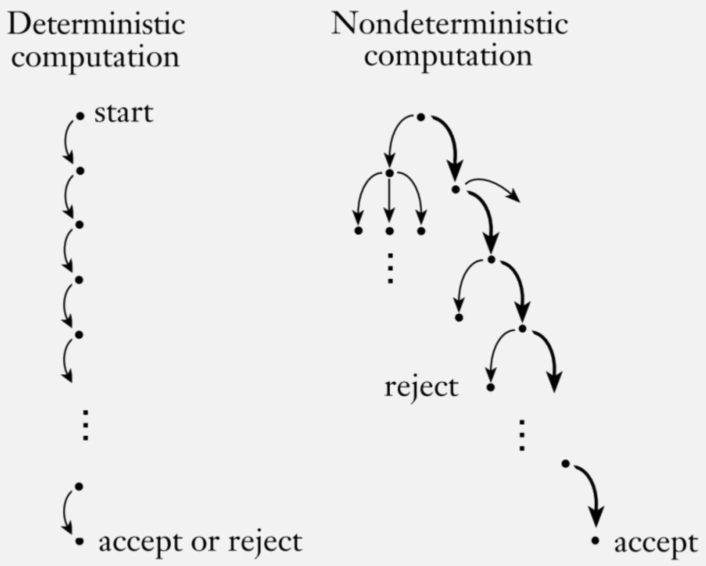
CS 420

Nondeterminism

Wednesday, February 8, 2023

UMass Boston Computer Science



Announcements

- HW 1 in
  - ~~due Tues 2/7 11:59pm EST~~
- HW 2 out
  - due Tues 2/14 11:59pm EST

Last Time: Is Union Closed For Regular Langs?

In this course, **we are interested in closed operations for a** **<u>set of languages</u>** (here the **set of regular languages**)

(*In general, a* **<u>set</u>** is **closed** under an operation if **applying the** <u>operation</u> to **members of the set** produces a **result in the same set**)

Want to prove this statement

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

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

Or this (same) **statement**

Last Time: Is Union Closed For Regular Langs?

(*In general*, a set is **closed** under an operation if applying the operation to members of the set produces a result in the same set)

THEOREM

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

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

Want to prove this statement

Or this (same) statement

A member of the set of regular languages is …

… a regular language, which itself is a set (of strings) …

… so the operations we're interested in are **set operations**

Last Time: Is Union Closed For Regular Langs?

Statements

Do we know anything about A_1 and A_2?

1. A_1 and A_2 are regular languages
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2

???

4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ (todo)
5. M recognizes $A_1 \cup A_2$

How to create this? Don't know what A_1 and A_2 are!

6. $A_1 \cup A_2$ is a regular language
7. The class of regular languages is closed under the union operation.

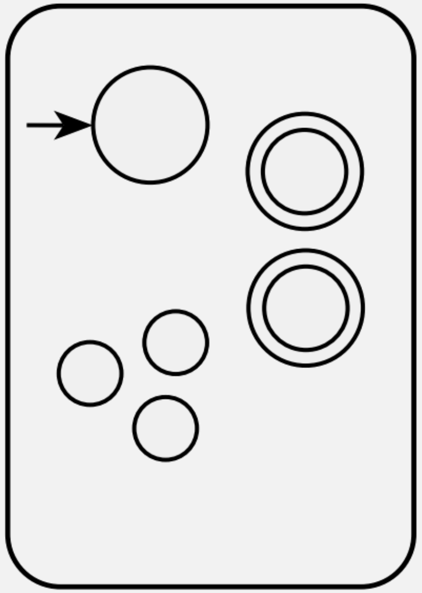
   In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Justifications

1. Assumption
2. Def of Regular Language
3. Def of Regular Language
4. Def of DFA
5. See examples
6. Def of Regular Language
7. From stmt #1 and #6

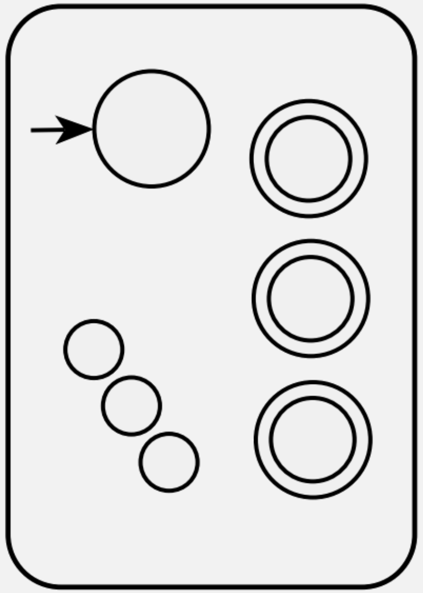
M_1

recognizes A_1



M_2

recognizes A_2



Regular language A_1
Regular language A_2

If we don't know what exactly these languages are, we still know these facts ...

A language is called a ***regular language*** if some finite automaton recognizes it.

DEFINITION

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

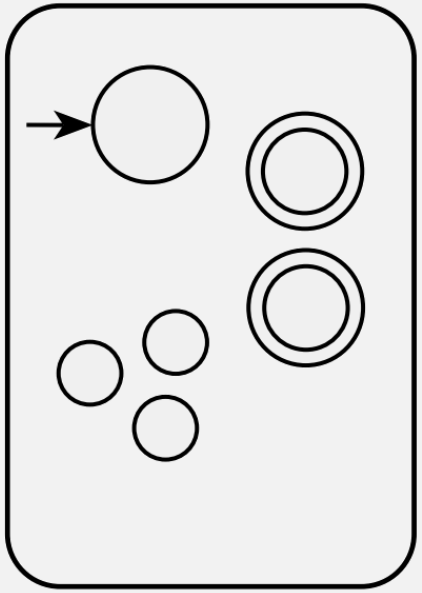
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \text{ recognize } A_1,$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2), \text{ recognize } A_2,$$

