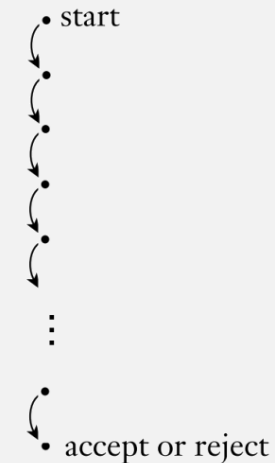


**UMB CS622**

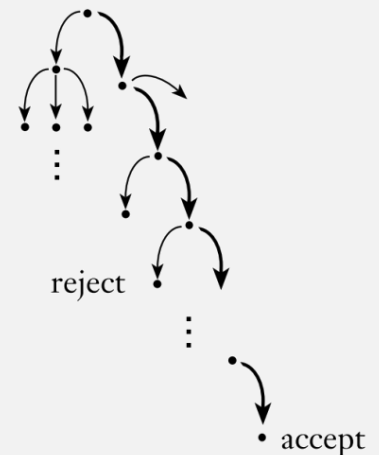
# Nondeterministic Finite Automata (NFAs)

Wednesday September 15, 2021

Deterministic  
computation



Nondeterministic  
computation



# Logistics

- HW1 due Sun 9/19 11:59pm EST
  - Upload solutions to Gradescope
  - LaTeX is great!
  - Handwritten and scanned/photo is perfectly fine
  - I must be able to read your answers!
    - Illegible solutions will not receive any credit
- Monday 9/13 lecture video posted
- Welcome new students!
  - Make sure to catch up ASAP

# *Last Time:* Finite State Automaton, a.k.a. DFAs

## DEFINITION 1.5

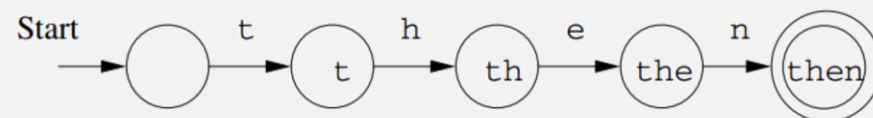
A *finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set called the *states*,
2.  $\Sigma$  is a finite set called the *alphabet*,
3.  $\delta: Q \times \Sigma \rightarrow Q$  is the *transition function*,
4.  $q_0 \in Q$  is the *start state*, and
5.  $F \subseteq Q$  is the *set of accept states*.

- Key characteristic:

- Has a **finite** number of states
- I.e., it's a computer with a finite amount of memory
  - Can't dynamically allocate

- Often used for **text matching**



# Combining DFAs?

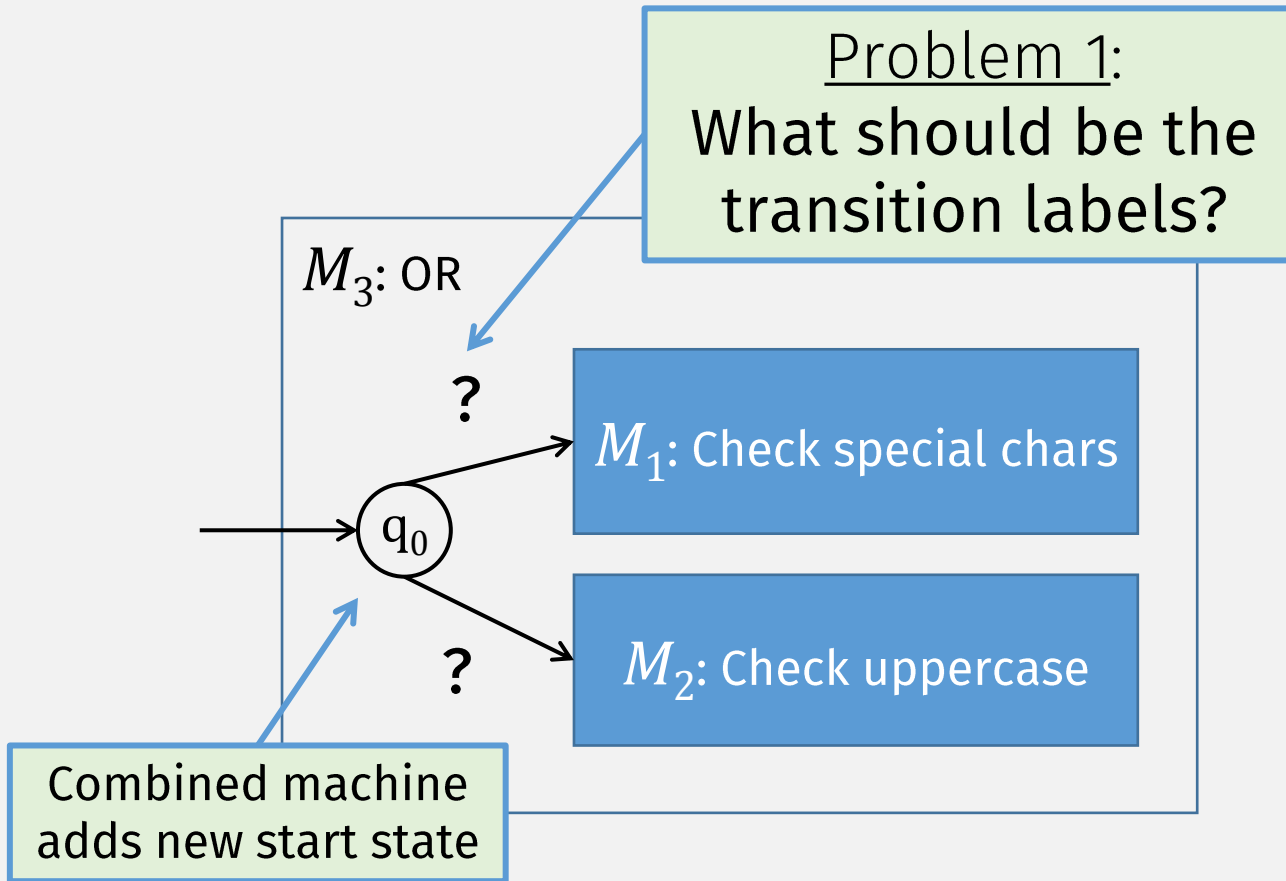
## Password Requirements

- » Passwords must have a minimum length of ten (10) characters - but more is better!
- » Passwords **must include at least 3** different types of characters:
  - » upper-case letters (A-Z) ← DFA
  - » lower-case letters (a-z) ← DFA
  - » symbols or special characters (% , & , \* , \$ , etc.) ← DFA
  - » numbers (0-9) ← DFA
- » Passwords cannot contain all or part of your email address ← DFA
- » Passwords cannot be re-used ← DFA

To match all requirements,  
can we combine smaller DFAs?

