6.045

Lecture 2: Finite Automata and Nondeterminism

6.045

Problem Set 0 is coming out soon!
Look for it on Piazza

Recitations start tomorrow

6.045

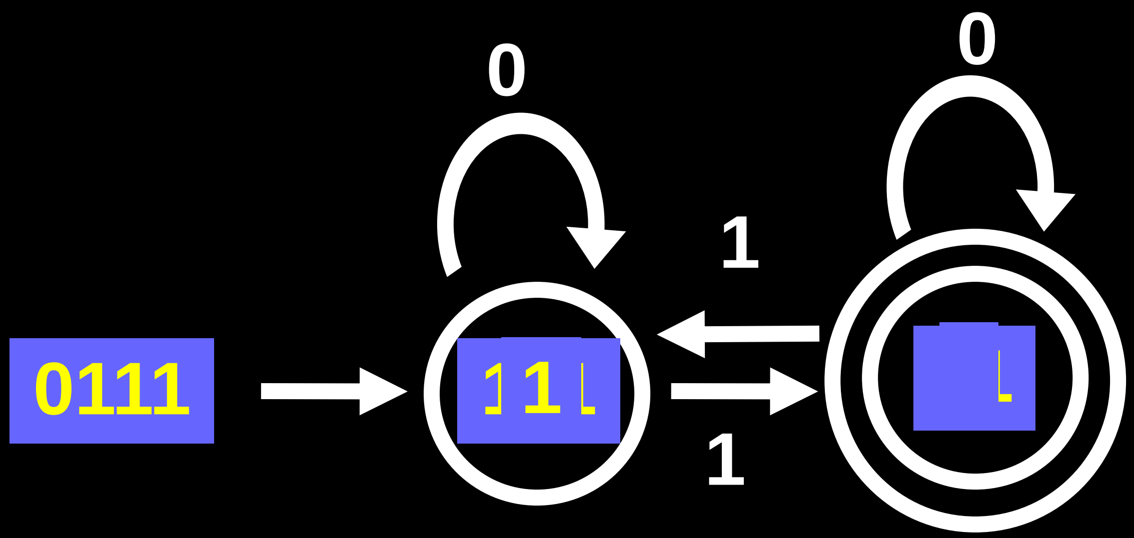
Hot Topics in Computing talk:

4:00 - 5:00pm
CSAIL's Patil Conference Room (32-G449).

Scott Aaronson on Quantum Computational Supremacy and Its Applications

Read string left to right

DFA with 2 states



The DFA **accepts** a string x if the process on x ends in a double circle

Above DFA accepts exactly those strings with an odd number of 1s

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

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

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

M accepts w if the (unique) path starting from q_0 with edge labels $w_1, \dots, w_n$ ends in a state in F.

M rejects w iff M does not accept w

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

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

Theorem: The union of two regular languages (over Σ) is also a regular language (over Σ)

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

Define M as follows:

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

$\quad = Q_1 \times Q_2$

$p_0 = (q_0, q'_0)$

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

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

How would you *prove* that this works?

Prove by induction on $|x|$:

M on x reaches state (p,q) $\Leftrightarrow$ M_1 on x reaches state p AND M_2 on x reaches state q

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

Idea: Simulate in parallel as before, but re-define $F = \{ (q_1, q_2) \mid q_1 \in F_1 \text{ AND } q_2 \in F_2 \}$

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 A is regular than so is $\neg$A,

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

Proof Idea: Flip the final and non-final states!

We can do much more…

The **Reverse** of a Language

Reverse of A:

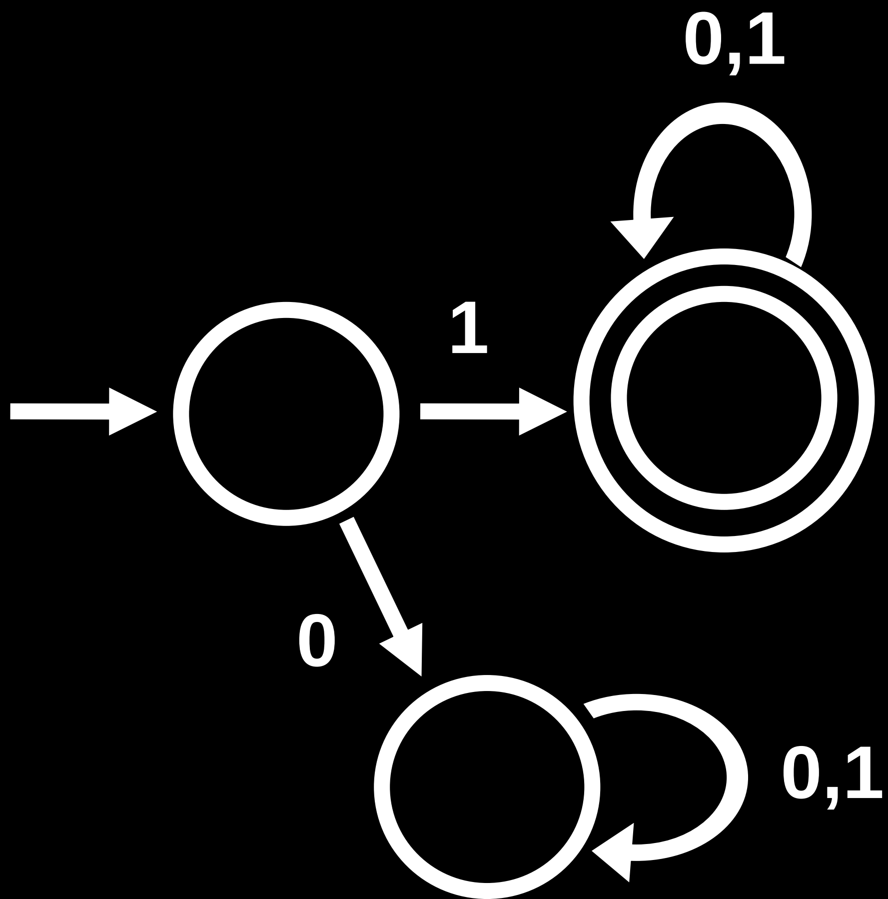
$A^R = \{ w_1 \cdots w_k \mid w_k \cdots w_1 \in A, w_i \in \Sigma\}$

Example: $\{0,10,110,0101\}^R = \{0,01,011,1010\}$

Intuition: If A is recognized by a DFA, then A^R is recognized by a "backwards" DFA that reads its strings from *right to left*!

