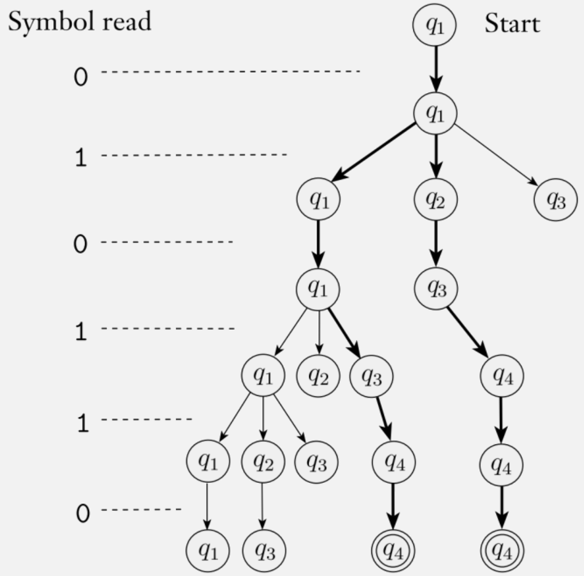
CS420

Computing with NFAs

Monday, February 13, 2023

UMass Boston CS



Announcements

- HW 2 out
  - Due 2/14 11:59pm EST
- TAs
  - Woody Lin
    - OH: Tue 2-3:30pm, McCormack 3rd floor, room 139
  - Richard Chang
    - OH: Friday 2-3:30pm, McCormack 3rd floor, room 139
- Quiz Preview (submit answer in gradescope):
  - In the course so far, what are possible meanings of the ε symbol?

HW 1 Observations

- Problems must be assigned to the correct pages
- Proof format must be a **Statements** and **Justifications** table
- Rejected string examples must use characters from Σ alphabet

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

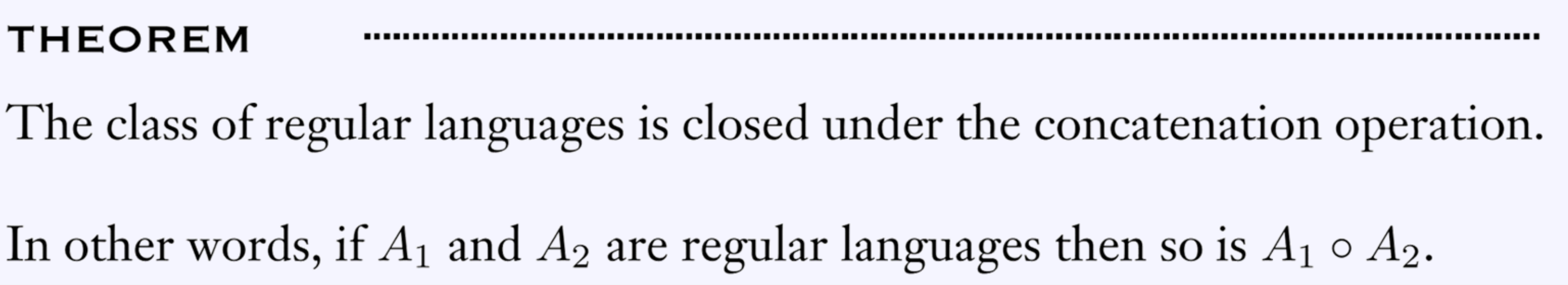
Last Time: Concatenation of Languages

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

If $A = \{\mathtt{fort}, \mathtt{south}\}$ $B = \{\mathtt{point}, \mathtt{boston}\}$

$$A \circ B = \{\mathtt{fortpoint}, \mathtt{fortboston}, \mathtt{southpoint}, \mathtt{southboston}\}$$

Last Time: Is Concatenation Closed?