Union

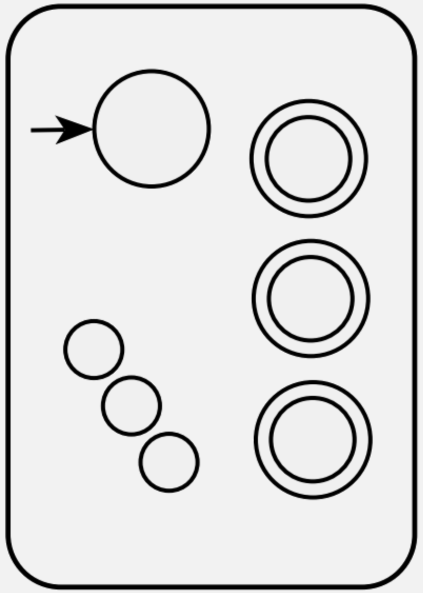
M_1

recognizes A_1



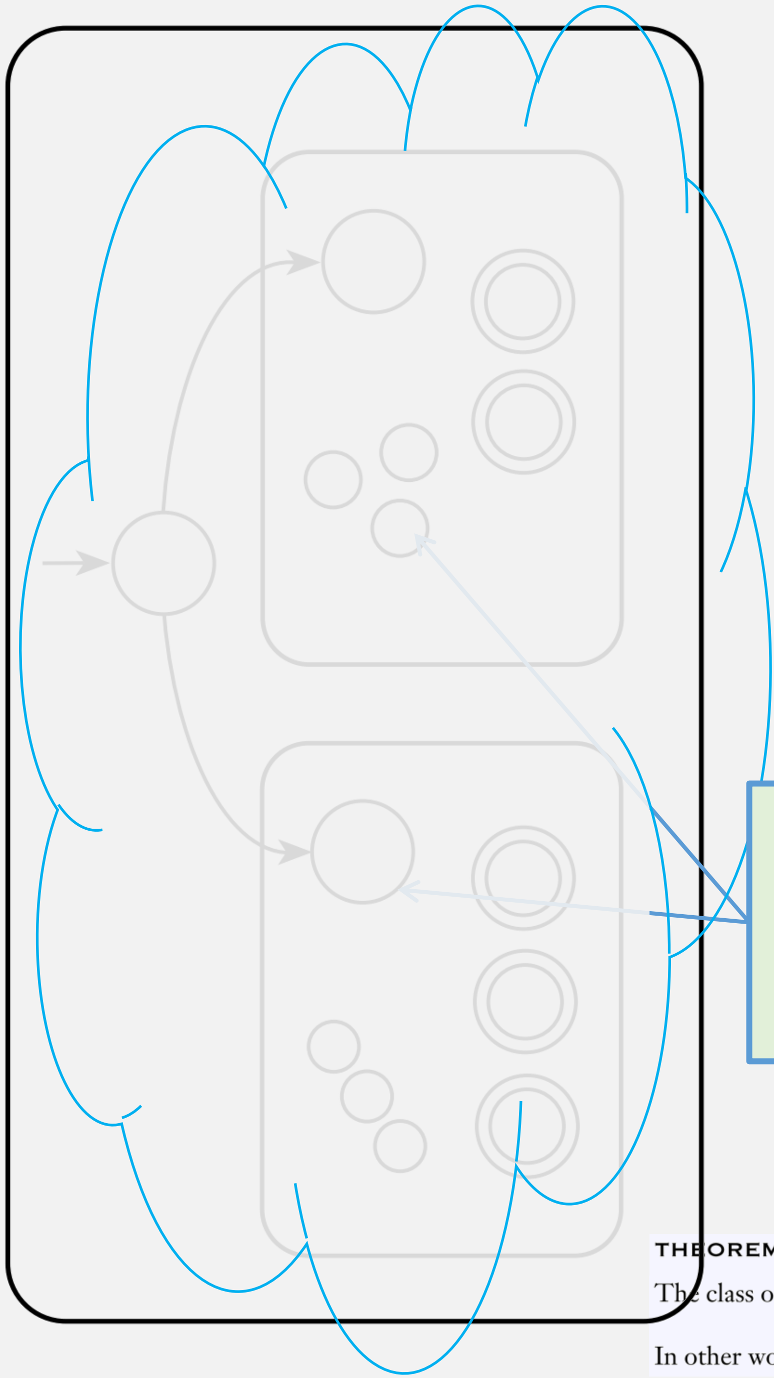
M_2

recognizes A_2



Want: M

Recognizes $A_1 \cup A_2$



Rough sketch Idea: M is a combination of M_1 and M_2 that checks whether its input is accepted by either M_1 and M_2

But, a DFA can only read its input once!

Need to somehow simulate "being in" both an M_1 and M_2 state simultaneously

THEOREM

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

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

Last Time: Union is Closed For Regular Langs

Proof

- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,

Want: M that can simultaneously be in both an M_1 and M_2 state

- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2

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***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

A state of M is a pair:
- the first part is a state of M_1 and
- the second part is a state of M_2

So the states of M is all possible combinations of the states of M_1 and M_2

Last Time: Union is Closed For Regular Langs

<u>*Proof*</u>

- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2

$a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

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. $\boxed{\delta \colon Q \times \Sigma \longrightarrow Q \text{ is the \textit{transition function}},}$[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

A step in M is includes <u>both</u>:
- a step in M_1, and
- a step in M_2

Last Time: Union is Closed For Regular Langs

<u>*Proof*</u>

- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) |\ r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- M transition fn: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$
- M start state: (q_1, q_2)

Start state of M is both start states of M_1 and M_2

Last Time: Union is Closed For Regular Langs

<u>*Proof*</u>

- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) | \; r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- M transition fn: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$
- M start state: (q_1, q_2)

Accept if <u>either</u> M_1 or M_2 accept

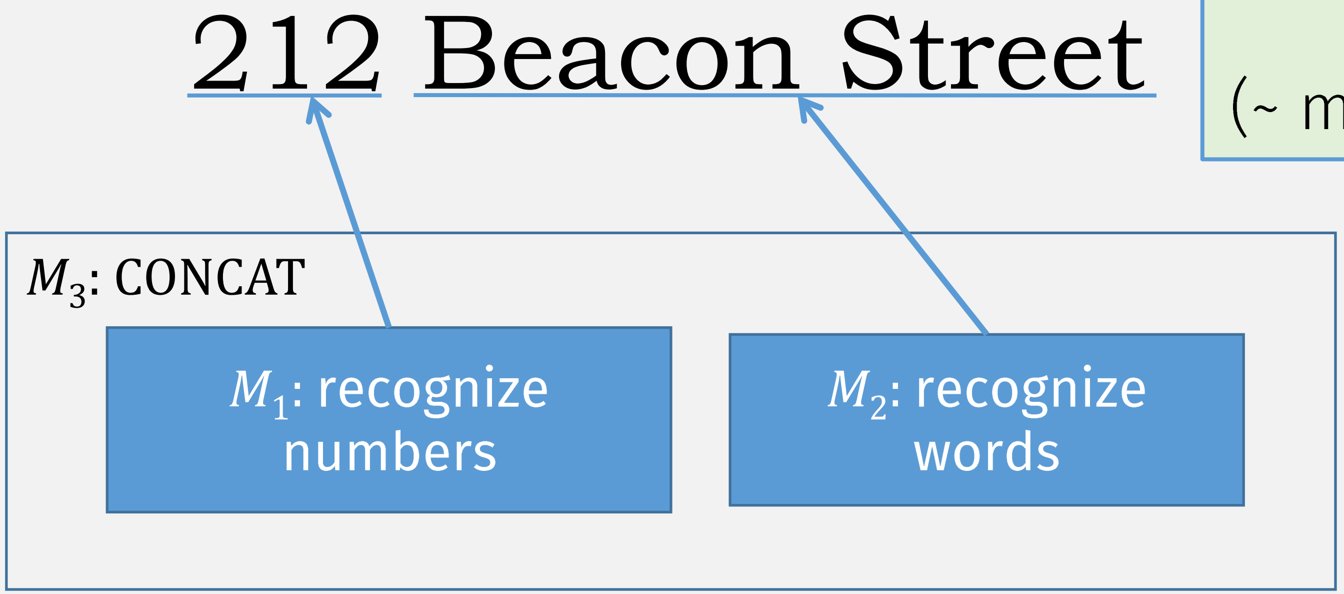
Remember:
Accept states must be subset of Q

- M accept states: $F = \{(r_1, r_2) | \; r_1 \in F_1 \boxed{\text{ or }} r_2 \in F_2\}$

(Q.E.D.) ■

Another operation: Concatenation

Example: **Recognizing street addresses**



We want this operation to be **closed** …
allows using DFAs as building blocks
(~ modular programming)

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

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

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

Is Concatenation Closed?

