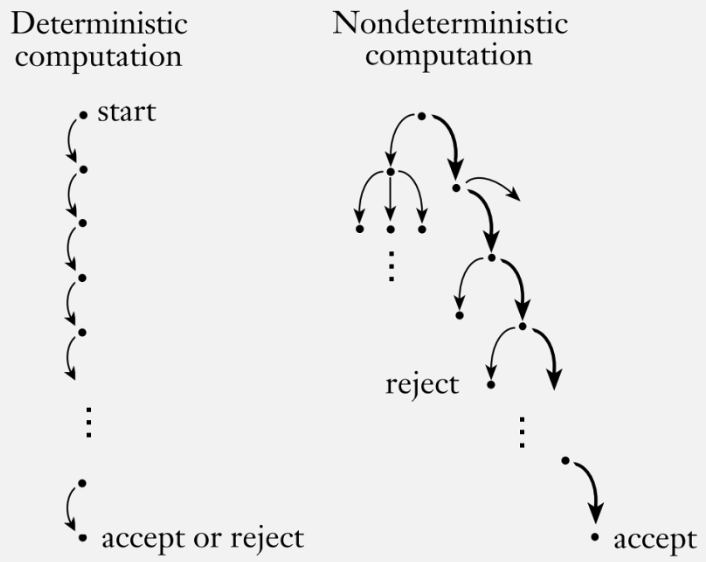
CS420

Nondeterminism and NFAs

Wednesday, February 2, 2022

UMass Boston Computer Science



Announcements

- HW 1
  - due Sunday 2/6 11:59pm EST

- Grades so far published in Gradescope
  - Use “regrade” feature for questions or regrade requests

- See Piazza post about asking HW questions
  - Remember this is a math course
  - So key terms have precise formal definitions
  - Step 1 to solving a problem is to understand all definitions

Last Time: Formal Definition of NFAs

DEFINITION

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

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

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. Σ 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***.

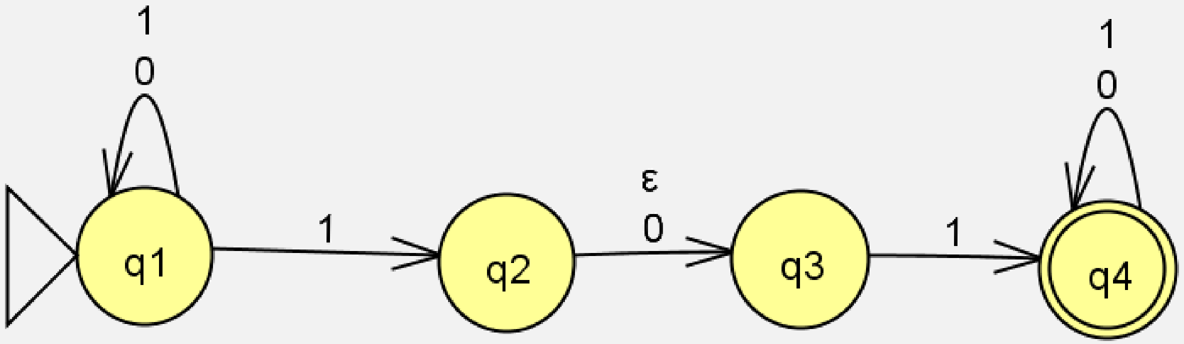
Different transition fn

NFA transition not required to read input, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Transition results in a set of states

Last Time: NFA Example

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



DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ 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