<https://www.umb.edu/it/password>

# Combining DFAs



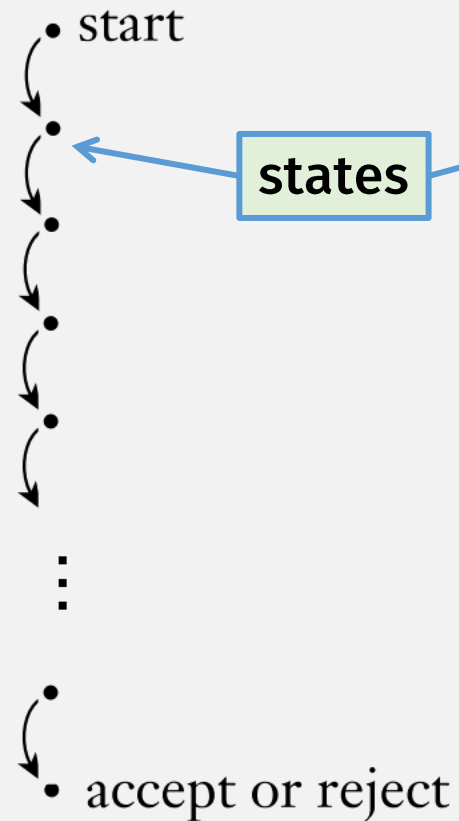
Problem 2:  
Once we enter one of the machines, can't go back to the other one!

**We need a different kind of machine!**

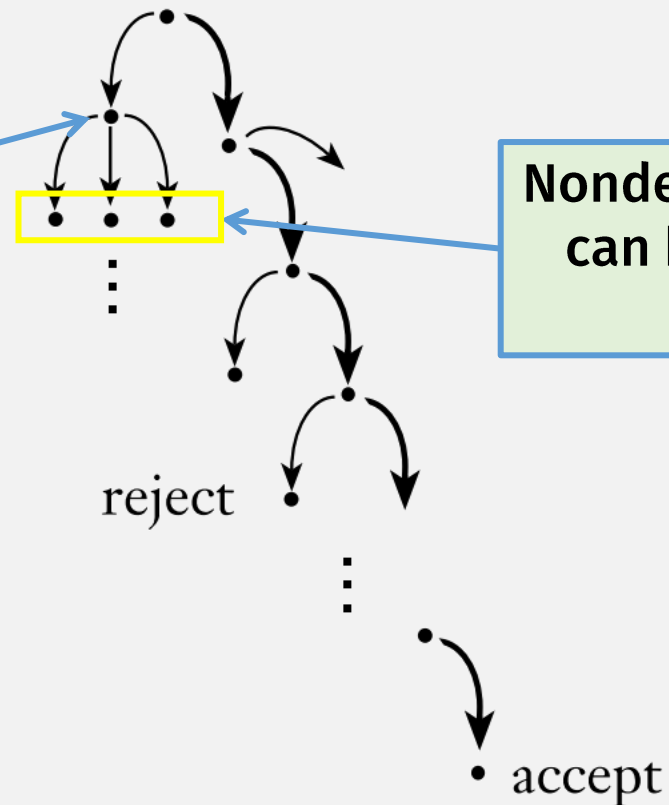
Idea: **nondeterminism** allows being in multiple states (i.e., multiple machines) at once!

# Nondeterminism

Deterministic  
computation



Nondeterministic  
computation



**Nondeterministic computation  
can be in multiple states at  
the same time**

# Nondeterministic Finite Automata (NFA)

## DEFINITION 1.37

Compare with DFA:

A *finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set called the *states*,
2.  $\Sigma$  is a finite set called the *alphabet*,
3.  $\delta: Q \times \Sigma \rightarrow Q$  is the *transition function*,
4.  $q_0 \in Q$  is the *start state*, and
5.  $F \subseteq Q$  is the *set of accept states*.

A *nondeterministic finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set of states,
2.  $\Sigma$  is a finite alphabet,
3.  $\delta: Q \times \Sigma_{\epsilon} \rightarrow \mathcal{P}(Q)$  is the transition function,
4.  $q_0 \in Q$  is the start state, and
5.  $F \subseteq Q$  is the set of accept states.

Difference

Power set, i.e. a transition results in set of states

# Power Sets

- A power set is the set of all subsets of a set
- Example:  $S = \{a, b, c\}$
- Power set of  $S =$ 
  - $\{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$
  - Note: includes the empty set!



# Nondeterministic Finite Automata (NFA)

## DEFINITION 1.37

A *nondeterministic finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set of states,
2.  $\Sigma$  is a finite alphabet,
3.  $\delta: Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$  is the transition function,
4.  $q_0 \in Q$  is the start state, and
5.  $F \subseteq Q$  is the set of accept states.

Transition label can be “empty”,  
i.e., machine can transition  
without reading input

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

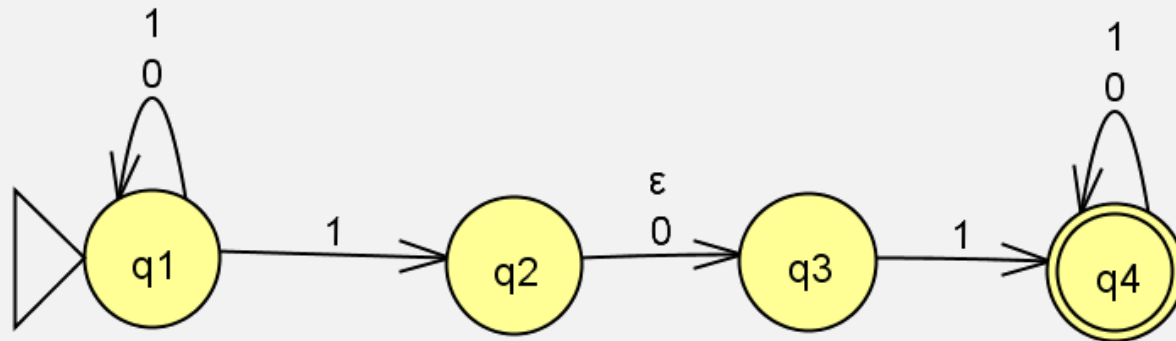
Compare with DFA:

A *finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set called the *states*,
2.  $\Sigma$  is a finite set called the *alphabet*,
3.  $\delta: Q \times \Sigma \longrightarrow Q$  is the *transition function*,
4.  $q_0 \in Q$  is the *start state*, and
5.  $F \subseteq Q$  is the *set of accept states*.