Question: If A is regular, then is A^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 M reads its input 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"

For every **language that can be recognized by a DFA that reads its input from *right* to *left*, there is an "normal" left-to-right DFA recognizing that same language**

Counterintuitive! DFAs have finite memory…

Strings can be *much longer* than the number of states

Reversing DFAs?

Let L be a regular language,
let M be a DFA that recognizes L

We want to build a DFA M^R that recognizes L^R

Know: M accepts w $\Leftrightarrow$ w describes a directed path in M
from start state to an accept state

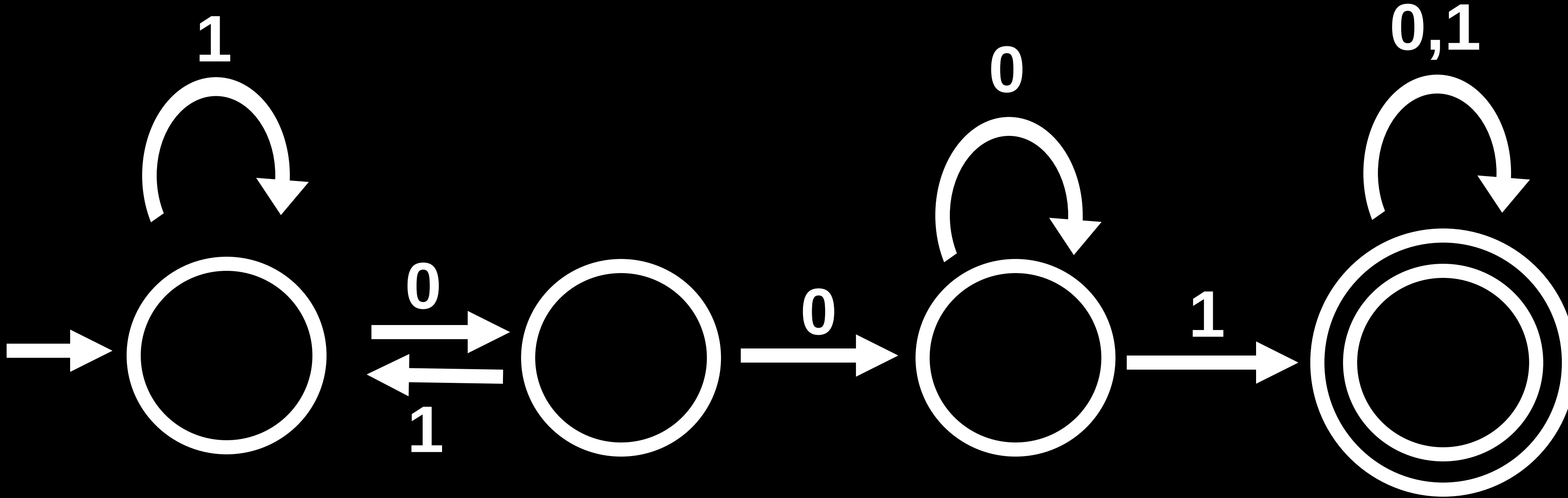
Want: M^R accepts w^R $\Leftrightarrow$ M accepts w

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

Problem: M^R IS NOT ALWAYS A DFA!