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

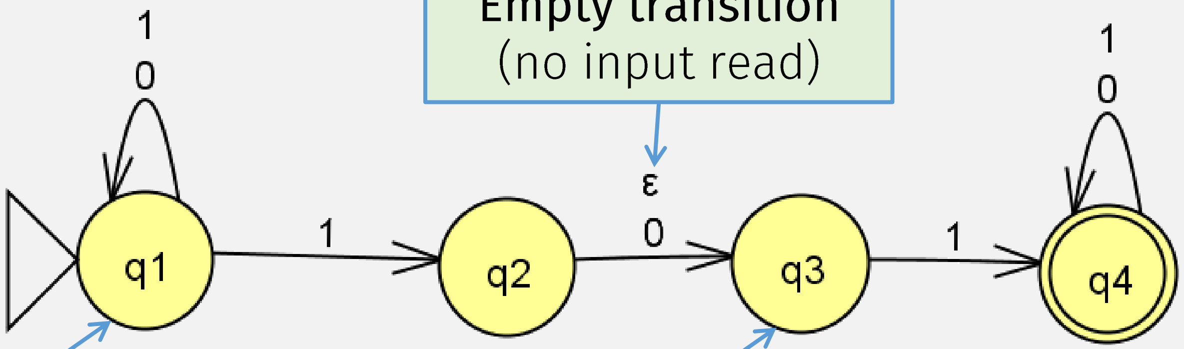
1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

Empty transition (no input read)

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

Result of transition is a set

4. q_1 is the start state, and
5. $F = \{q_4\}$.

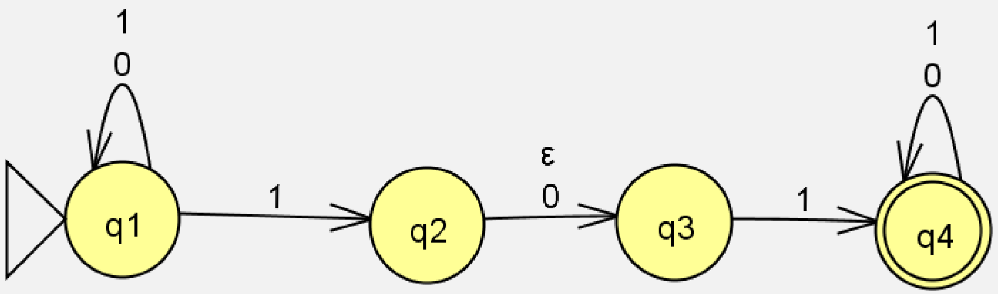


Empty transition (no input read)