# NFA Example

- Come up with a formal description of the following NFA:



## DEFINITION 1.37

A *nondeterministic finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set of states,
2.  $\Sigma$  is a finite alphabet,
3.  $\delta: Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$  is the transition function,
4.  $q_0 \in Q$  is the start state, and
5.  $F \subseteq Q$  is the set of accept states.

The formal description of  $N_1$  is  $(Q, \Sigma, \delta, q_1, F)$ , where

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

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

3.  $\delta$  is given as

Result of transition  
can be empty set

	0	1	$\epsilon$
$q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
$q_2$	$\{q_3\}$	$\emptyset$	$\{q_3\}$
$q_3$	$\emptyset$	$\{q_4\}$	$\emptyset$
$q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

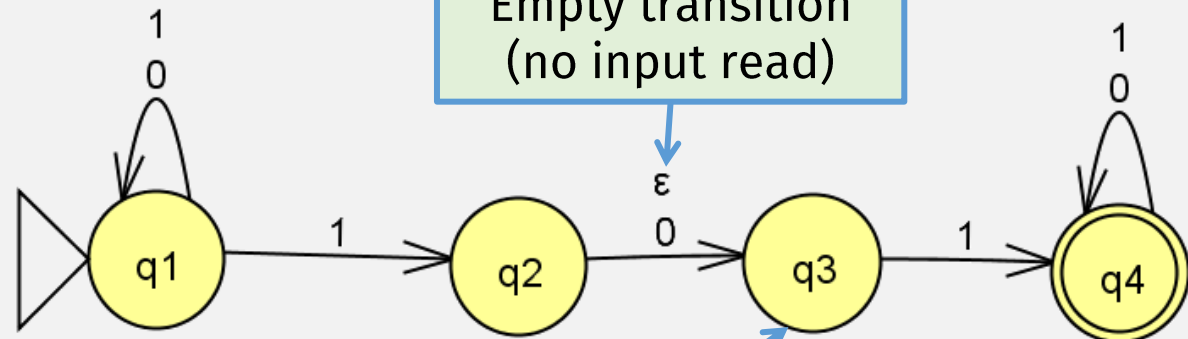
$$\delta: Q \times \Sigma_{\epsilon} \longrightarrow \mathcal{P}(Q)$$

Empty transition  
(no input read)

4.  $q_1$  is the start state, and

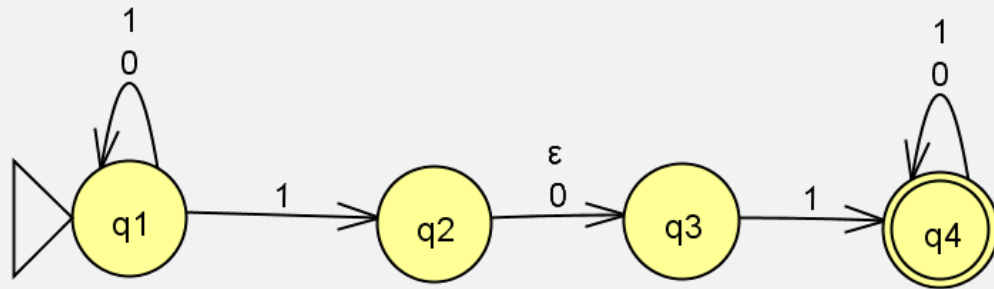
5.  $F = \{q_4\}$ .

Empty transition  
(no input read)