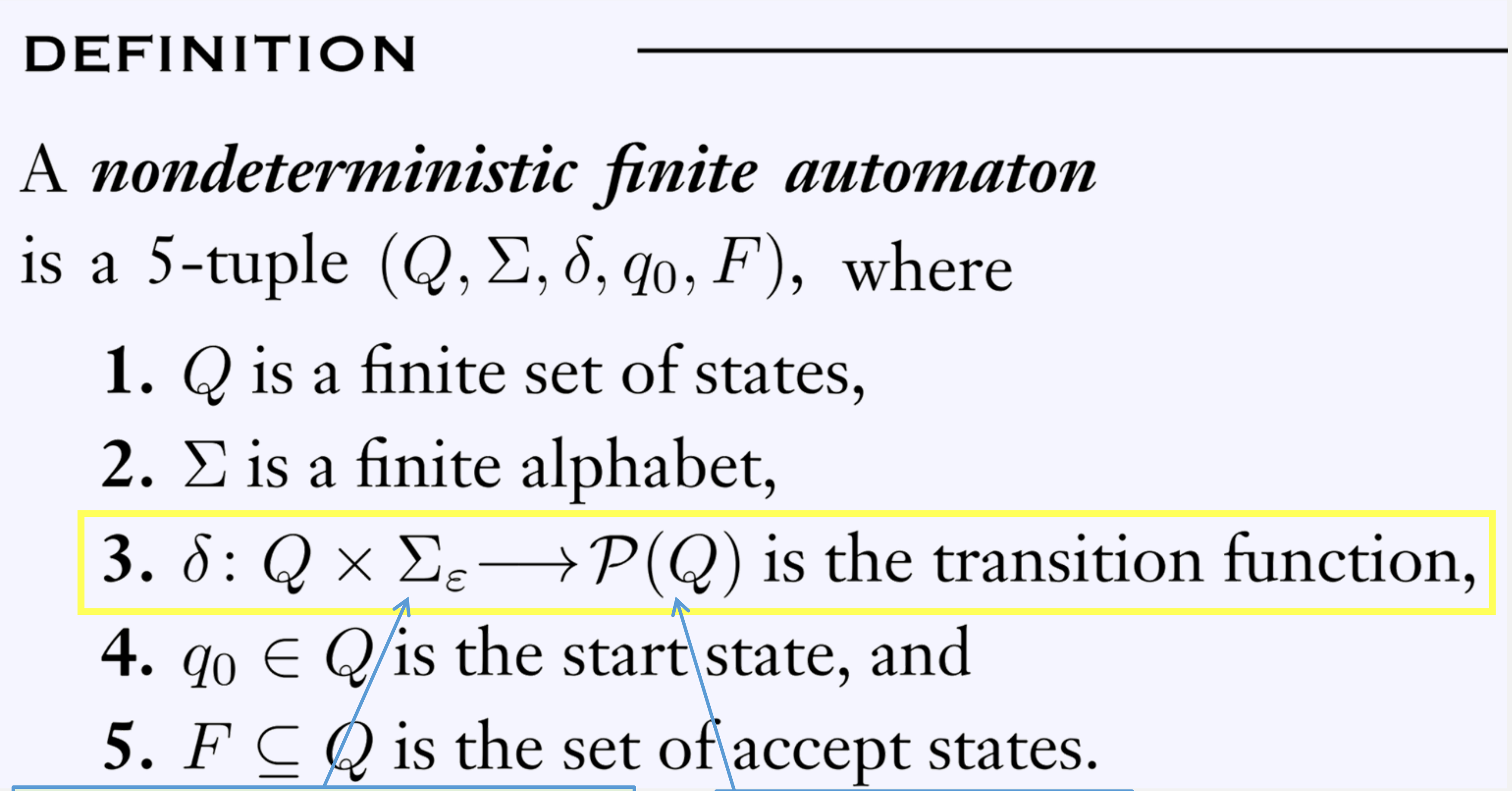
THEOREM

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 $A_1 \circ A_2$.

- Cannot? **combine** A_1 and A_2**'s machine to make a DFA because:**
  - **Unclear when to switch?** (can only read input once)
- **Need a different kind of machine!**