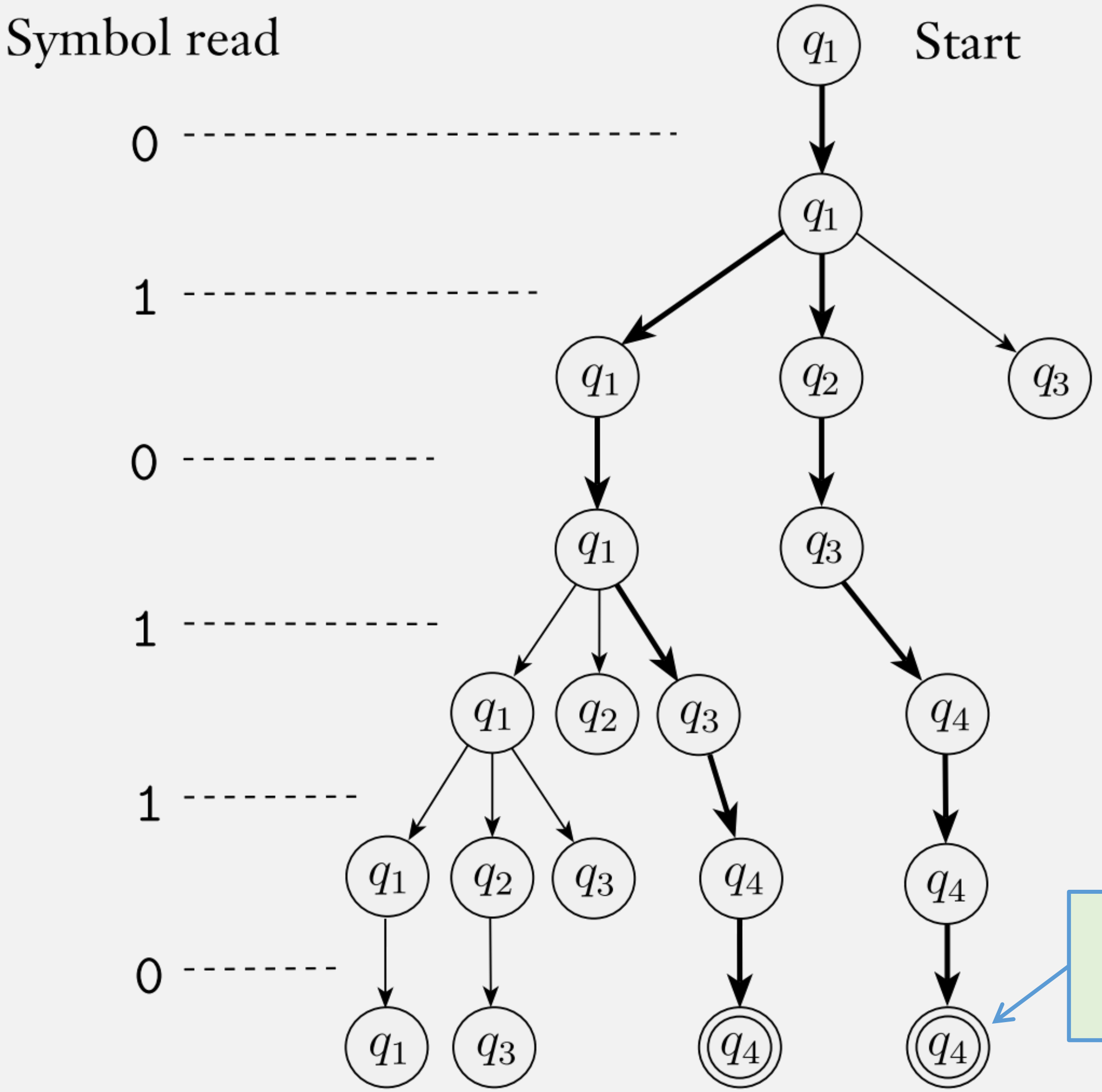
Multiple 1 transitions

No 0 transition

Running Programs, NFAs (JFLAP demo): `010110`



NFA Computation Sequence



An NFA **accepts** its input if at least one path ends in an accept state

Each step can branch into multiple states at the same time!

So this is an **accepting computation**

Flashback: DFA Computation Model

Informally

- Machine = a DFA
- Input = string of chars

Machine accepts input if:

- Start in "start state"
- Repeat:
  - Read 1 char, change state according to transition fn
- Result =
  - "Accept" if last state is "Accept" state
  - "Reject" otherwise

Formally

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

M ***accepts*** w if

sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with

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

NFA

~~*Flashback:* DFA~~ Computation Model

Informally

- Machine = a ~~DFA~~ NFA
- Input = string of chars

Machine accepts input if:

- Start in "start state"
- Repeat:
  - Read 1 char, change state according to transition fn
- Result =
  - "Accept" if last state is "Accept" state
  - "Reject" otherwise

Formally

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

M ***accepts*** w if

sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with

- $r_0 = q_0$
- ~~$\delta(r_i, w_{i+1}) = r_{i+1}$,~~ for $i = 0, \ldots, n-1$

  $r_{i+1} \in \delta(r_i, w_{i+1})$ ← This is now a set
- $r_n \in F$

DEFINITION

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

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

Flashback: DFA Extended Transition Fn

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

- ***Inputs***:
  - Beginning state $q \in Q$
  - Input string $w = w_1 w_2 \cdots w_n$
- ***Output***:
  - Ending state

(Defined recursively)

Empty string

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

First char

Remaining chars

- Recursive case: $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n)$

Recursive call

Single transition step

NFA

~~*Flashback:* DFA~~ Extended Transition Fn

Define the extended transition function: ~~$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$~~

$\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$

Result is set of states

- ***Inputs***:
  - Beginning state $q \in Q$
  - Input string $w = w_1 w_2 \cdots w_n$
- ***Output***:
  - ~~Ending state~~ Set of ending states

(Defined recursively)

- **Base case**: ~~$\hat{\delta}(q, \varepsilon) = q$~~ $\hat{\delta}(q, \epsilon) = \{q\}$
- **Recursive case**: ~~$\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w$~~