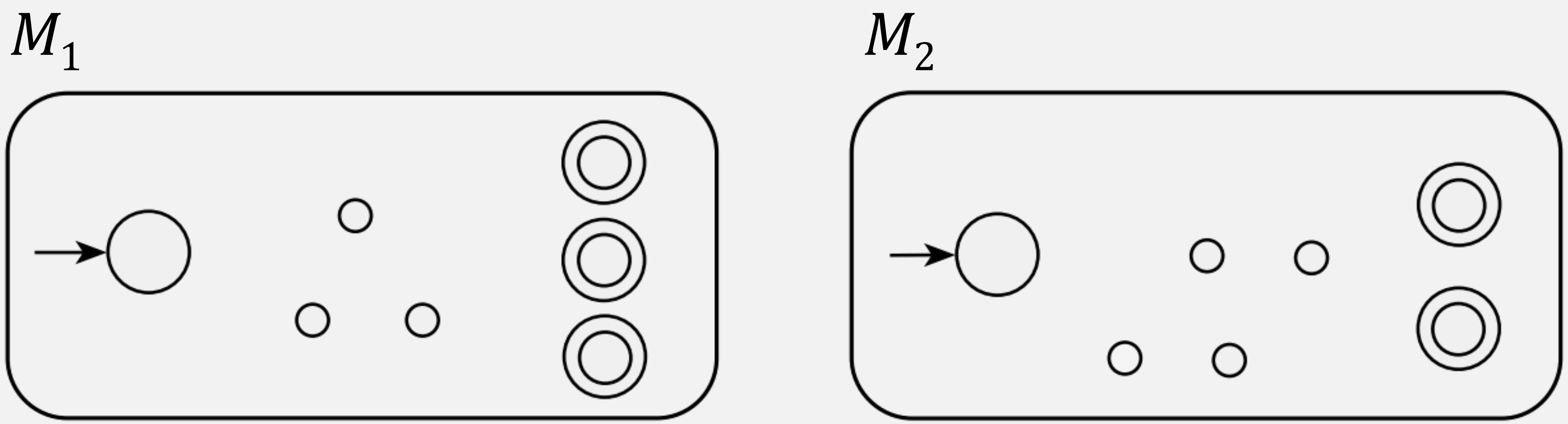
THEOREM

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 $A_1 \circ A_2$.

- **Construct a <u>new</u> machine M recognizing $A_1 \circ A_2$?** (like union)
  - Using **DFA M_1** (which recognizes A_1),
  - and **DFA M_2** (which recognizes A_2)

Concatentation

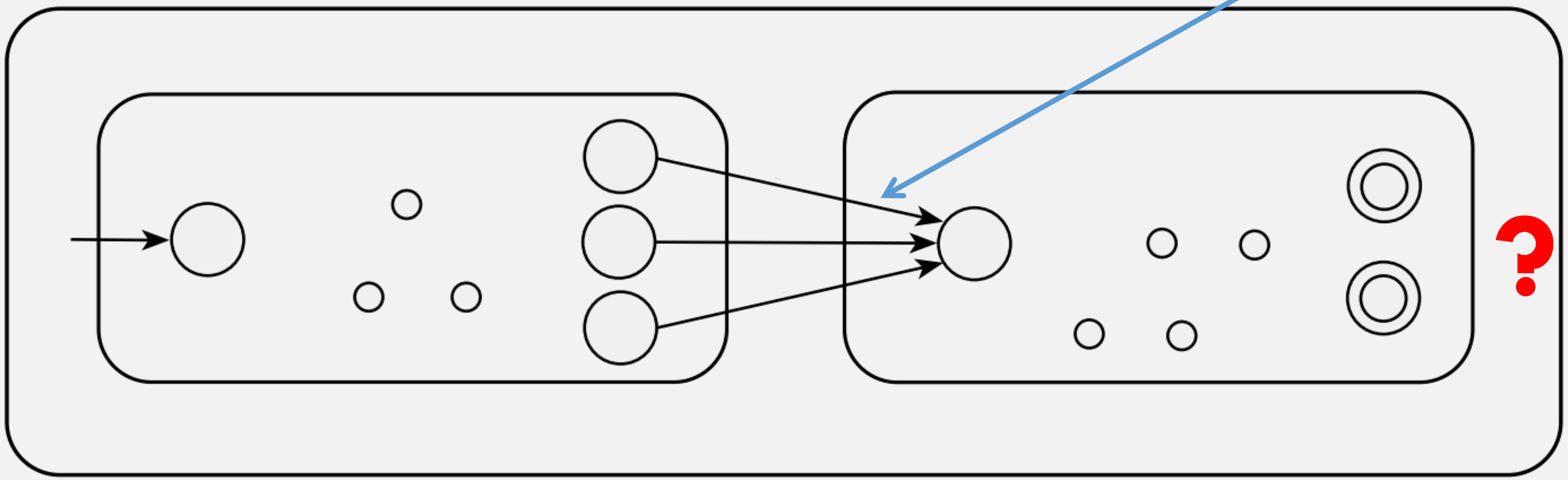


<u>PROBLEM</u>: Can only read input once, can't backtrack

Let M_1 recognize A_1, and M_2 recognize A_2.

<u>Want</u>: Construction of M to recognize $A_1 \circ A_2$

Need to switch machines at some point, but when?



???

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

Overlapping Concatenation Example

- Let M_1 recognize language A = { `jen`, `jens` }
- and M_2 recognize language B = { `smith` }
- Want: Construct M to recognize $A \circ B$ = { `jensmith`, `jenssmith` }

- If M sees `jen` ...
- M must decide to <u>either</u>:

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

Overlapping Concatenation Example

- Let M_1 **recognize language** $A = \{$ **`jen`**, $\boxed{\texttt{jens}}$ $\}$
- and M_2 **recognize language** $B = \{$ $\boxed{\texttt{smith}}$ $\}$
- Want: **Construct** M **to recognize** $A \circ B = \{$ **`jensmith`**, **`jenssmith`** $\}$

- If M **sees** **`jen`** **...**
- M **must decide to** __either__:
  - **stay in** M_1 (correct, if full input is $\boxed{\texttt{jens}}\boxed{\texttt{smith}}$)

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

Overlapping Concatenation Example

- Let M_1 recognize language A = { $\boxed{\texttt{jen}}$, `jens` }
- and M_2 recognize language B = { $\boxed{\texttt{smith}}$ }
- Want: Construct M to recognize $A \circ B$ = { `jensmith`, `jenssmith` }

- If M sees `jen` ...
- M must decide to <u>either</u>:
  - stay in M_1 (correct, if full input is `jenssmith`)
  - or switch to M_2 (correct, if full input is $\boxed{\texttt{jen}}\boxed{\texttt{smith}}$)
- <u>But</u> to recognize $A \circ B$, it needs to handle <u>both cases</u>!!
  - Without backtracking

A **DFA** can't do this!