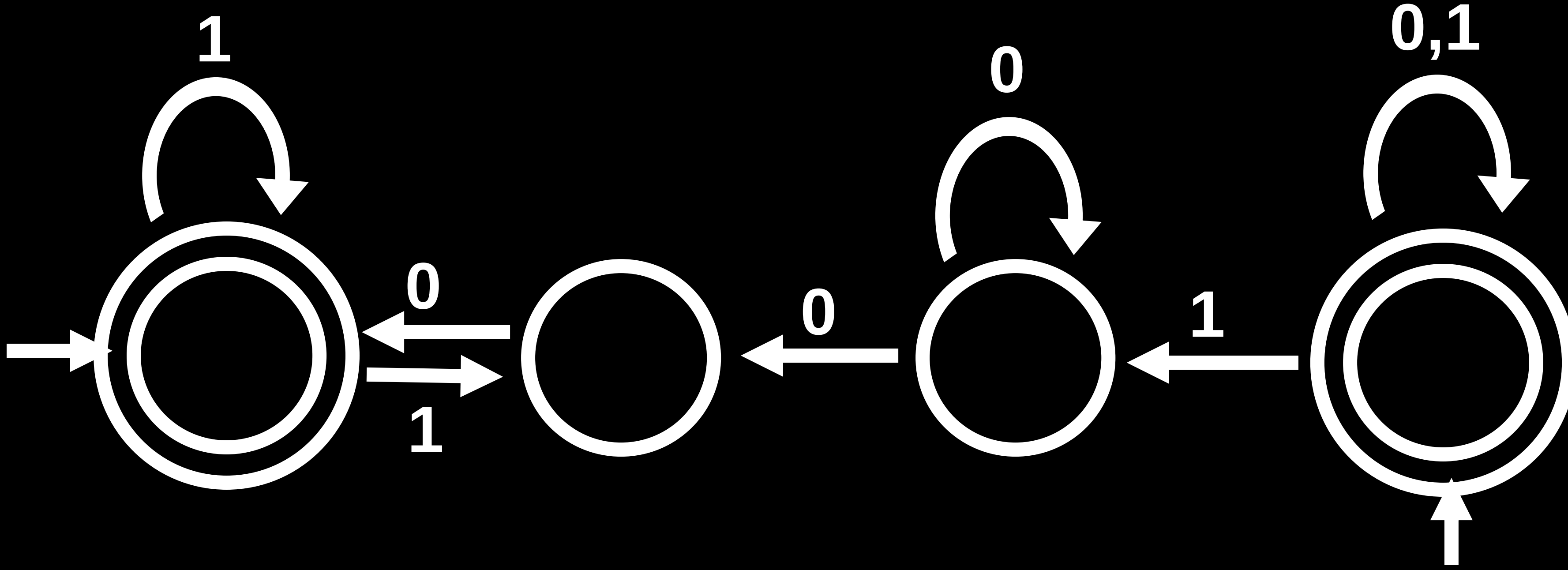
It could have many start states

Some states may have
more than one
transition for a given symbol,
or it may have no transition at all!



1
0
0,1
0
0
1
1

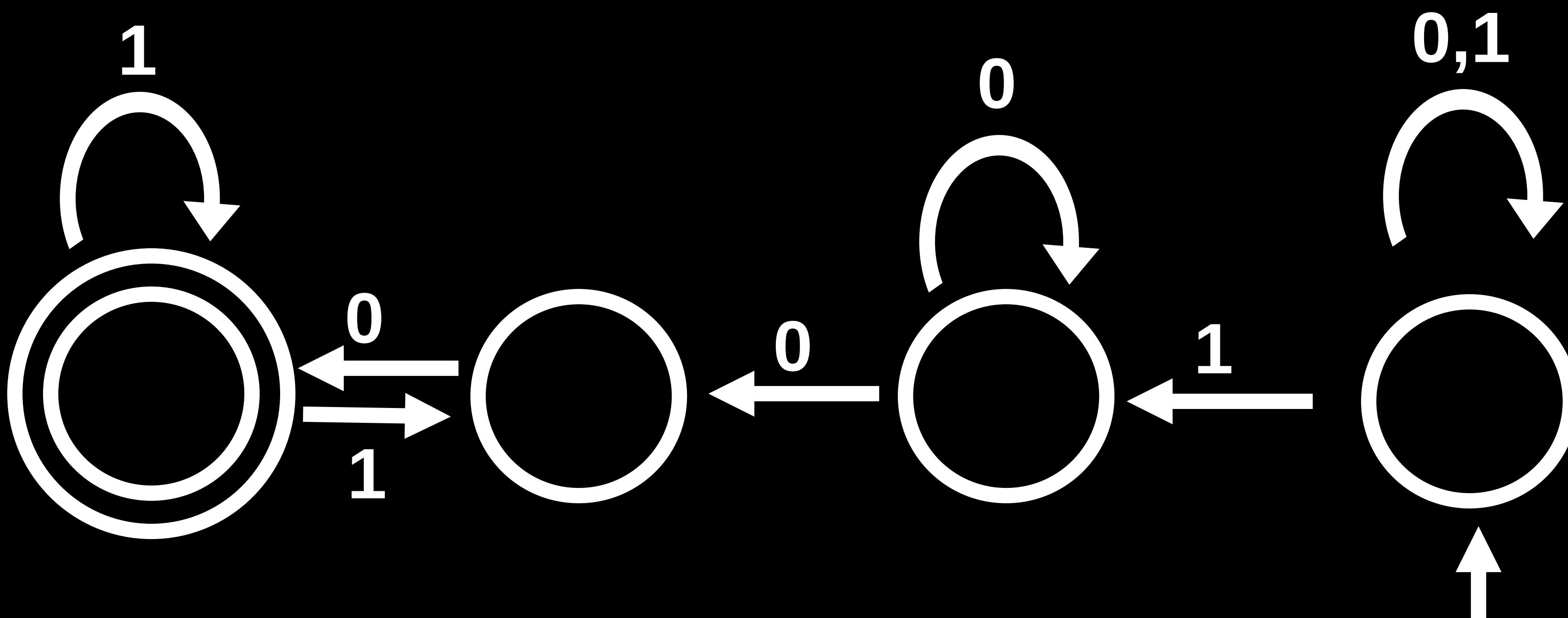
Non-deterministic Finite Automata (NFA)



What happens with 100?

We will say this new kind of machine accepts string x if *there is some path reading in x* that reaches *some accept state* from *some start state*