Last Time: NFA Formal Definition



DEFINITION

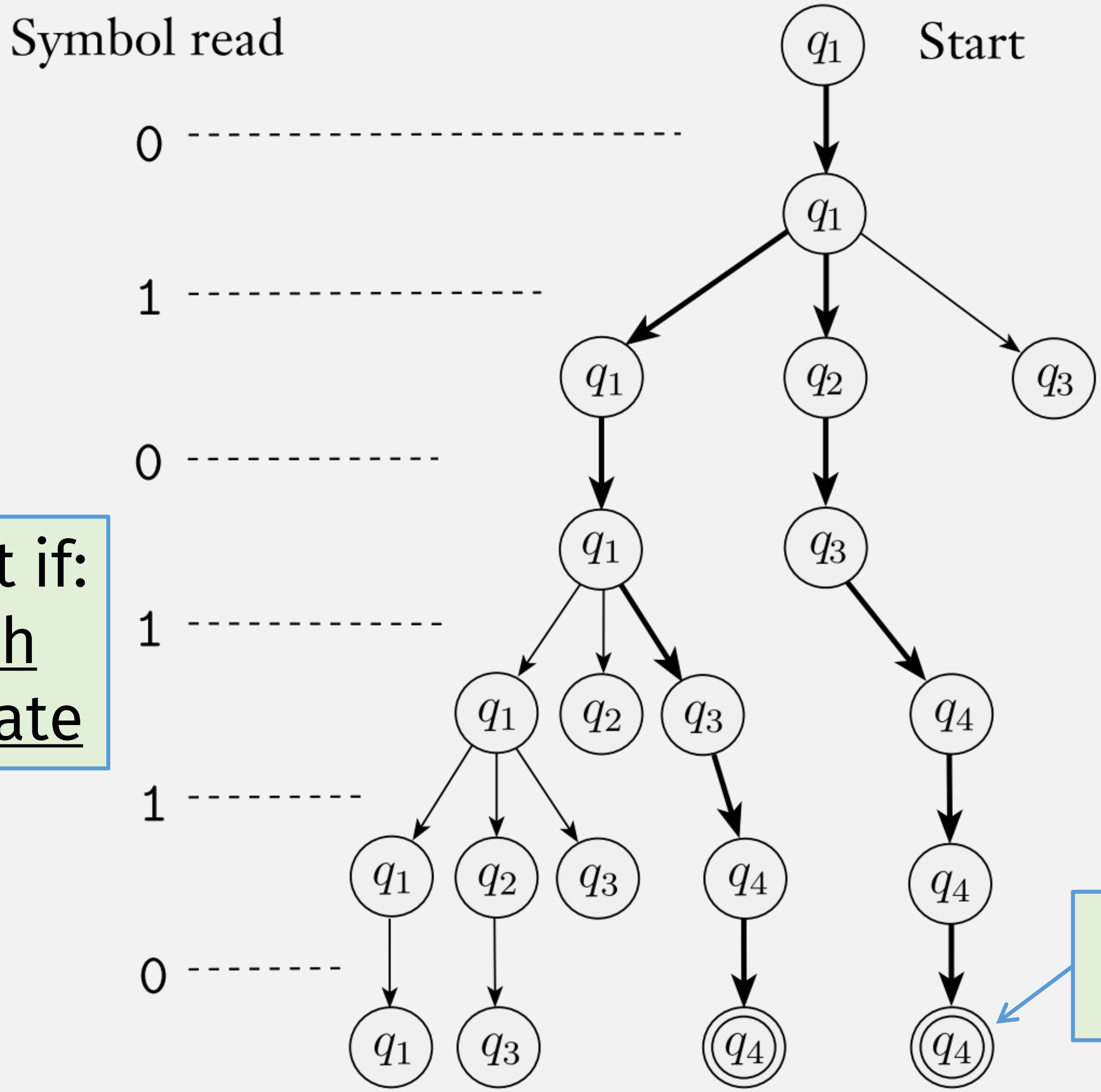
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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.

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

Transition results in a <u>set of states</u>

Last Time: 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

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

Machine “accepts” input if:

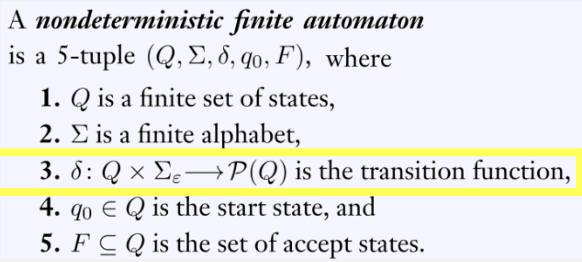
- Start in “start state”
  (and states connected to start state with ε transitions)
- Repeat:
  - Read 1 char;
  - Change states according to the transition table
- Result =
  - Last states have an “Accept” state

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 $i = 1, \ldots, n$
- $r_n \in F$



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.

