try pumping.



THE PUMPING LEMMA

$L = \{ 0^n 1^n \mid n > 0 \}$

HINT: Assume L is regular, and try pumping.



References:

CMU FLAC: 15-453 Formal Languages, Automata and Computation, http://www.cs.cmu.edu/~lblum/flac/schedule.html, **Spring 2014**

Stanford CS154: Introduction to Automata and Complexity Theory, http://infolab.stanford.edu/~ullman/ialc/spr10/spr10.html, **Spring 2009**

John E. Hopcroft, etc., ***Introduction to Automata Theory, Languages, and Computation(Third Edition)*****. Pearson; 2006**