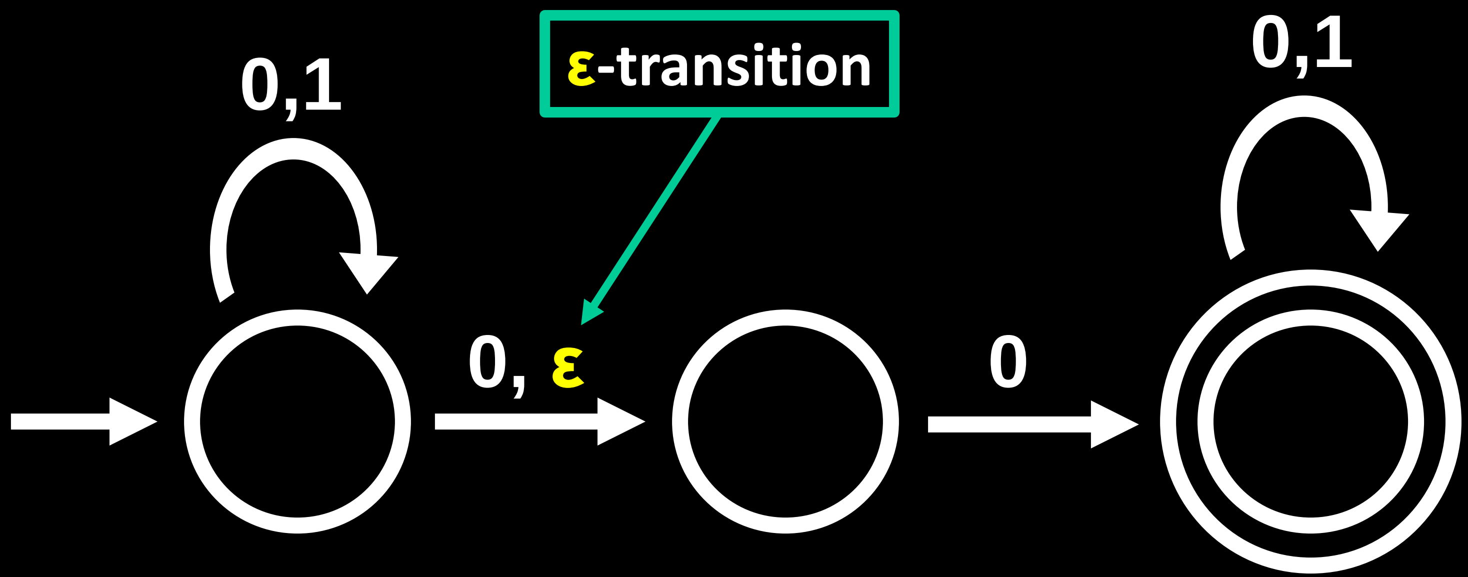
Non-deterministic Finite Automata (NFA)



Then, this machine recognizes: {w | w contains 100}

We will say this new kind of machine **accepts string x**
if *there is some path reading in x* that
reaches *some accept state* from *some start state*

Another Example of an NFA



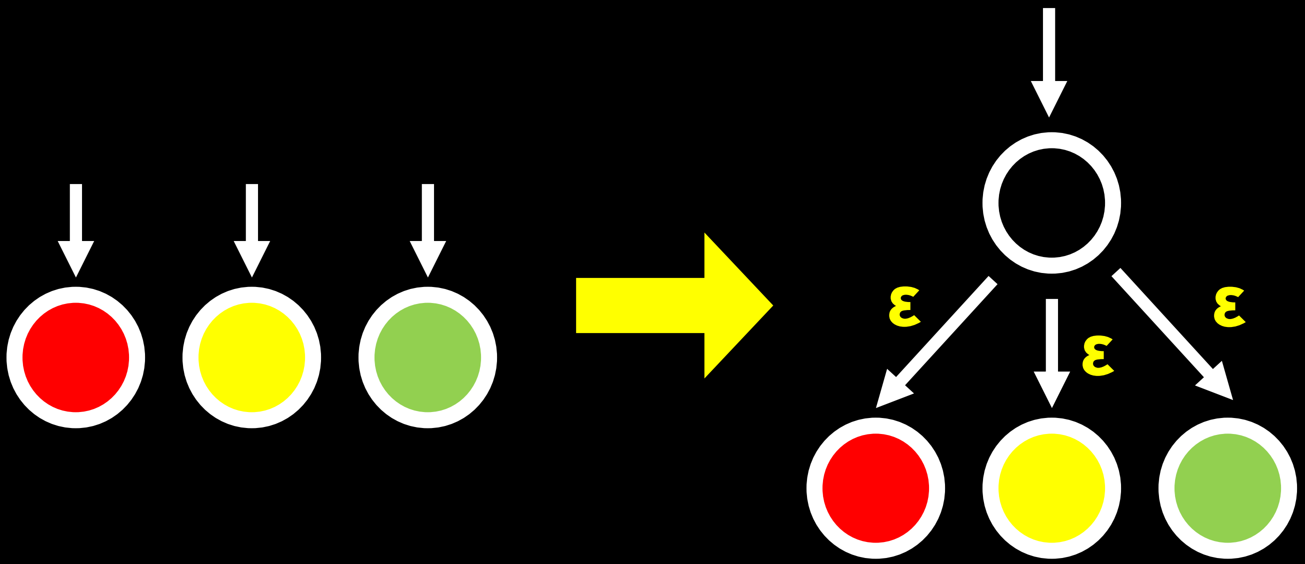
At each state, we'll allow ***any*** number (including zero) of out-arrows for letters $\sigma \in \Sigma$, **including ε**

Set of strings accepted by this NFA = {w | w contains a 0}

Multiple Start States

We allow *multiple* start states for NFAs, and Sipser allows only one

Can easily convert NFA with many start states into one with a single start state:



A *non-deterministic* finite automaton (NFA) is a 5-tuple $N = (Q, \Sigma, \delta, Q_0, F)$ where

Q is the set of states

Σ is the alphabet

Not deterministic!