NFA

~~*Flashback:* DFA~~ Computation Model

Informally

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

Machine “accepts” input if:

- Start in “start state”
  (and states connected to start state with ε transitions)
- Repeat:
  - Read 1 char;
  - Change states according to the transition table
- Result =
  - Last states have an “Accept” state

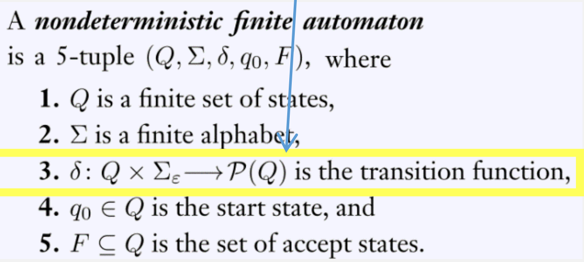
Formally (i.e., mathematically)

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

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

Next states is now a set



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:** $\hat{\delta} : Q \times \Sigma^* \to Q$

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)

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

First chars

("smaller argument")

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

last char

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

Recursive call

Single transition step

$\delta : Q \times \Sigma_{\varepsilon} \longrightarrow \mathcal{P}(Q)$ is the transition function

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 $w_i \in \Sigma$

Result is set of states

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

Result is set of states

Domain:
- Beginning state $q \in Q$
- Input string $w = w_1 w_2 \cdots w_n \quad \text{where } w_i \in \Sigma$

Range:
- Ending ~~state~~ set of states

(Defined recursively, on length of input string)

Empty string

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

nonEmpty string

Single transition steps for last char

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

Recursive call on first chars (smaller argument)

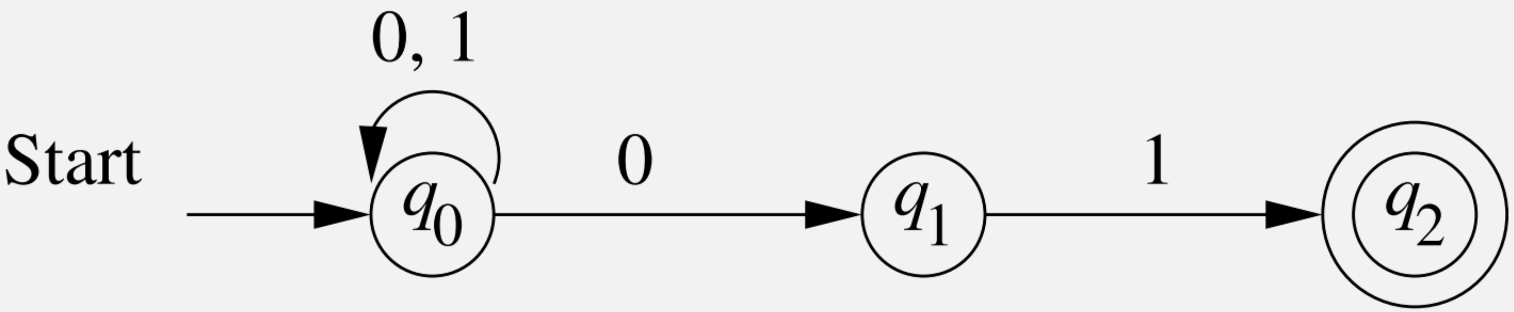
$\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$

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

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

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

NFA Extended δ Example



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

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

Combine result of recursive call with "last step"

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

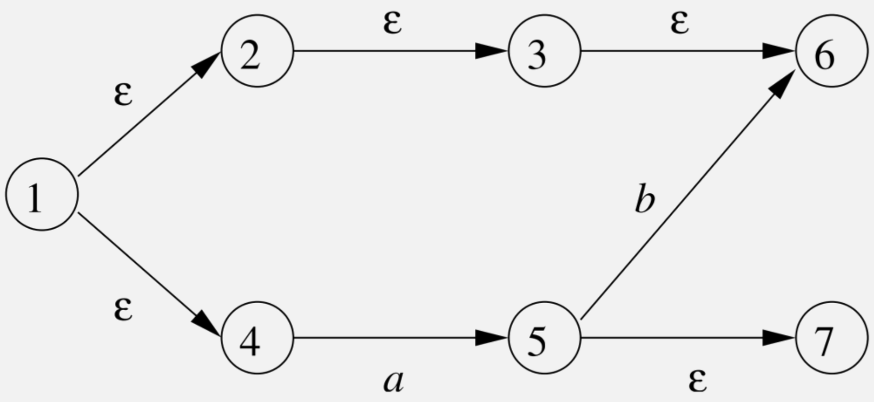
- <u>Inductive</u> case:

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

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)

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

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

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 $w = w_1 w_2 \cdots w_n \quad \text{where } w_i \in \Sigma$

Range:

- Ending set of states

(Defined recursively, on length of input string)

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

“Take single step, then follow all empty transitions”

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

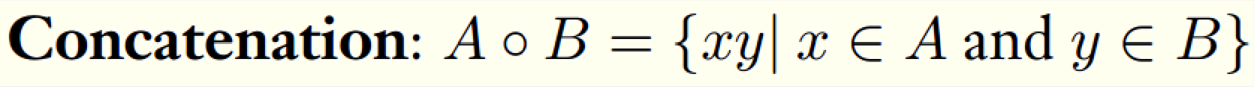
Summary: NFA vs DFA Computation

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

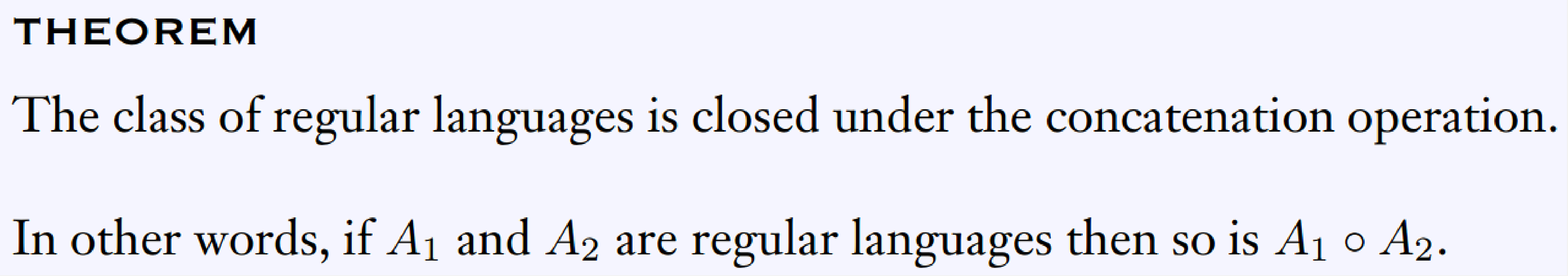
NFAs

- 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 is Closed?



THEOREM

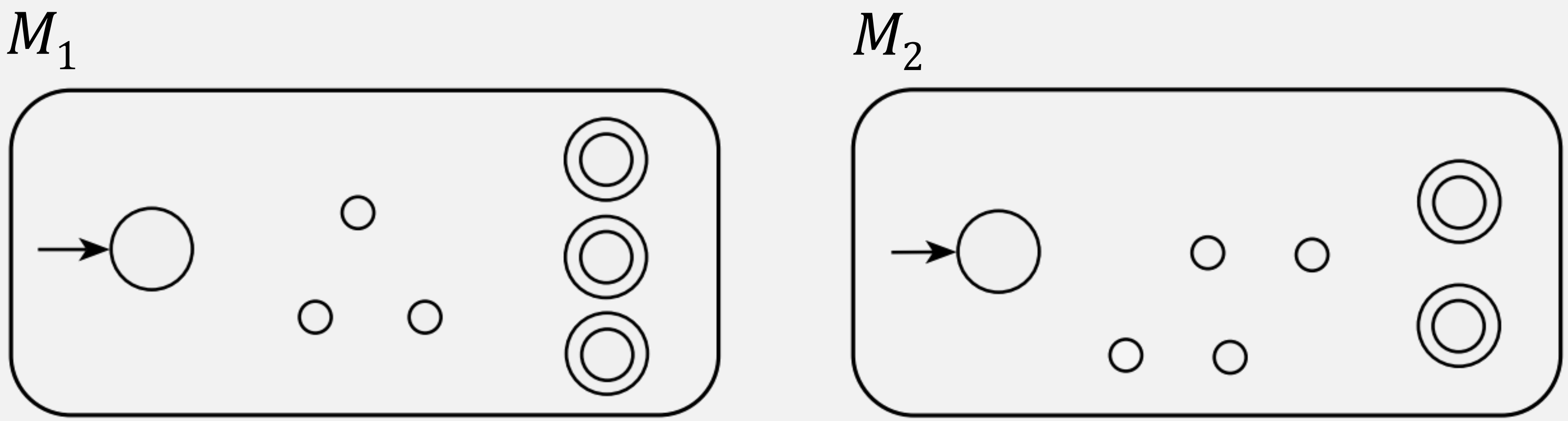
The class of regular languages is closed under the concatenation operation.