Is Concatenation Closed?

FALSE?

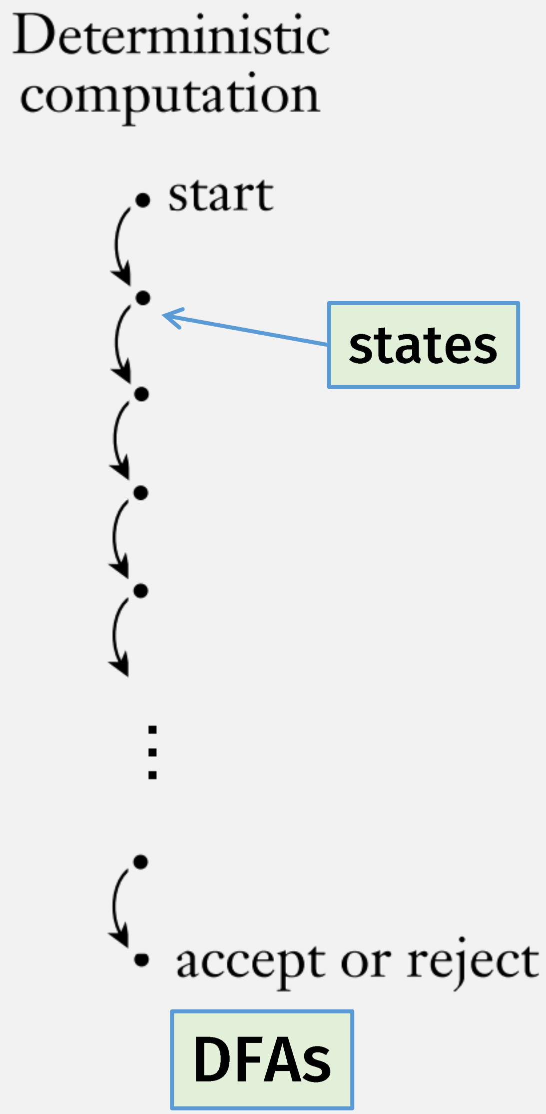
THEOREM

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

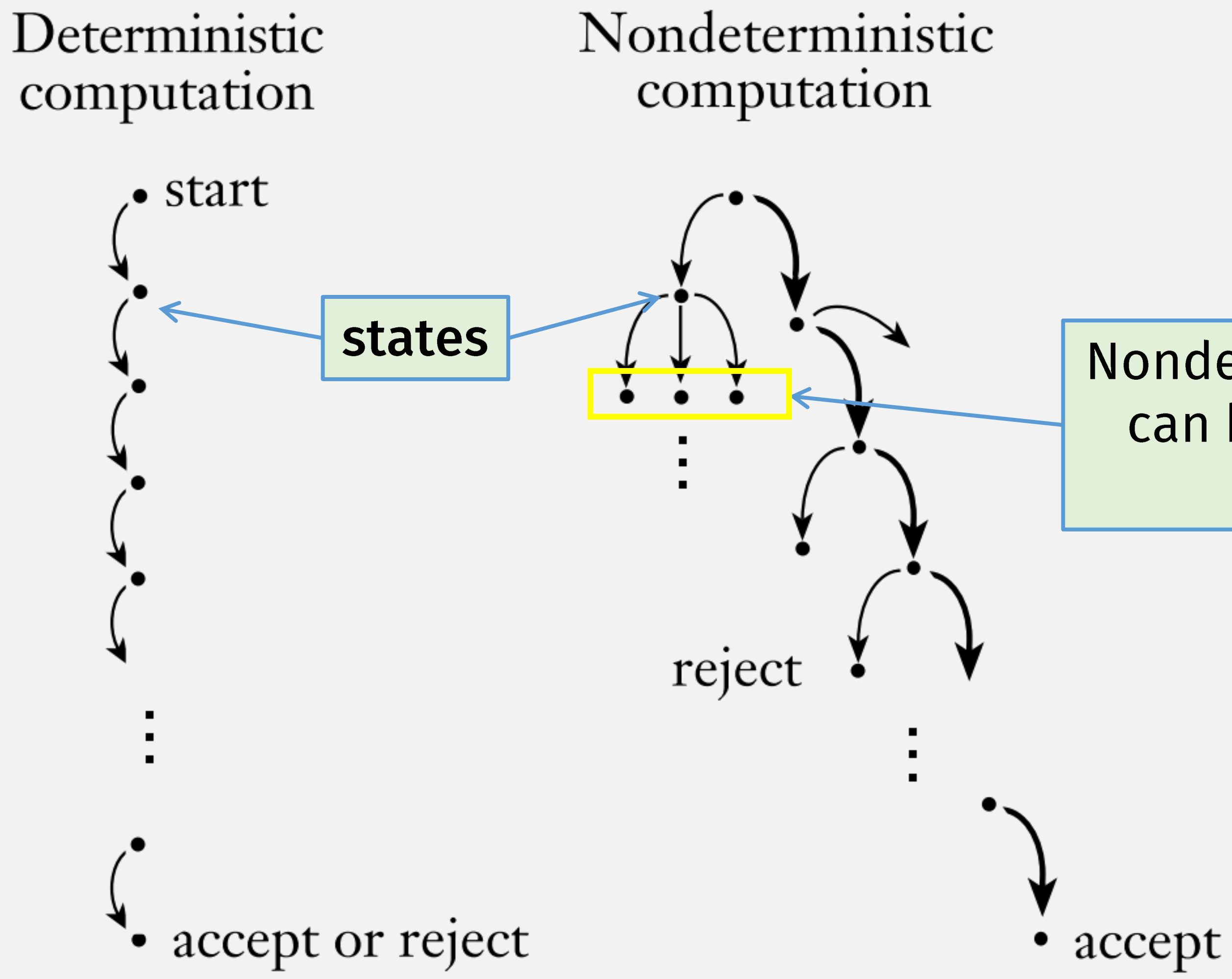
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

- Cannot **combine A_1 and A_2's machine because:**
  - **Not clear when to switch machines?** (can only read input once)
- **Requires a <u>new kind of machine</u>!**
- <u>But</u> **does this mean concatenation <u>is not closed</u> for regular langs?**

Deterministic vs Nondeterministic



Deterministic vs Nondeterministic



Nondeterministic computation can be in multiple states at the same time

DFAs

New FA

Finite Automata: The Formal Definition

DEFINITION

deterministic

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Also called a **Deterministic Finite Automata (DFA)**

Precise Terminology is Important

- A **finite automata** or **finite state machine** (**FSM**) defines …
  … **computation** with a <u>finite</u> **number of states**
- There are <u>many kinds</u> of FSMs
- We've learned <u>one kind</u>, the **Deterministic Finite Automata (DFA)**
  - (So up to now, the terms DFA and FSM refer to the same definition)
- But now we learn <u>other kinds</u>, e.g., **Nondeterministic Finite Automata (NFA)**
- <u>Be careful with terminology!</u>

Nondeterministic Finite Automata (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. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ 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***.