No zero (0) transition

# Running Programs, NFAs (JFLAP demo): **010110**



Symbol read

0

1

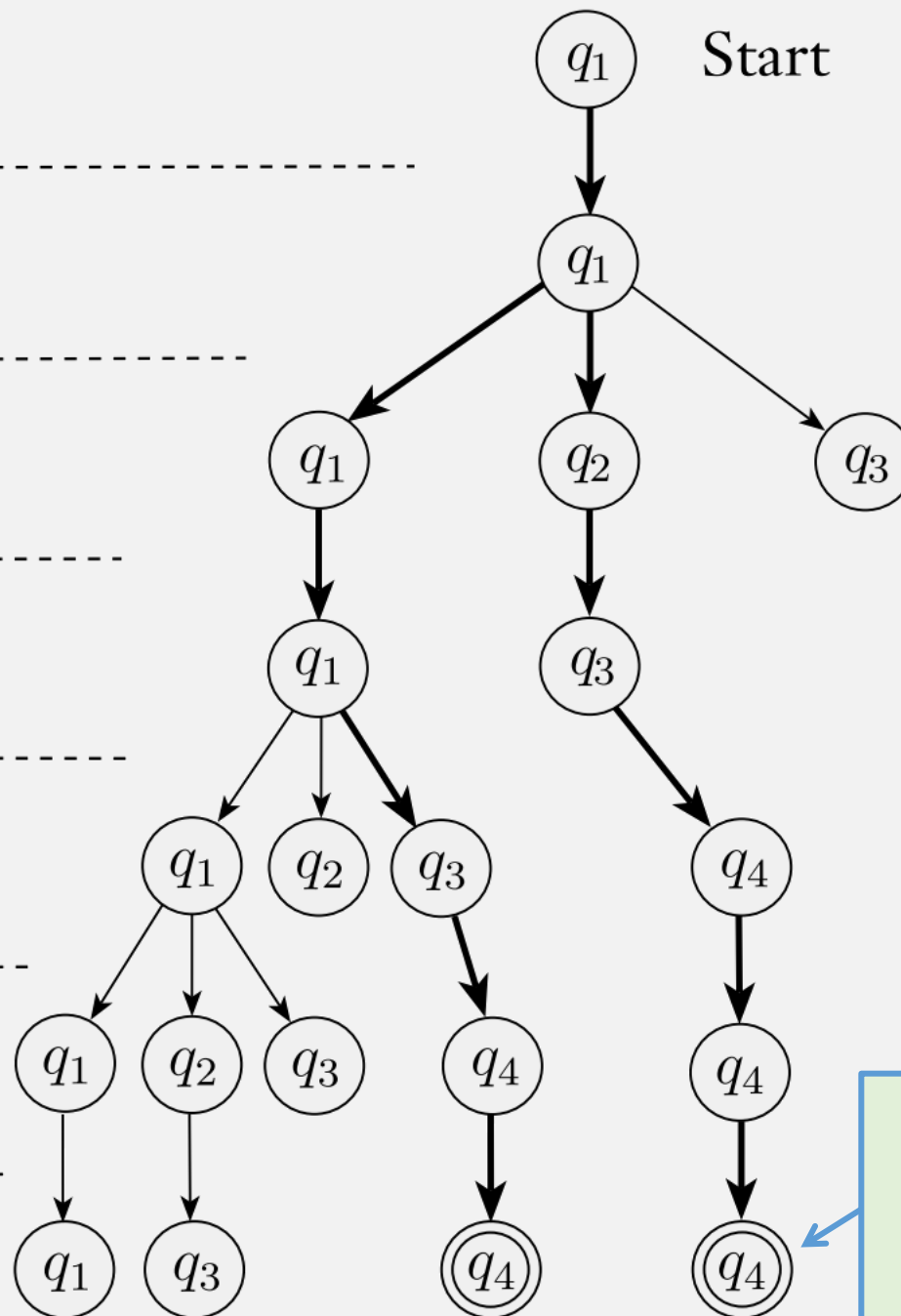
0

1

1

0

Start



A nondeterministic machine can be in multiple states at the same time!

This is an **accepting computation** because at least one path ends in an accept state

# NFAs vs DFAs

## DFAs

- Can only be in one state
- Transition:
  - Must read 1 char
- Acceptance:
  - If final state is accept state

## NFAs

- Can be in multiple states
- Transition
  - Can read no chars
  - i.e., empty transition
- Acceptance:
  - If one of final states is accept state

# Running an NFA Program: Formal Model

Define the extended transition function:  $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$

- *Inputs:*
  - Some beginning state  $q$  (not necessarily the start state)
  - Input string  $w = w_1 w_2 \cdots w_n$
- *Output:*
  - Set of ending states

(Defined recursively)

- Base case:  $\hat{\delta}(q, \epsilon) = \{q\}$

- Recursive case:

- If:  $\hat{\delta}(q, w') = \{q_1, \dots, q_k\}$

where  $w' \in \Sigma^* = w_1 \cdots w_{n-1}$   
and  $w_n \in \Sigma$

All except last char

Combine all possible state transitions for last char

- Then:  $\hat{\delta}(q, w'w_n) = \bigcup_{i=1}^k \delta(q_i, w_n)$

Last char

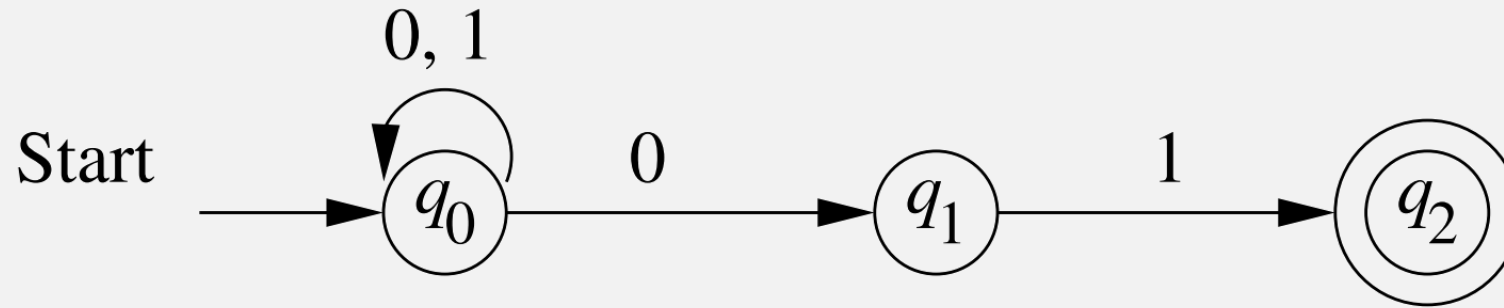
$$\hat{\delta}(q, \epsilon) = \{q\}$$

$$\hat{\delta}(q, w'w_n) = \bigcup_{i=1}^k \delta(q_i, w_n)$$

where  $w' \in \Sigma^* = w_1 \cdots w_{n-1}$   
and  $w_n \in \Sigma$

$$\hat{\delta}(q, w') = \{q_1, \dots, q_k\}$$

# NFA Extended delta Example



- $\hat{\delta}(q_0, \epsilon) =$

- $\hat{\delta}(q_0, 0) =$

- $\hat{\delta}(q_0, 00) =$

- $\hat{\delta}(q_0, 001) =$



# Adding Empty Transitions

- Define the set  $\varepsilon\text{-REACHABLE}(q)$ 
  - ... to be all states reachable from  $q$  via one or more empty transitions

(Defined recursively)

- Base case:  $q \in \varepsilon\text{-REACHABLE}(q)$

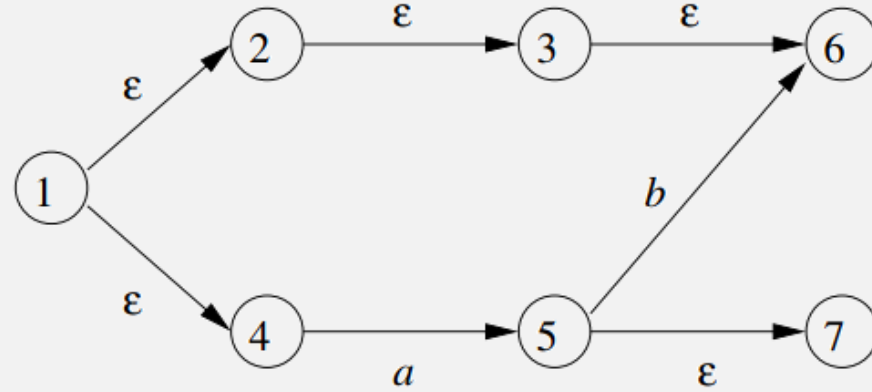
- Inductive case:

A state is in the reachable set if ...

$$\varepsilon\text{-REACHABLE}(q) = \{r \mid p \in \varepsilon\text{-REACHABLE}(q) \text{ and } r \in \delta(p, \varepsilon)\}$$

... there is an empty transition to it from another state in the reachable set

# $\epsilon$ -REACHABLE Example



$$\epsilon\text{-REACHABLE}(1) = \{1, 2, 3, 4, 6\}$$

# Running an NFA Program: Formal Model

Define the extended transition function:  $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$

- *Inputs:*
  - Some beginning state  $q$  (not necessarily the start state)
  - Input string  $w = w_1 w_2 \cdots w_n$
- *Output:*
  - Set of ending states

(Defined recursively)

- Base case:  $\hat{\delta}(q, \epsilon) = \epsilon\text{-REACHABLE}(q)$

- Recursive case:

- If:  $\hat{\delta}(q, w') = \{q_1, \dots, q_k\}$

where  $w' \in \Sigma^* = w_1 \cdots w_{n-1}$   
and  $w_n \in \Sigma$

- Then:  $\hat{\delta}(q, w'w_n) = \epsilon\text{-REACHABLE}\left(\bigcup_{i=1}^k \delta(q_i, w_n)\right)$

transitions for last char

Reminder:

DFA  $M$  *accepts*  $w$  if  
 $\hat{\delta}(q_0, w)$  is in  $F$

# An NFA's Language

- For NFA  $N = (Q, \Sigma, \delta, q_0, F)$   $N$  *accepts*  $w$  if  $\hat{\delta}(q_0, w) \cap F \neq \emptyset$ 
  - i.e., if the final states have at least one accept state
- Language of  $N = L(M) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$
- Q: How does an NFA's language relate to regular languages
  - Reminder: A language is regular if a DFA recognizes it

# NFAs and Regular Languages

## Theorem:

- A language  $A$  is regular **if and only if** some NFA  $N$  recognizes it.

# How to Prove a Theorem: $X \Leftrightarrow Y$

- $X \Leftrightarrow Y$  = “ $X$  if and only if  $Y$ ” =  $X$  iff  $Y$  =  $X \Leftrightarrow Y$
- Proof at minimum has 2 parts:
  1.  $\Rightarrow$  if  $X$ , then  $Y$ 
    - i.e., assume  $X$ , then use it to prove  $Y$
    - “forward” direction
  2.  $\Leftarrow$  if  $Y$ , then  $X$ 
    - i.e., assume  $Y$ , then use it to prove  $X$
    - “reverse” direction

# NFAs and Regular Languages

## Theorem:

- A language  $A$  is regular **if and only if** some NFA  $N$  recognizes it.
- Must prove:
  - $\Rightarrow$  If  $A$  is regular, then some NFA  $N$  recognizes it
    - Easier
    - We know: if  $A$  is regular, then a **DFA** recognizes it.
    - Easy to convert DFA to an NFA! (see HW2)
  - $\Leftarrow$  If an NFA  $N$  recognizes  $A$ , then  $A$  is regular.
    - Harder
    - Idea: Convert NFA to DFA

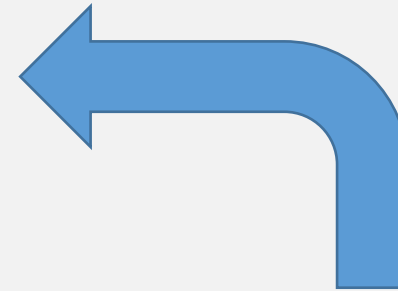
# How to convert NFA→DFA?

A *finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set called the *states*,
2.  $\Sigma$  is a finite set called the *alphabet*,
3.  $\delta: Q \times \Sigma \longrightarrow Q$  is the *transition function*,
4.  $q_0 \in Q$  is the *start state*, and
5.  $F \subseteq Q$  is the *set of accept states*.

## Proof idea:

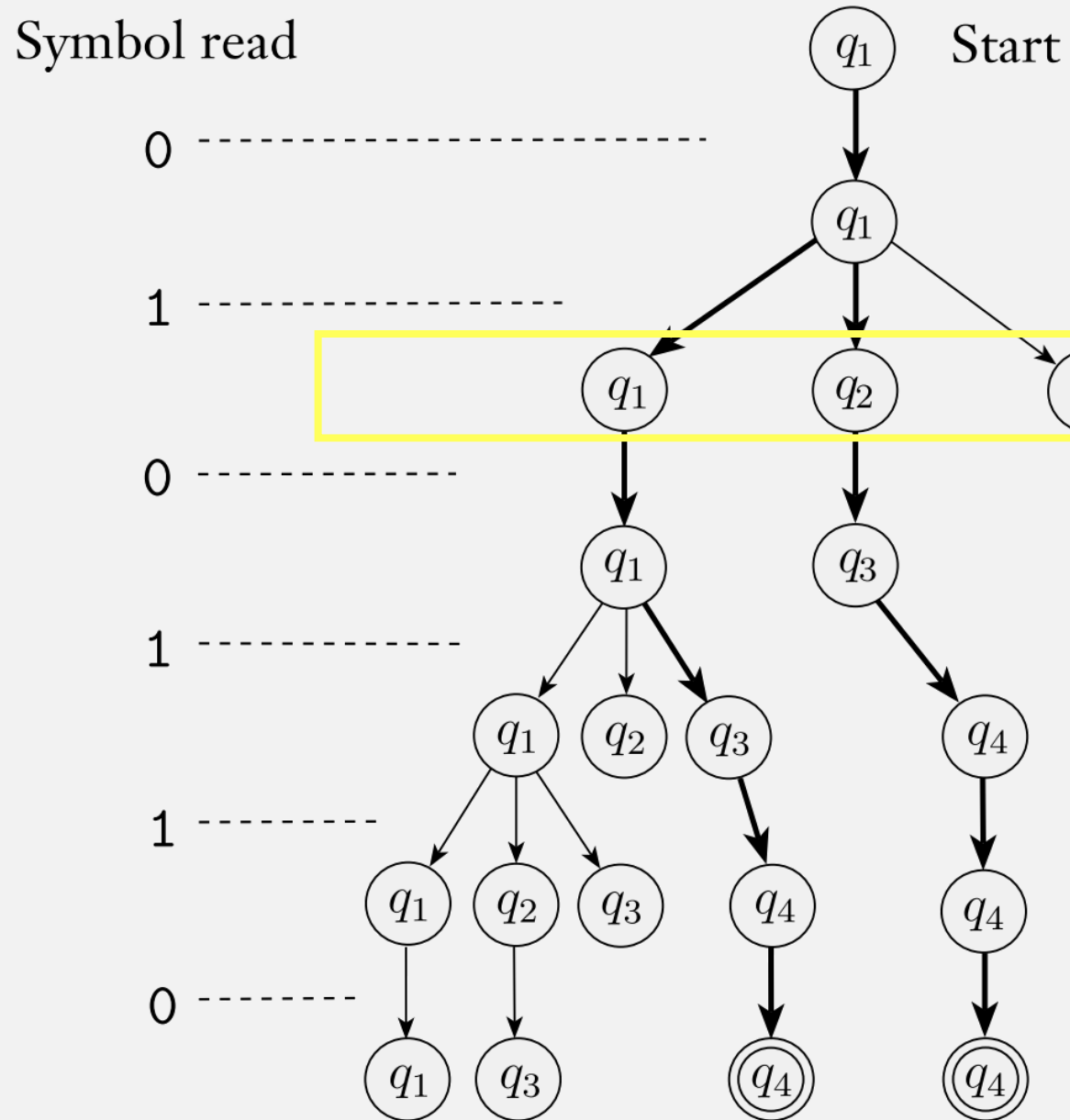
Let each “state” of the DFA be a set of states in the NFA