First char

$$\delta(q, w_1) = \{q_1, \ldots, q_k\}$$

$$\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \hat{\delta}(q_i, w_2 \cdots w_n)$$

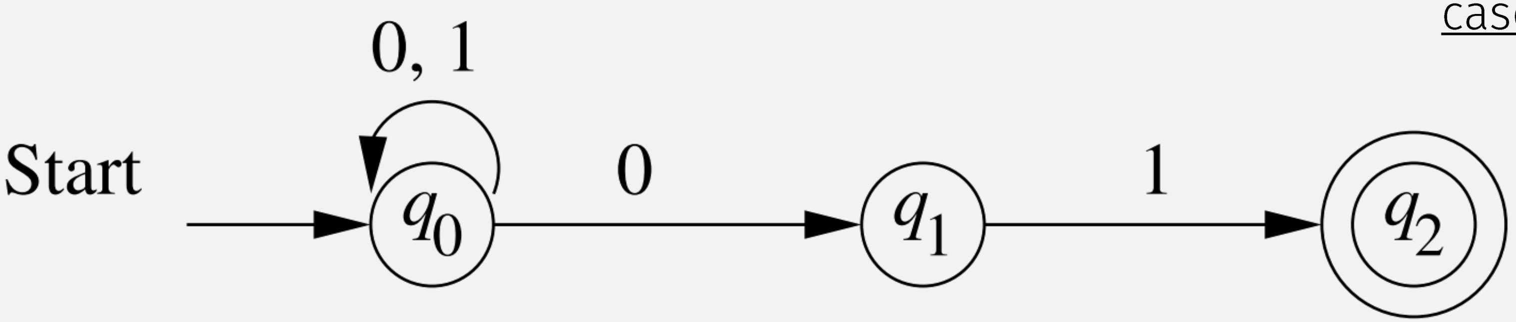
Combine results of recursive calls for each q_i

NFA Extended δ Example

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

Recursive case: $\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \hat{\delta}(q_i, w_2 \cdots w_n)$

where: $\delta(q, w_1) = \{q_1, \ldots, q_k\}$



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

Stay in start state

We haven't considered empty transitions!

- $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$

Same as single step δ

Combine results of recursive calls with "rest of input"

- $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

Adding Empty Transitions

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

(Defined recursively)

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

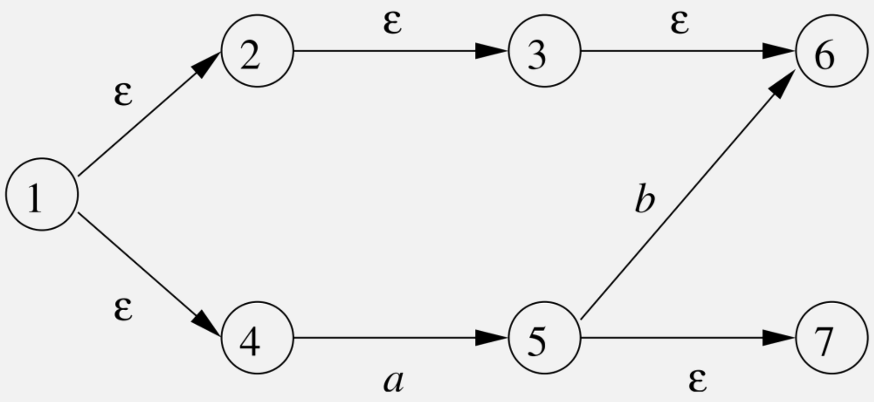
- Inductive case:

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

A state is in the reachable set if …

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

ε-REACHABLE Example



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

No empty transitions

NFA Extended Transition Function

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

- *Inputs*:
  - Some beginning state q
  - 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: $\delta(q, w_1) = \{q_1, \ldots, q_k\}$
  - Then: $\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \hat{\delta}(q_i, w_2 \cdots w_n)$

With empty transitions

NFA Extended Transition Function

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

- *Inputs*:
  - Some beginning state q
  - Input string $w = w_1 w_2 \cdots w_n$
- *Output*:
  - Set of ending states