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

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

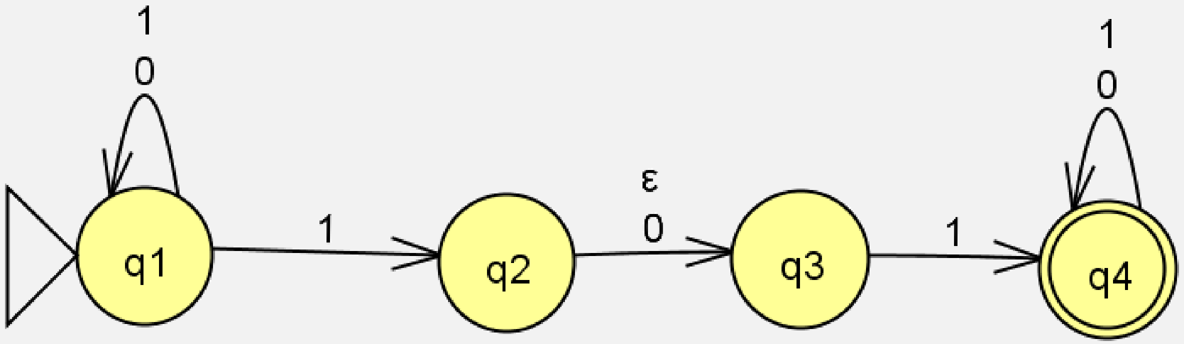
1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

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

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

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

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

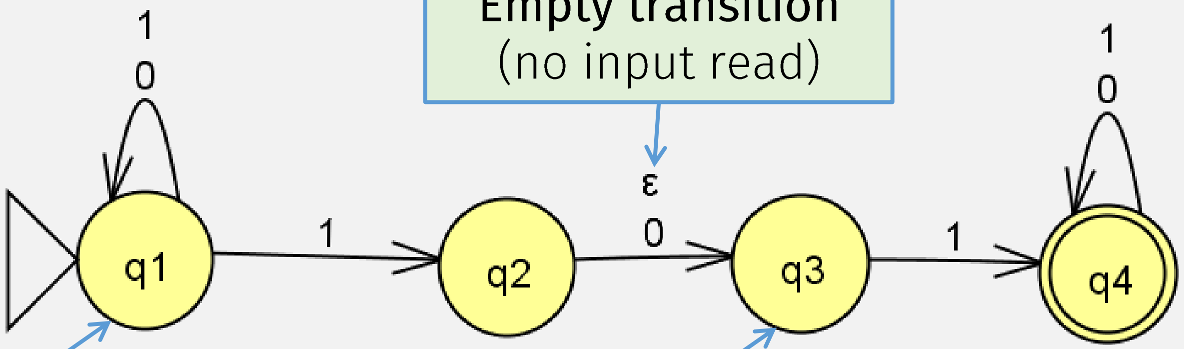
1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

Empty transition (no input read)

Result of transition is a set

	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	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

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



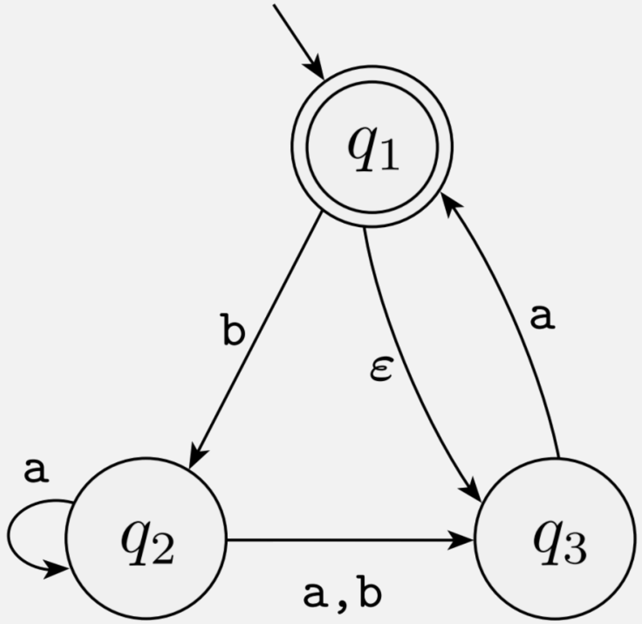
Empty transition (no input read)

Multiple 1 transitions

No 0 transition

In-class Exercise

- Come up with a formal description for the following NFA
  - $\Sigma = \{ \texttt{a}, \texttt{b} \}$



DEFINITION

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

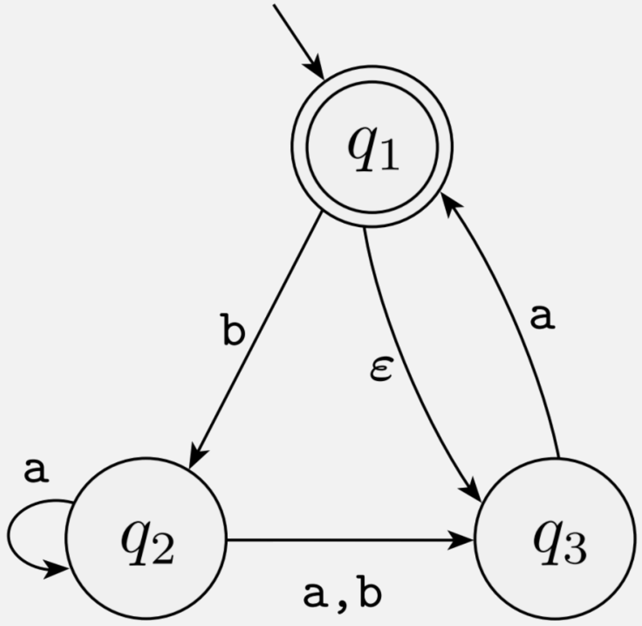
1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

In-class Exercise Solution

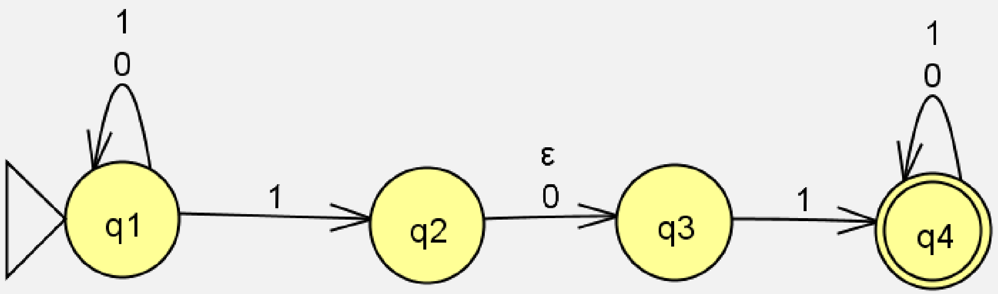
Let $N = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{ q_1, q_2, q_3 \}$
- $\Sigma = \{ \mathtt{a}, \mathtt{b} \}$
- δ ...
- $q_0 = q_1$
- $F = \{ q_1 \}$