A *nondeterministic finite automaton* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set of states,
2.  $\Sigma$  is a finite alphabet,
3.  $\delta: Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$  is the transition function,
4.  $q_0 \in Q$  is the start state, and
5.  $F \subseteq Q$  is the set of accept states.





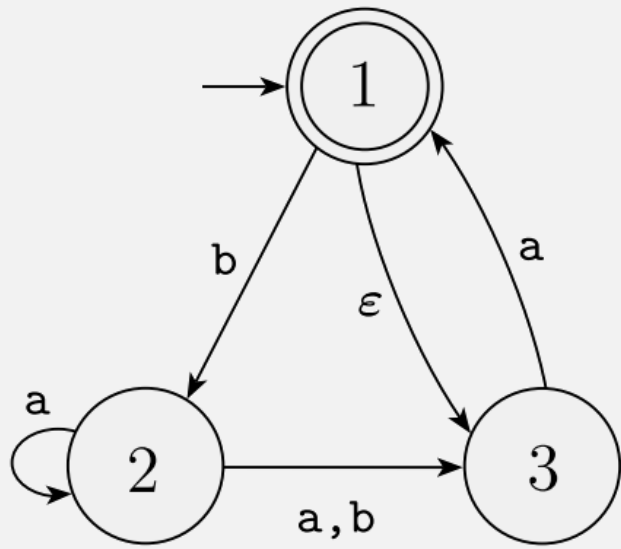
In a DFA, all these states at each step must be only **one** state

So design a state in the DFA to be a **set of NFA states!**

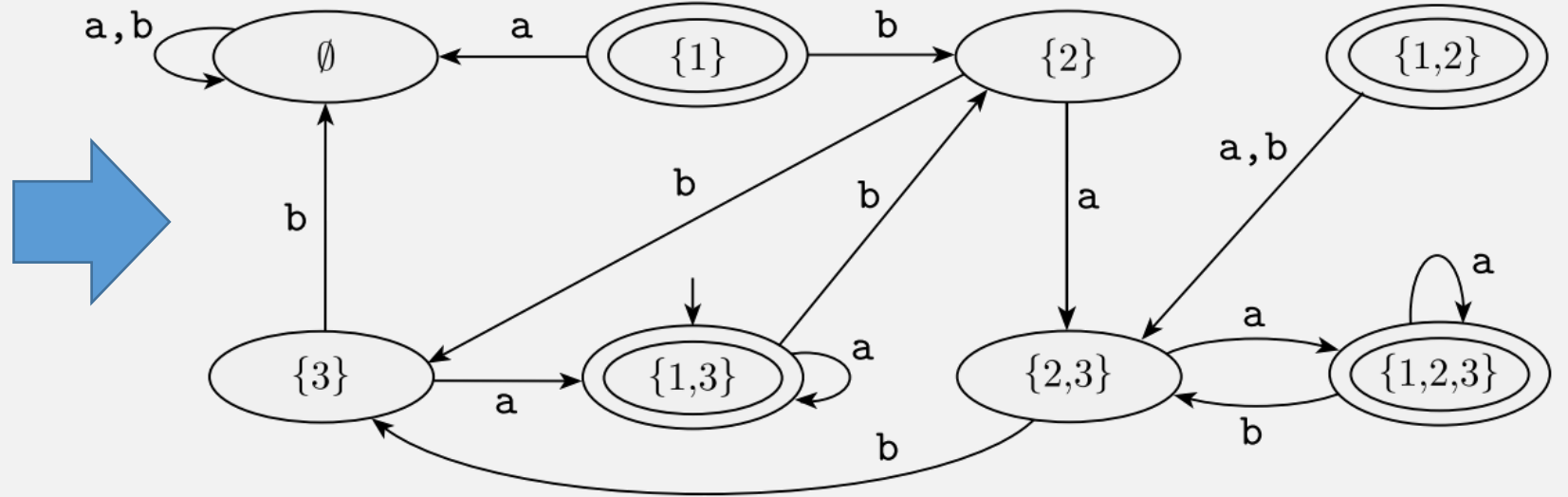
# Convert NFA→DFA, Formally

- Let NFA  $N = (Q, \Sigma, \delta, q_0, F)$
- An equivalent DFA  $M$  has states  $Q' = \mathcal{P}(Q)$  (power set of  $Q$ )

# Example:



The NFA  $N_4$



A DFA  $D$  that is equivalent to the NFA  $N_4$

Is this correct?