(Defined recursively)

- Base case: $\hat{\delta}(q, \epsilon) = \{q\}$ $\varepsilon\text{-REACHABLE}(q)$
- Recursive case:
  - If: $\delta(q, w_1) = \{q_1, \ldots, q_k\}$ ➡ $\bigcup_{i=1}^{k} \varepsilon\text{-REACHABLE}(q_i) = \{r_1 \ldots, r_m\}$
  - Then: $\hat{\delta}(q, w) = \bigcup_{i=1}^{k\ m} \hat{\delta}(q_i\ r_i, w_2 \cdots w_n)$

Summary: 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

Concatenation: $A \circ B = \{xy | \, x \in A \text{ and } y \in B\}$

Last Time: Concatenation of Languages

Let the alphabet Σ be the standard 26 letters $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$.

If $A = \{\mathtt{good}, \mathtt{bad}\}$ and $B = \{\mathtt{boy}, \mathtt{girl}\}$, then

$$A \circ B = \{\mathtt{goodboy}, \mathtt{goodgirl}, \mathtt{badboy}, \mathtt{badgirl}\}$$

Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$

Last Time: Concatenation is Closed?

THEOREM

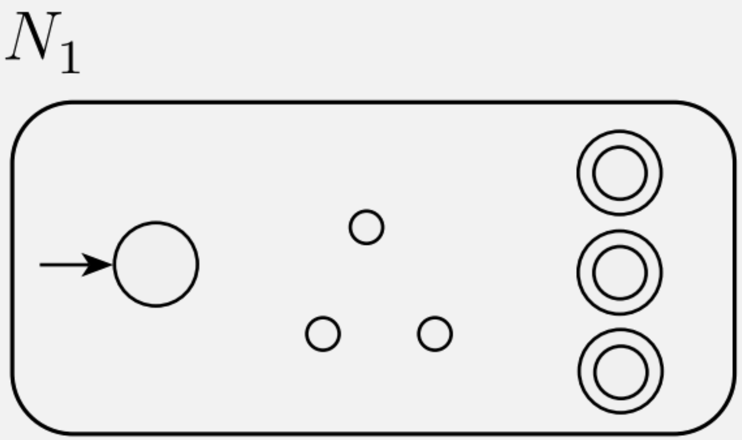
The class of regular languages is closed under the concatenation operation.

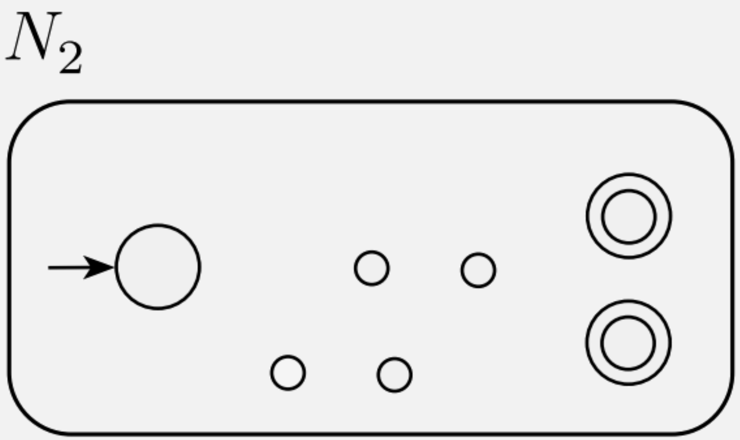
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