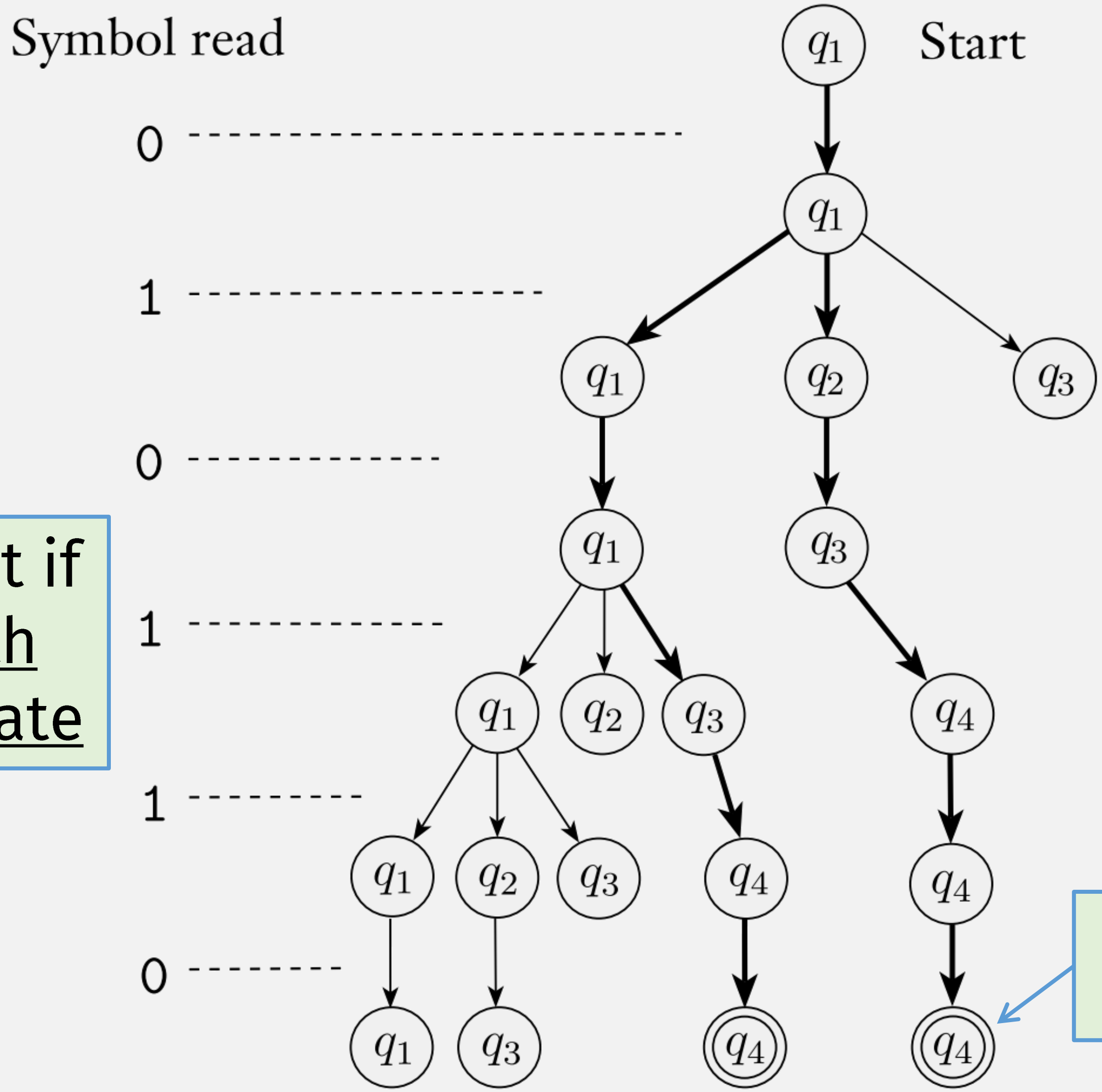
$\delta(q_1, \mathtt{a}) = \{ \}$
$\delta(q_1, \mathtt{b}) = \{ q_2 \}$
$\delta(q_1, \varepsilon) = \{ q_3 \}$
$\delta(q_2, \mathtt{a}) = \{ q_2, q_3 \}$
$\delta(q_2, \mathtt{b}) = \{ q_3 \}$
$\delta(q_2, \varepsilon) = \{ \}$
$\delta(q_3, \mathtt{a}) = \{ q_1 \}$
$\delta(q_3, \mathtt{b}) = \{ \}$
$\delta(q_3, \varepsilon) = \{ \}$



NFA Computation (JFLAP demo): `010110`



NFA Computation Sequence



NFA **accepts** input if at least one path ends in 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, e.g. "`1101`"

Machine "accepts" input if:

- Start in "start state"
- Repeat:
  - Read 1 char;
  - Change state according to the transition table
- Result =
  - Last state is "Accept" state

Formally (i.e., mathematically)

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

M ***accepts*** w if

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

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

NFA

~~*Flashback:* DFA~~ Computation Model

Informally	*Formally (i.e., mathematically)*

- Machine = ~~a DFA~~ an NFA
- Input = string of chars, e.g. “`1101`”

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

Machine “accepts” input if:

- Start in “start state”
- Repeat:
  - Read 1 char;
  - Change state according to the transition table
- Result =
  - Last state is “Accept” state

M ***accepts*** w if

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

- $r_0 = q_0$
- $\cancel{r_i = \delta(r_{i-1}, w_i),}$ for $i = 1, \ldots, n$

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

- $r_n \in F$

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.

$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

Flashback: DFA Extended Transition Function

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

Domain:

- Beginning state $q \in Q$ (not necessarily the start state)
- Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

Range:

- Ending state (not necessarily an accept state)

(Defined recursively, on length of input string)

Empty string

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

nonEmpty string

First char

Remaining chars ("smaller argument")

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

Recursive call

Single transition step

$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

Alternate Extended Transition Function

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

Domain:

- **Beginning state** $q \in Q$ (not necessarily the start state)
- **Input string** $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

Range:

- **Ending state** (not necessarily an accept state)

(Defined recursively, on length of input string)

Empty string

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

nonEmpty string

Recursive call: (smaller argument) computation "so far"

$\delta(\hat{\delta}(q, w_1 \cdots w_{n-1}), w_n)$

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

Single transition step, on last char

NFA

Extended Transition Function

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

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

Domain:

- Beginning state $q \in Q$
- Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

Range:

- Ending ~~state~~ set of states

Result is set of states

NFA Extended Transition Function

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

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

Domain:

- Beginning state $q \in Q$
- Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

Range:

- Ending ~~state~~ set of states

Result is set of states

(Defined recursively, on length of input string)

Empty string

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

NFA

Extended Transition Function

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

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

Result is set of states

<u>Domain</u>:

- Beginning state $q \in Q$
- Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

<u>Range</u>:

- Ending ~~state~~ set of states

(Defined recursively, on length of input string)

Empty string

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

nonEmpty string

All single transition steps for last char

- <u>Recursive</u> case: $\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$