# NFA→DFA

Have:  $N = (Q, \Sigma, \delta, q_0, F)$

Want to: construct a **DFA**  $M = (Q', \Sigma, \delta', q_0', F')$

1.  $Q' = \mathcal{P}(Q)$ . A state for  $M$  is a set of states in  $N$

2. For  $R \in Q'$  and  $a \in \Sigma$ ,  $R = \text{a state in } M = \text{a set of states in } N$

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

To compute next state for  $R$ ,  
compute next states of each NFA state  $r$  in  $R$ ,  
then union results into one set

3.  $q_0' = \{q_0\}$

4.  $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

# NFA→DFA Proof of Correctness

- Let  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
- And let  $\text{NFA} \rightarrow \text{DFA}(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$
- Correctness criteria:  $L(N) = L(D)$
- We will prove a stronger statement:  $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ 
  - That is, for all strings  $w$ , the DFA and NFA end in the same set of states

# NFA→DFA Proof of Correctness

- Let  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
- And let  $\text{NFA} \rightarrow \text{DFA}(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

Theorem:  $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$

This produces a set bc we defined states to be sets of states

Proof: (by induction on length of  $w$ )

This produces a set bc of the definition of NFAs

- Base case  $w = \epsilon$   $\hat{\delta}_D(\{q_0\}, \epsilon)$  and  $\hat{\delta}_N(q_0, \epsilon)$  are  $\{q_0\}$

- Inductive case  $w = xa$

- IH:  $\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$ , call this set of states  $R$

- NFA last step (from  $\delta_N$  definition)  $\bigcup_{r \in R} \delta_N(r, a)$

- DFA last step (from NFA→DFA definition)  $\bigcup_{r \in R} \delta_D(r, a)$

Go back and review previous definitions to confirm

# NFA $\rightarrow$ DFA <sub>$\epsilon$</sub>

- Have:  $N = (Q, \Sigma, \delta, q_0, F)$
- Want to: construct a DFA  $M = (Q', \Sigma, \delta', q_0', F')$

1.  $Q' = \mathcal{P}(Q)$ .

Almost the same, except ...

2. For  $R \in Q'$  and  $a \in \Sigma$ ,

$$\delta'(R, a) = \bigcup_{r \in R} \cancel{\delta(r, a)} \quad \epsilon\text{-REACHABLE}(\delta(r, a))$$

3.  $q_0' = \cancel{\{q_0\}} \quad \epsilon\text{-REACHABLE}(\{q_0\})$

4.  $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

# NFA $\rightarrow$ DFA $_{\varepsilon}$ Proof of Correctness

- Let  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
- And let NFA $\rightarrow$ DFA $_{\varepsilon}(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$
- Correctness criteria:  $L(N) = L(D)$
- We will prove a stronger statement:  $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ 
  - That is, for all strings  $w$ , the DFA and NFA end in the same set of states

(Same as before)



# NFA $\rightarrow$ DFA $_{\epsilon}$ Proof of Correctness

- Let  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
- And let  $\text{NFA}\rightarrow\text{DFA}(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

Theorem:  $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$

Almost the same, except ...

Proof: (by induction on length of  $w$ )

- Base case  $w = \epsilon$   $\hat{\delta}_D(\{q_0\}, \epsilon)$  and  $\hat{\delta}_N(q_0, \epsilon)$  are  $\epsilon$ -REACHABLE( $q_0$ )
- Inductive case  $w = xa$ 
  - IH:  $\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$ , call this set of states  $R$
  - NFA last step (from  $\delta_N$  definition)  $\bigcup_{r \in R} \epsilon\text{-REACHABLE}(\delta_N(r, a))$
  - DFA last step (from NFA $\rightarrow$ DFA definition)  $\bigcup_{r \in R} \epsilon\text{-REACHABLE}(\delta_N^{67}(r, a))$

# Proving that NFAs Recognize Reg Langs

## Theorem:

A language  $A$  is regular **if and only if** some NFA  $N$  recognizes it.

## Proof:

$\Rightarrow$  If  $A$  is regular, then some NFA  $N$  recognizes it

- We know: if  $A$  is regular, then a DFA recognizes it
- So convert DFA to an NFA

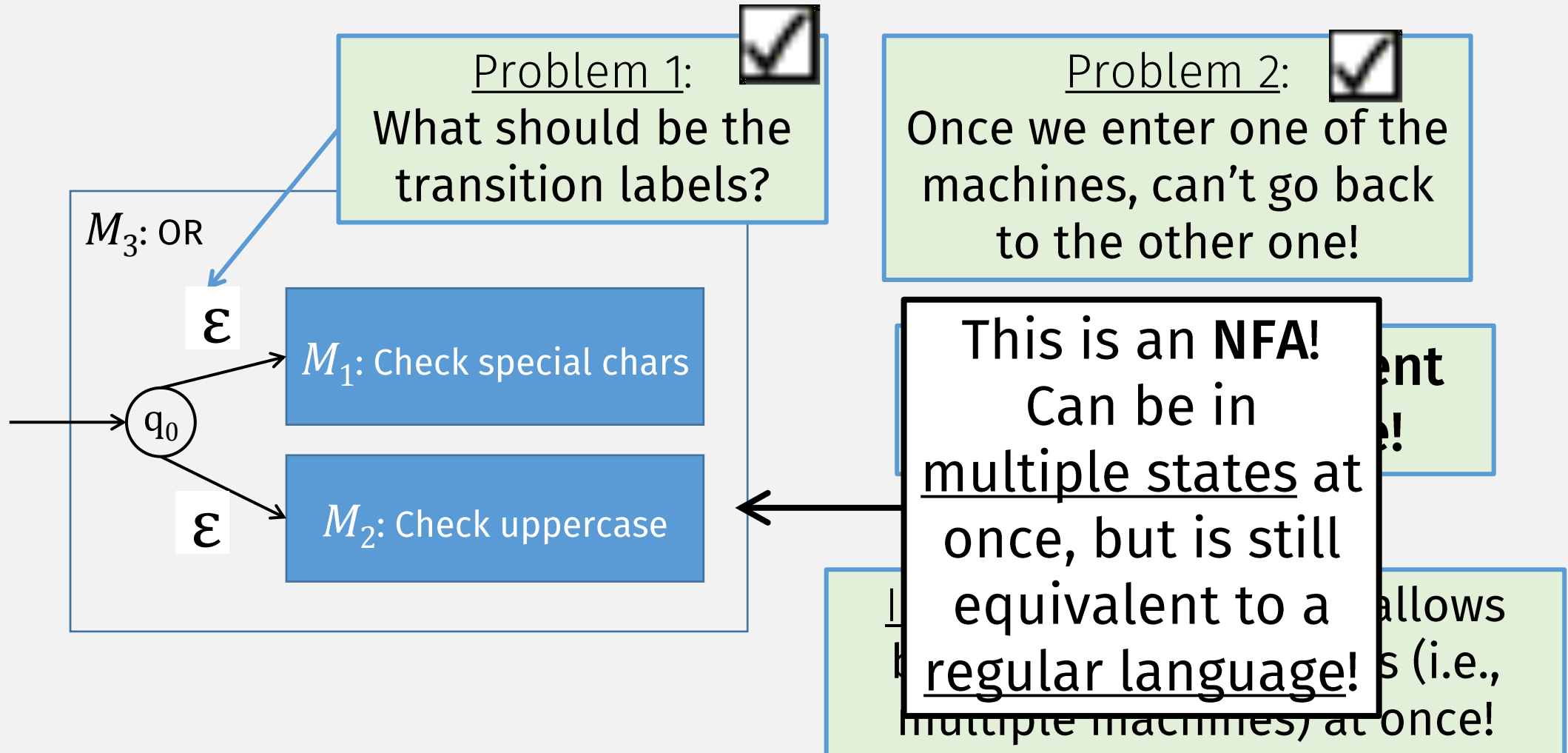
$\Leftarrow$  If an NFA  $N$  recognizes  $A$ , then  $A$  is regular