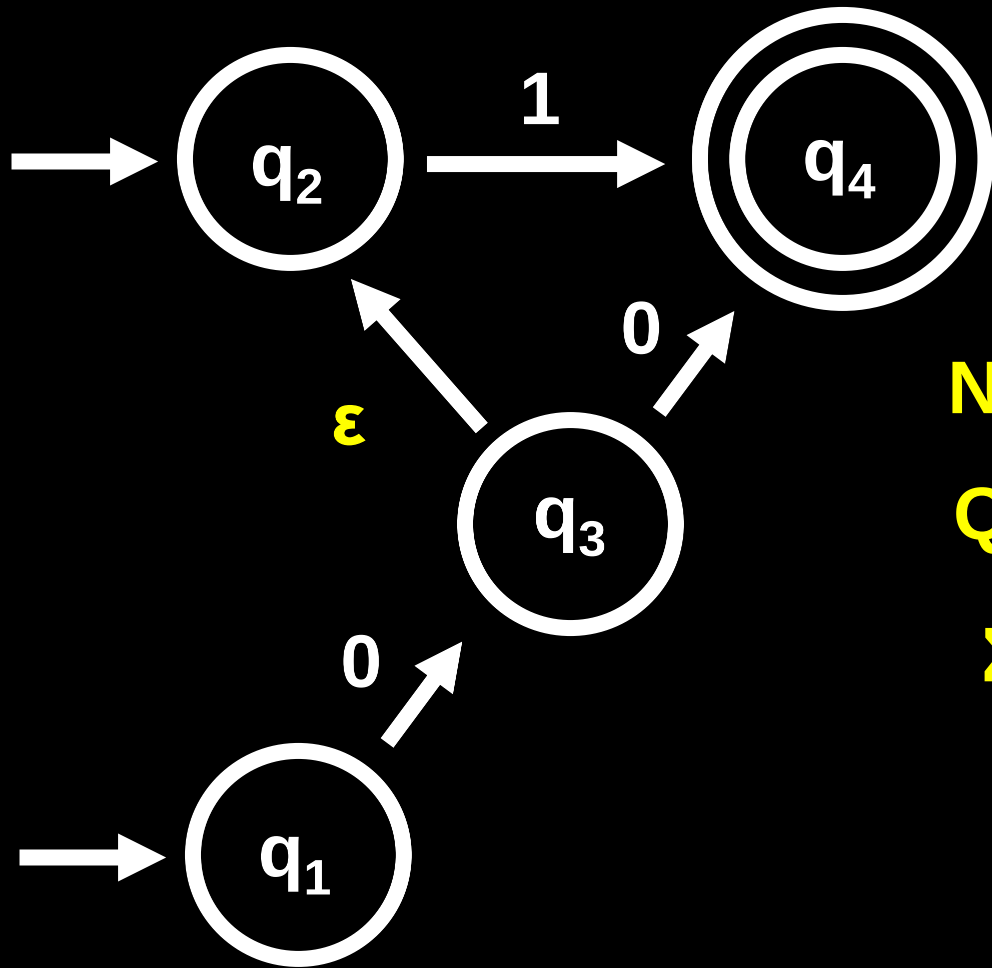
$\delta : Q \times \Sigma_\varepsilon \rightarrow 2^Q$ is the transition function

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

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

2^Q is the set of all possible subsets of Q

$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$



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

$Q = \{q_1, q_2, q_3, q_4\}$

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

$Q_0 = \{q_1, q_2\}$

$F = \{q_4\}$

$\delta(q_2,1) = \{q_4\}$ $\delta(q_4,1) = \varnothing$

$\delta(q_3,1) = \varnothing$

$\delta(q_1,0) = \{q_3\}$

Set of strings accepted = $\{1,00,01\}$

Def. Let $w \in \Sigma^*$. Let N be an NFA. **N accepts w** if there's a sequence of states $r_0, r_1, \ldots, r_k \in Q$ and w can be written as $w_1 \cdots w_k$ with $w_i \in \Sigma \cup \{\varepsilon\}$ such that

1. $r_0 \in Q_0$
2. $r_i \in \delta(r_{i-1}, w_i)$ for all $i = 1, \ldots, k$, and
3. $r_k \in F$

L(N) = the language recognized by N
= set of all strings that NFA N accepts

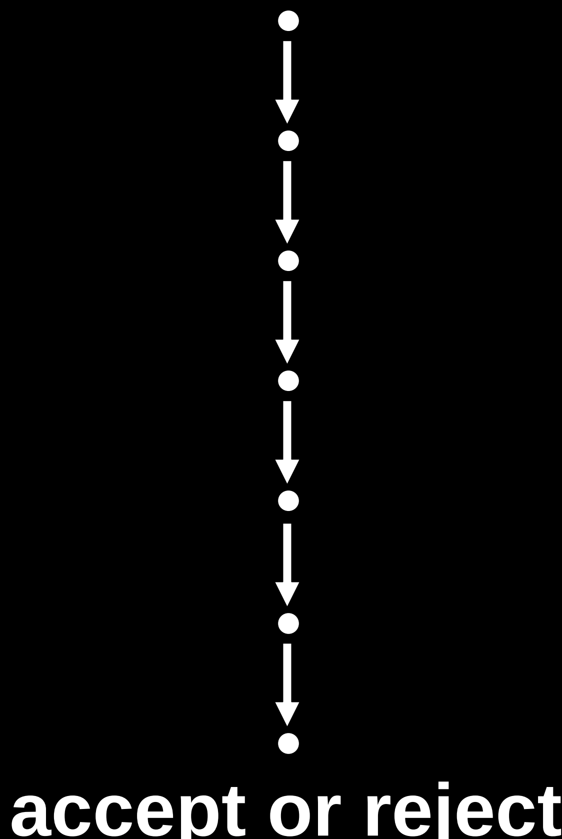
A language L' is recognized by an NFA N if L' = L(N).