Recursive call: (smaller argument) computation "so far"

where: $\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$

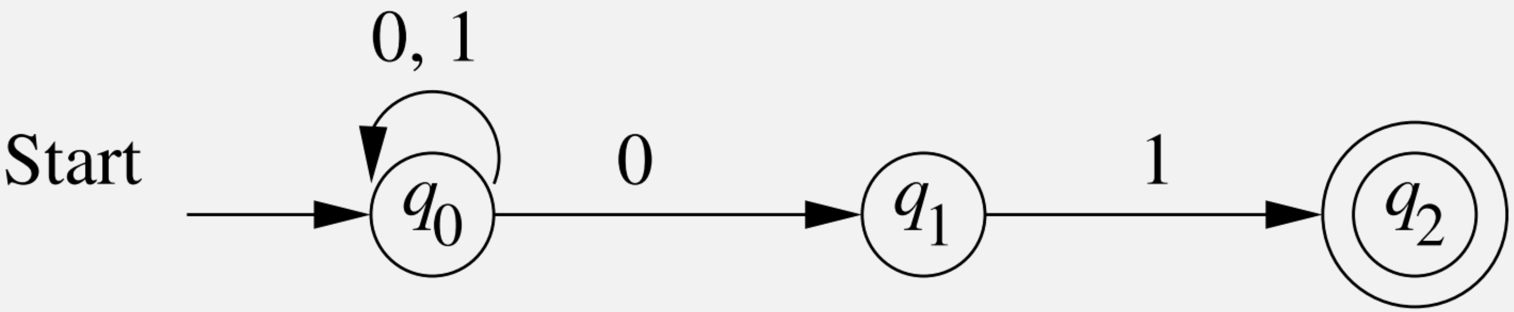
NFA Extended δ Example

<u>Base case:</u> $\hat{\delta}(q, \epsilon) = \{q\}$

<u>Recursive case:</u> $\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$

<u>where:</u>

$\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$



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

Stay in start state

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

Same as single step δ

Combine result of recursive call with "last step"

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

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

We haven't considered **empty transitions!**

Adding Empty Transitions

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

(Defined recursively)

- **<u>Base</u> case:** $q \in \varepsilon\text{-REACHABLE}(q)$

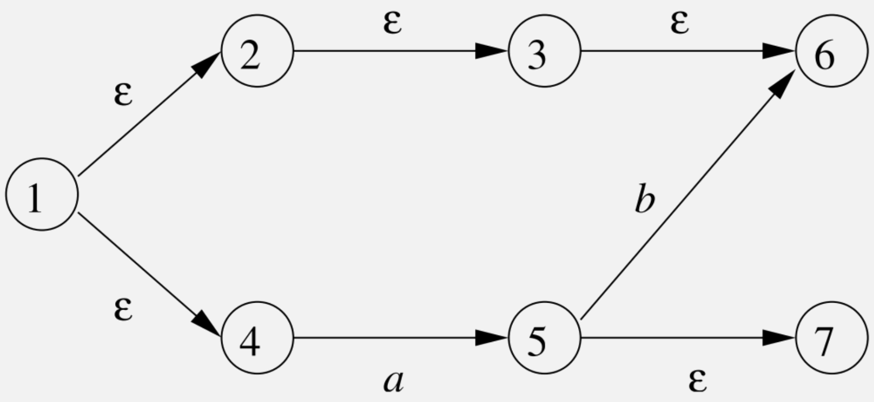
- **<u>Inductive</u> case:**

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

ε-REACHABLE Example



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

No empty transitions

NFA Extended Transition Function

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

<u>Domain</u>:

- Beginning state $q \in Q$
- Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

<u>Range</u>:

- Ending set of states

(Defined recursively, on length of input string)

- <u>Base</u> case: $\hat{\delta}(q, \epsilon) = \{q\}$
- <u>Recursive</u> case: $\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$

  where: $\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \dots, q_k\}$

With empty transitions

NFA Extended Transition Function

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

Domain:

- Beginning state $q \in Q$
- Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

Range:

- Ending set of states

(Defined recursively, on length of input string)

"Take single step, then follow all empty transitions"

- Base case: $\hat{\delta}(q, \epsilon) = \cancel{\{q\}}$ $\varepsilon\text{-REACHABLE}(q)$
- Recursive case: $\hat{\delta}(q, w) = \cancel{\bigcup_{i=1}^{k} \delta(q_i, w_n)}$ $\varepsilon\text{-REACHABLE}(\bigcup_{i=1}^{k} \delta(q_i, w_n))$

where: $\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$

Summary: NFAs vs DFAs

DFAs

- Can only be in <u>one</u> state

- Transition:
  - <u>Must read 1 char</u>

- Acceptance:
  - If final state <u>is</u> accept state

NFAs

- Can be in <u>multiple</u> states

- Transition
  - <u>Can read no chars</u>
  - i.e., empty transition

- Acceptance:
  - If <u>one of</u> 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

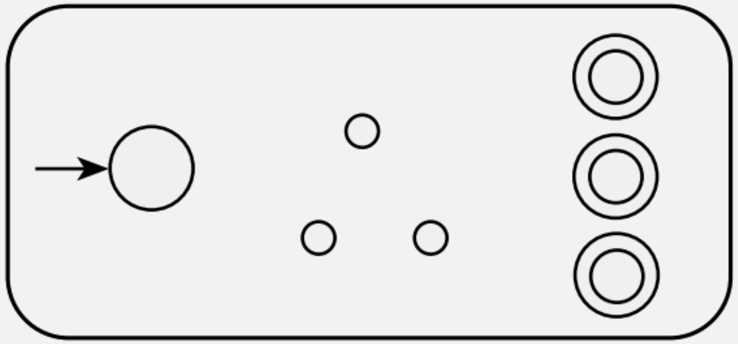
The class of regular languages is closed under the concatenation operation.