- We know: if a DFA recognizes a language, then it is regular

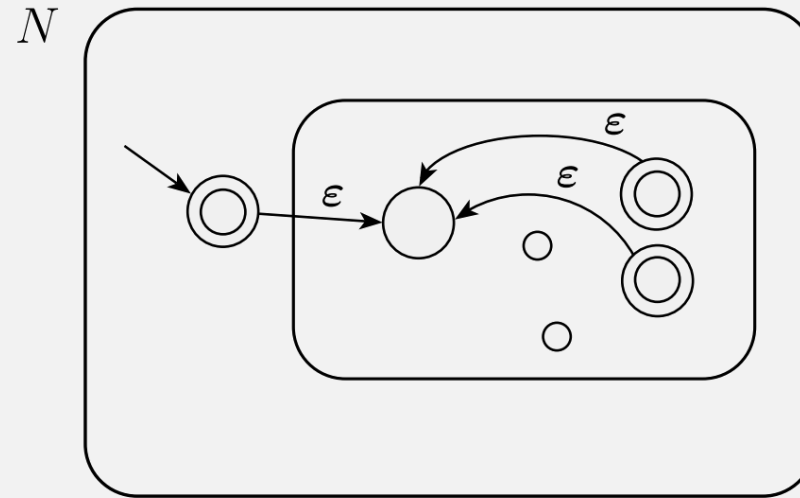
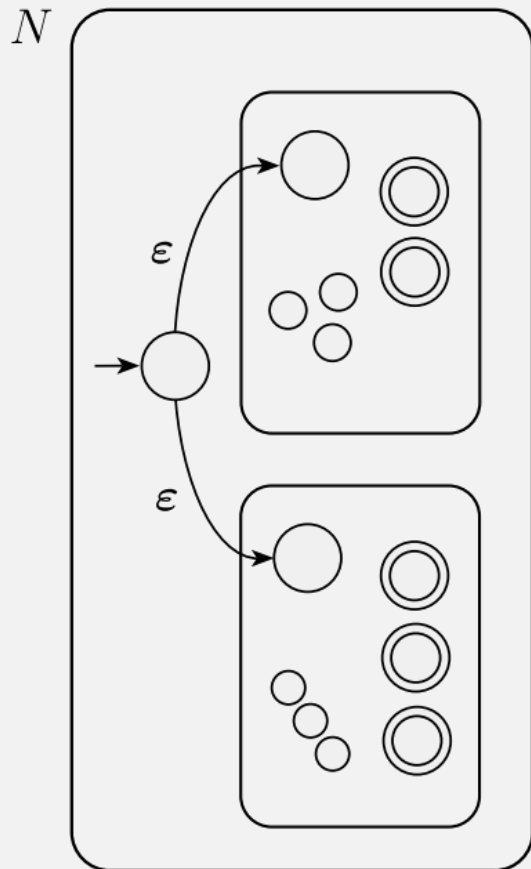


- So convert NFA to DFA ...
- ... Using NFA $\rightarrow$ DFA algorithm we just defined! ■ (Q.E.D.)

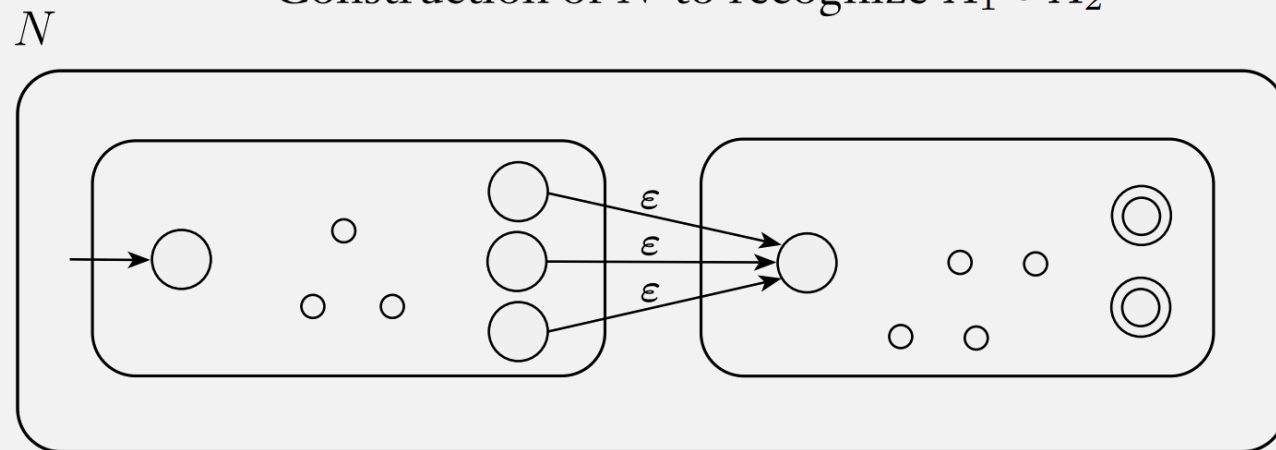
# Combining DFAs



# Next Time: More “Combining” Operations



Construction of  $N$  to recognize  $A_1 \circ A_2$



# **In-class Quiz 9/15**

On gradescope