Def. Let $w \in \Sigma^*$. Let N be an NFA.
N accepts w if there's some path of states in N,
from a state in Q_0 to a state in F,
with edges labeled $w_1 \cdots w_k$ with $w_i \in \Sigma \cup \{\varepsilon\}$
such that $w = w_1 \cdots w_k$

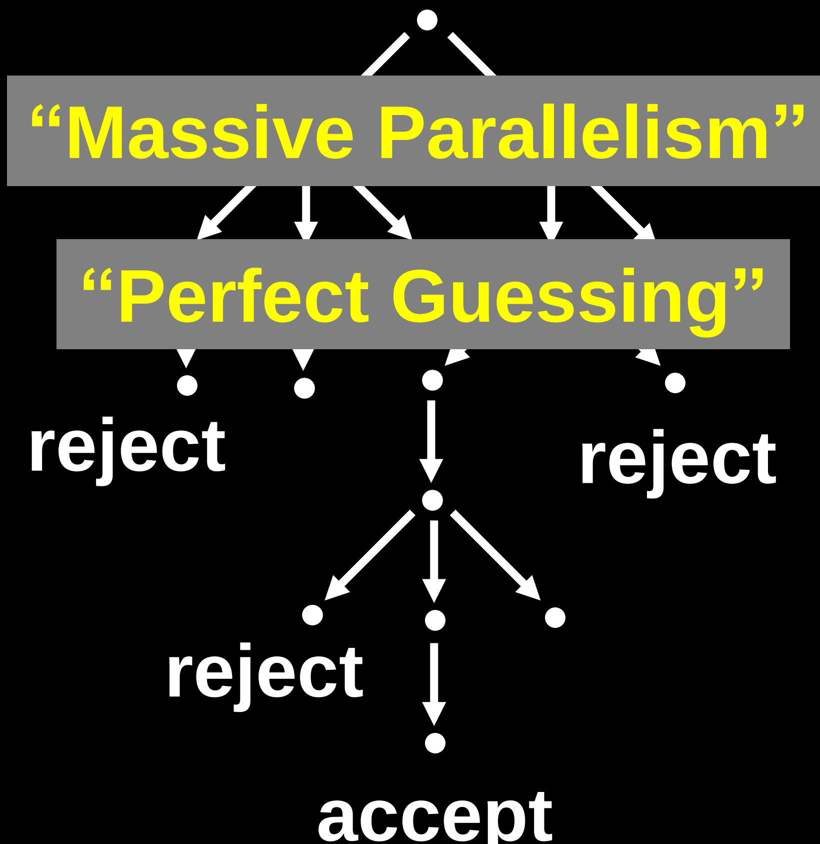
L(N) = the language recognized by N
= set of all strings that NFA N accepts

A language L' is recognized by an NFA N
if L' = L(N).

Deterministic Computation



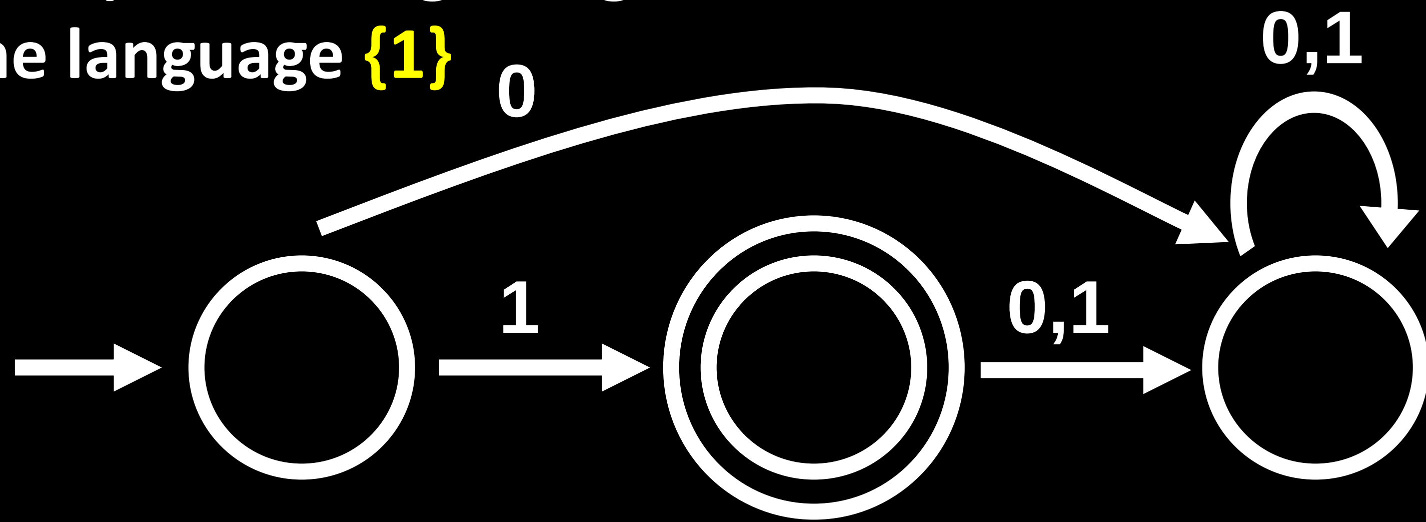
Non-Deterministic Computation



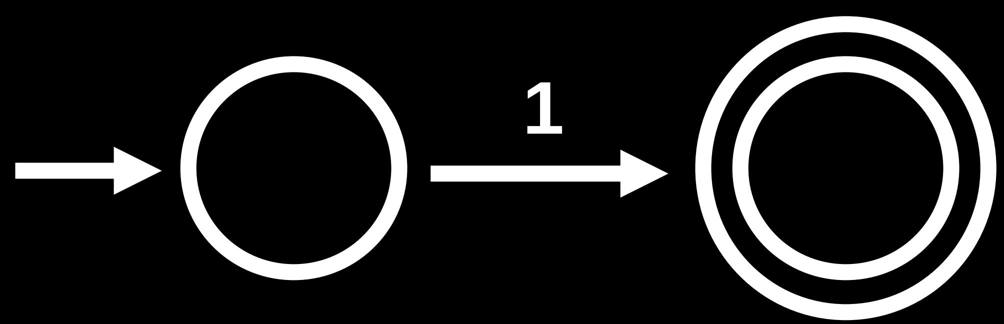
Are these equally powerful???

NFAs are generally simpler than DFAs

A (minimal) DFA recognizing the language {1}



An NFA recognizing the language {1}



Every NFA can be perfectly simulated by some DFA! 