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

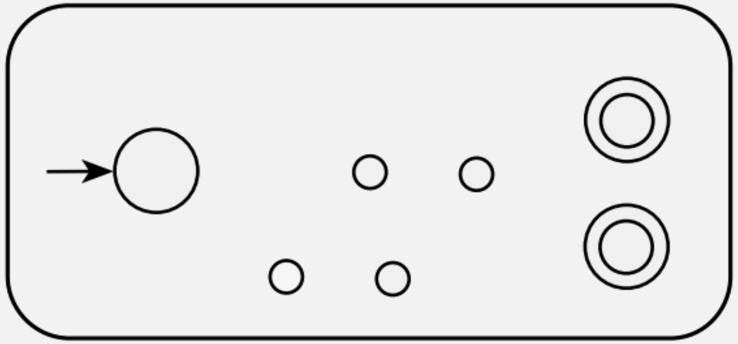
Proof: Construct a new machine

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

N_1



N_2



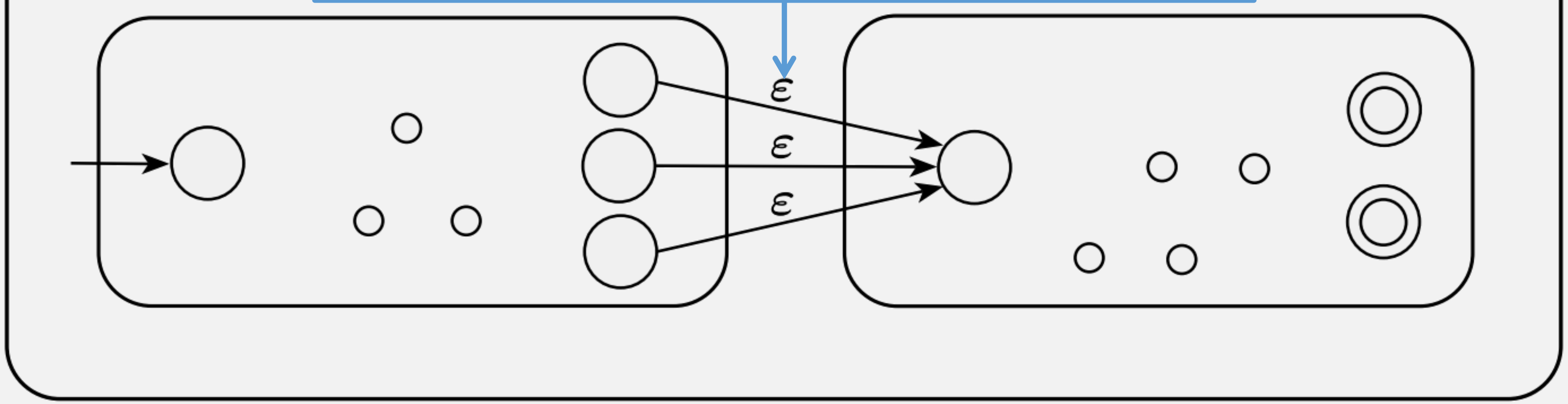
Let N_1 recognize A_1, and N_2 recognize A_2.

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

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

N

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



Flashback: Is Union Closed For Regular Langs?

Statements	**Justifications**
1. A_1 and A_2 are regular languages	1. Assumption
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1	2. Def of Regular Language
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2	3. Def of Regular Language
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. See examples
6. $A_1 \cup A_2$ is a regular language	6. Def of Regular Language
7. The class of regular languages is closed under the union operation. In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.	7. From stmt #1 and #6

Is Concat Closed For Regular Langs?

Statements

1. A_1 and A_2 are regular languages
2. A **NFA** $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1
3. A **NFA** $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2
4. Construct **NFA** $N =$??? ??? (todo)
5. M recognizes $\cancel{A_1 \cup A_2}$ $\boxed{A_1 \circ A_2}$
6. $\boxed{A_1 \circ A_2}$ $\cancel{A_1 \cup A_2}$ is a regular language
7. 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 $A_1 \circ A_2$.

Justifications

1. Assumption
2. Def of Regular Language
3. Def of Regular Language
4. Def of **NFA**
5. See examples
6. Def of Regular Language
7. From stmt #1 and #6

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.

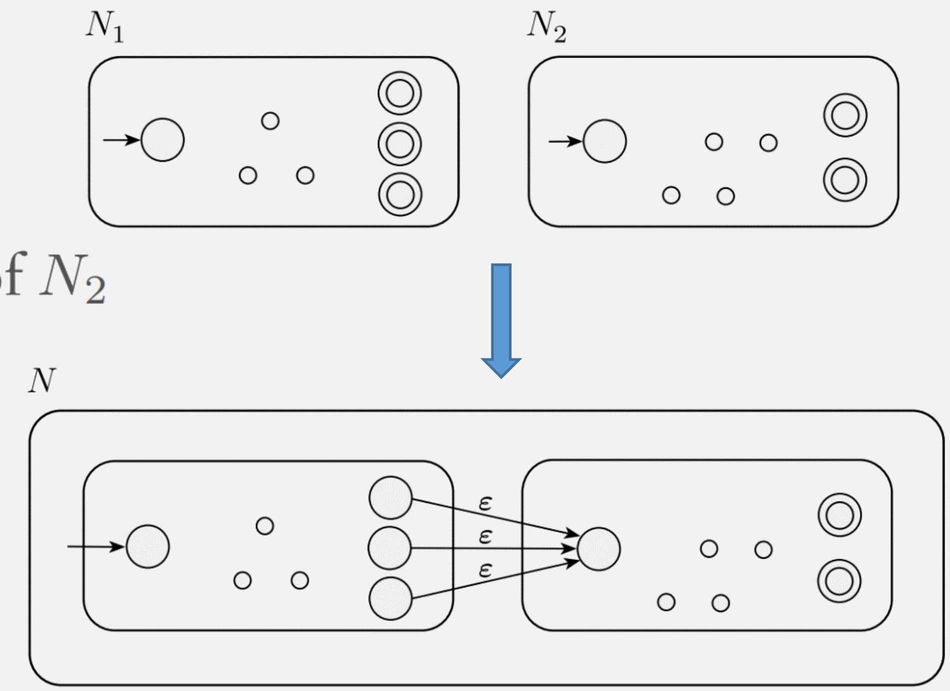
Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

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?

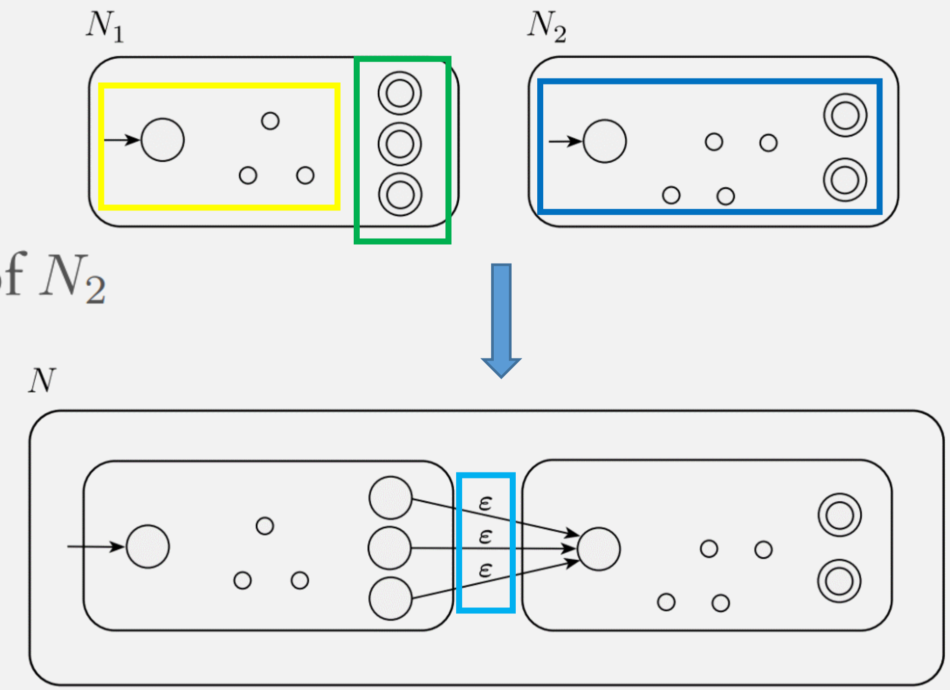
PROOF

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 $a \in \Sigma_\varepsilon$,

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



??? ■

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 $A = \{w \mid M \text{ accepts } w\}$

- 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:**

- $\boxed{\text{NFAs} \Leftrightarrow \text{regular languages}}$

Check-in Quiz 2/8

On gradescope