Proof: Construct a new machine

- How does it know when to switch machines?
  - Can only read input once

Concatentation



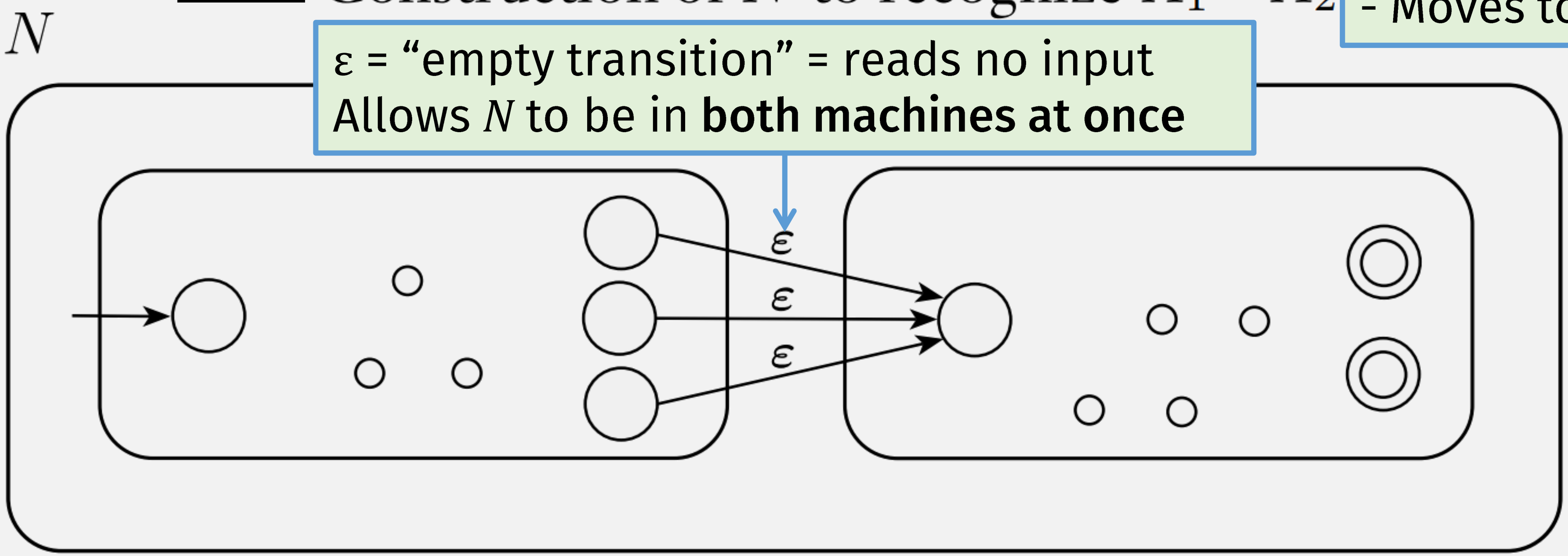


Let N_1 recognize A_1, and N_2 recognize A_2.

N is an NFA! It simultaneously:
- Keeps checking 1st part with N_1
  and
- Moves to N_2 to check 2nd part

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

ε = "empty transition" = reads no input
Allows N to be in **both machines at once**



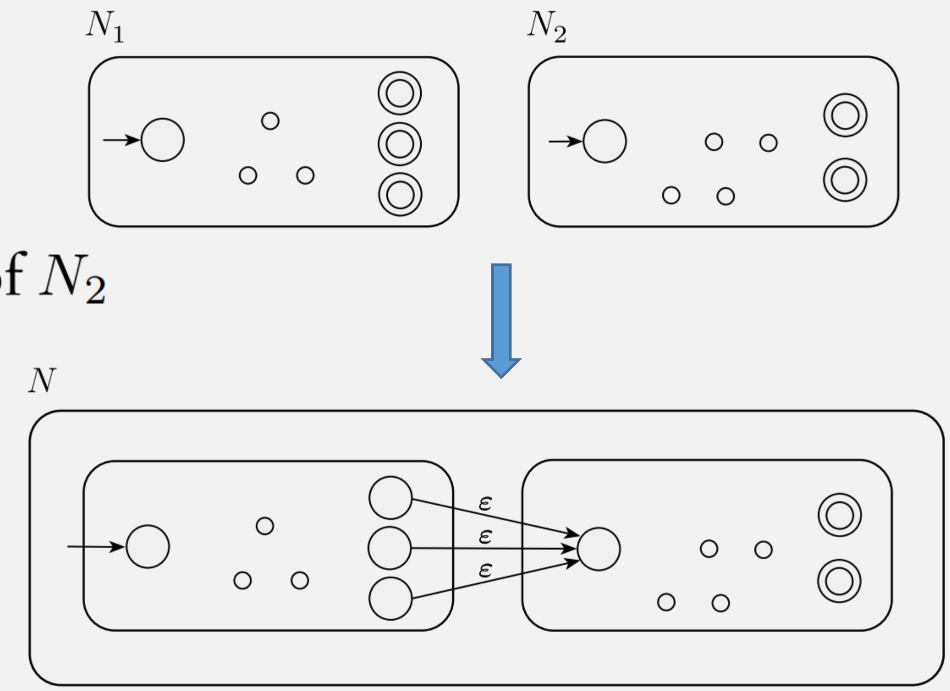
Concatenation is Closed for Regular Langs

PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of N_1
3. The accept states F_2 are the same as the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,



Concatenation is Closed for Regular Langs

Wait, is this true?

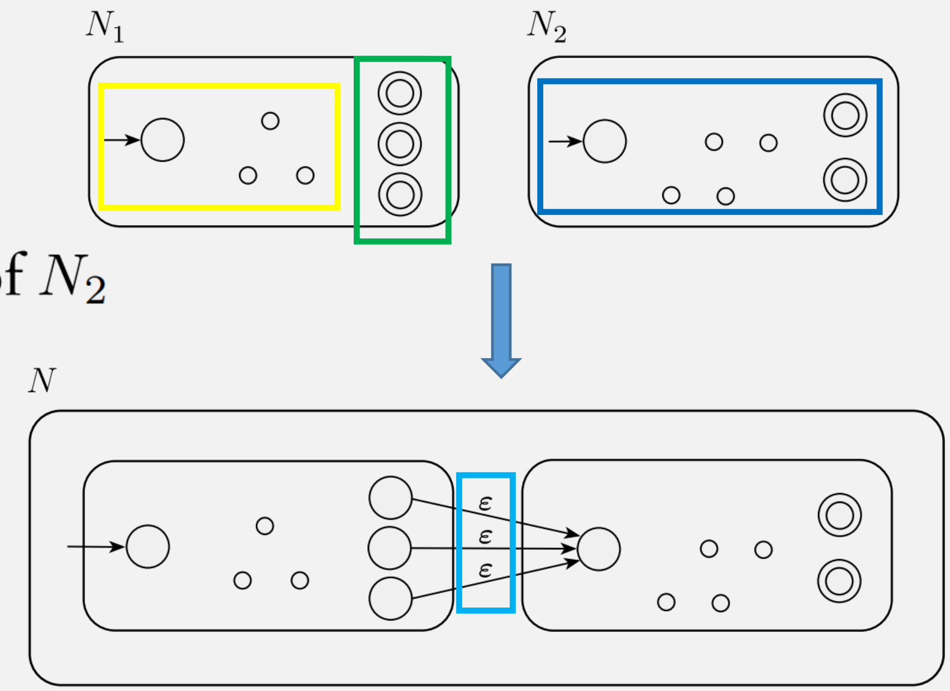
PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of N_1
3. The accept states F_2 are the same as the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(?, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(?, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(?, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(?, a) & q \in Q_2. \end{cases}$$



??? ∎

Flashback: A DFA's Language

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

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

- M ***recognizes language*** A if $A = \{w |\ M \text{ accepts } w\}$

- A language is a **regular language** if a DFA recognizes it

An NFA's Language

- For NFA $N = (Q, \Sigma, \delta, q_0, F)$

intersection

accept states

- N ***accepts*** w if $\hat{\delta}(q_0, w) \cap F \neq \emptyset$ ← not empty
  - i.e., if the final states have at least one accept state

- Language of N = $L(N)$ = $\left\{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$

Q: How does an NFA's language relate to regular languages

- Definition: A language is regular if a **DFA** recognizes it

Is Concatenation Closed for Reg Langs?

- Concatenation of DFAs produces an NFA

- To finish the proof, we must prove that NFAs *also* recognize regular languages.

- Specifically, we must prove:
  - NFAs ⇔ regular languages

Check-in Quiz 2/2

On gradescope