Theorem: For every NFA N, there is a DFA M such that L(M) = L(N)

Corollary: A language A is regular if and only if A is recognized by an NFA

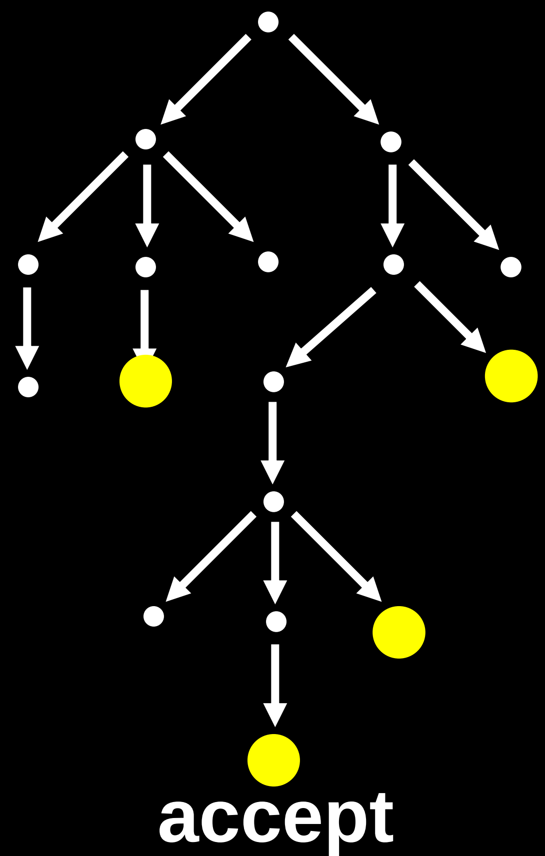
Corollary: A is regular iff A^R is regular

left-to-right DFAs $\equiv$ right-to-left DFAs

From NFAs to DFAs

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

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



To learn if NFA N accepts, our M will do the computation of N *in parallel*, maintaining the set of *all* possible states of N that can be reached so far

Idea:

Set $Q' = 2^Q$

From NFAs to DFAs: Subset Construction

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

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

$$Q' = 2^Q$$

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

For $S \in Q'$, $\sigma \in \Sigma$: $\delta'(S,\sigma) = \bigcup_{q \in S} \varepsilon(\,\delta(q,\sigma)\,)$ *

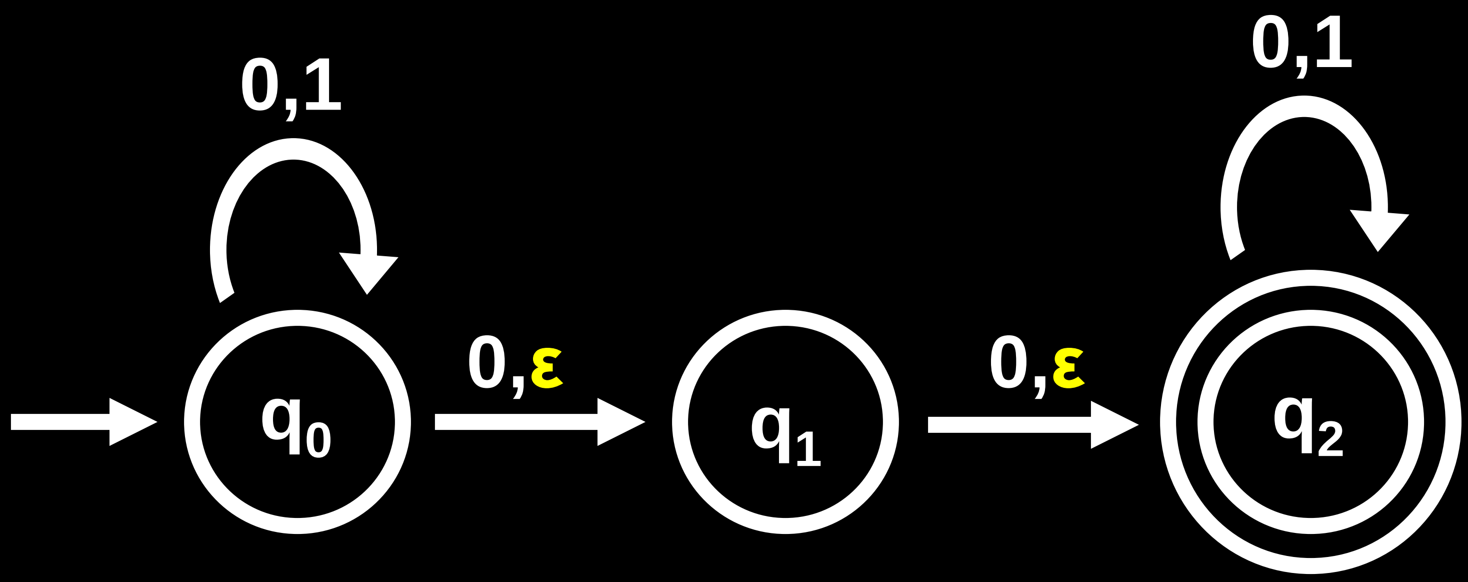
$$q_0' = \varepsilon(Q_0)$$

$F' = \{ S \in Q' \mid S \text{ contains } \textit{some } f \in F \}$

* For $S \subseteq Q$, the **ε-closure of S** is

$\varepsilon(S) = \{r \in Q$ reachable from some $q \in S$ by taking zero or more ε-transitions$\}$

Example of the ε-closure



$$\varepsilon(\{q_0\}) = \{q_0, q_1, q_2\}$$

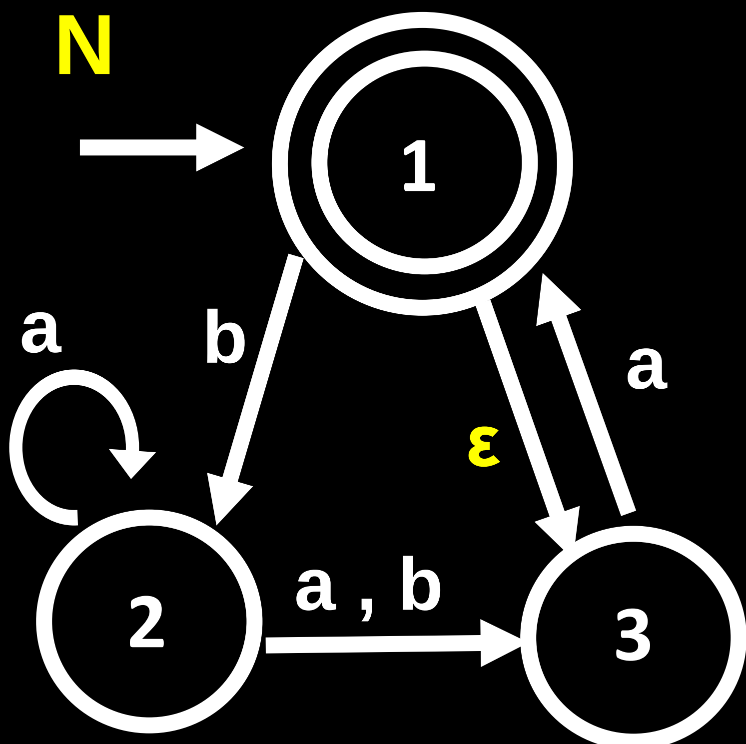
$$\varepsilon(\{q_1\}) = \{q_1, q_2\}$$

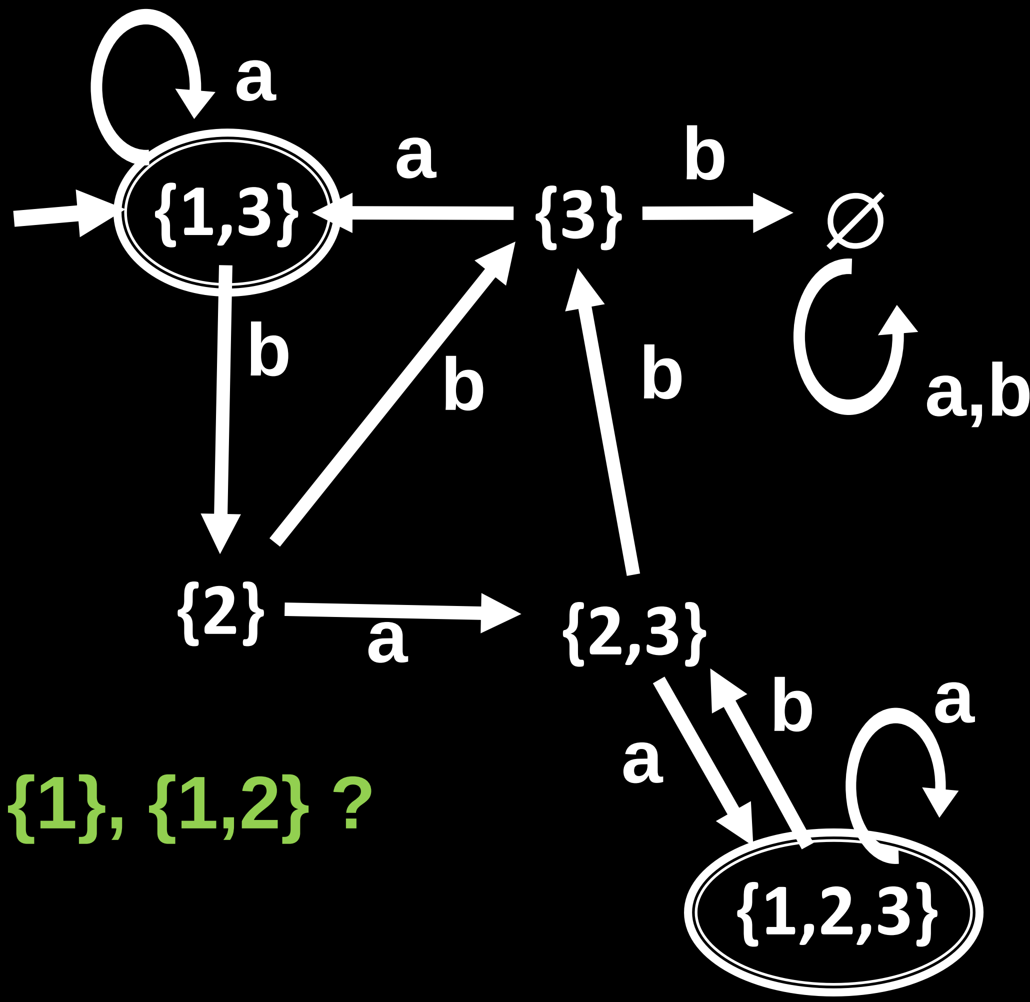
$$\varepsilon(\{q_2\}) = \{q_2\}$$

Given: NFA **N** = ({1,2,3}, {a,b}, δ , {1}, {1})

Construct: Equivalent DFA **M**

M = ($2^{\{1,2,3\}}$, {a,b}, δ′, {1,3}, …)





ε({1}) = {1,3}

{1}, {1,2} ?

Reverse Theorem for Regular Languages

The reverse of a regular language
is also a regular language

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**

Proof Sketch?

Given a DFA for a language L, "reverse" its arrows, and flip its start and accept states, getting an NFA. Convert that NFA back to a DFA!

Using NFAs in place of DFAs can make proofs about regular languages *much* easier!

Remember this on homework/exams!