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 M_1 recognize A_1, and M_2 recognize A_2.

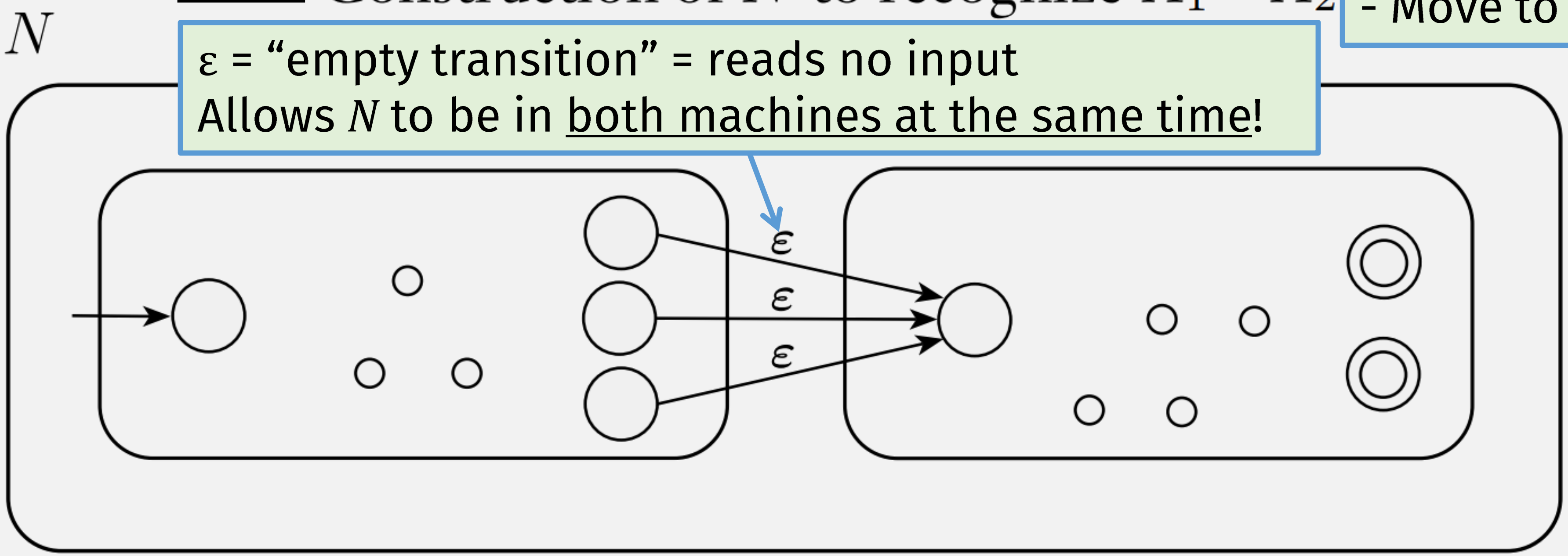
Want: Construction of N to recognize $A_1 \circ A_2$

N is an **NFA**! Now it can:

- Keep checking 1st part with M_1

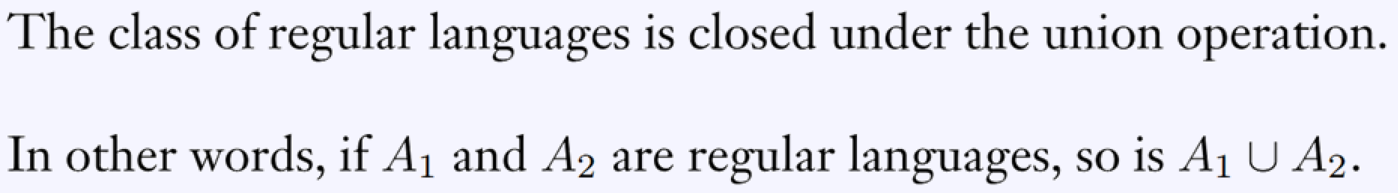
and

- Move to M_2 to check 2nd part

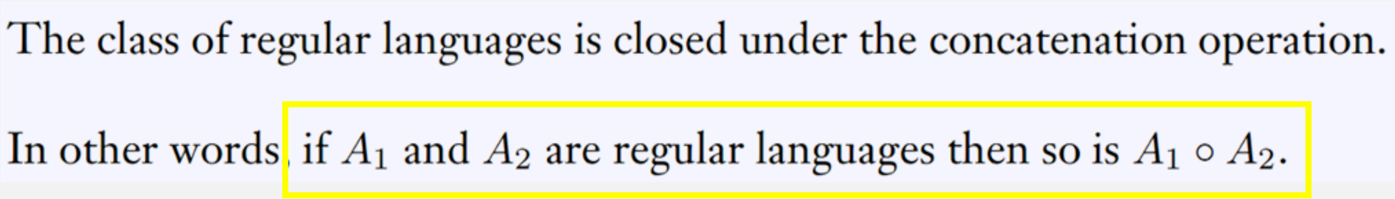


ε = "empty transition" = reads no input
Allows N to be in both machines at the same time!

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	**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 **NFA** $N =$ **???** (todo)	4. Def of **NFA**
5. N recognizes ~~$A_1 \cup A_2$~~ $\boxed{A_1 \circ A_2}$	5. See examples
6. $\boxed{A_1 \circ A_2}$ ~~$A_1 \cup A_2$~~ is a regular language	6. Does NFA recognize regular lang?
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 $A_1 \circ A_2$.	7. From stmt #1 and #6

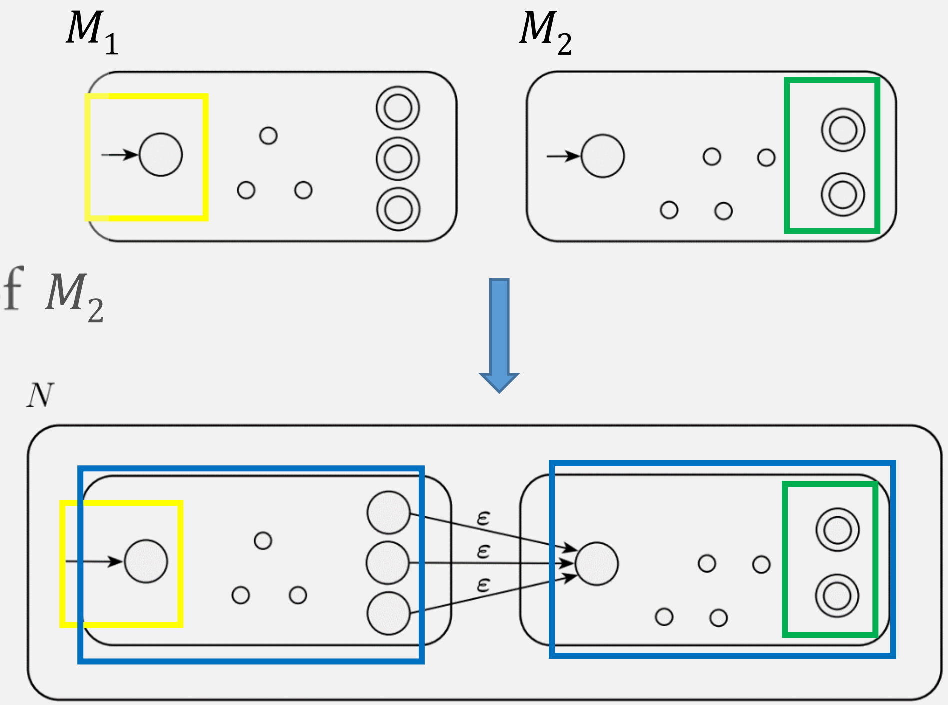
Concatenation is Closed for Regular Langs

PROOF

Let DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ 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 M_1
3. The accept states F_2 are the same as the accept states of M_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?

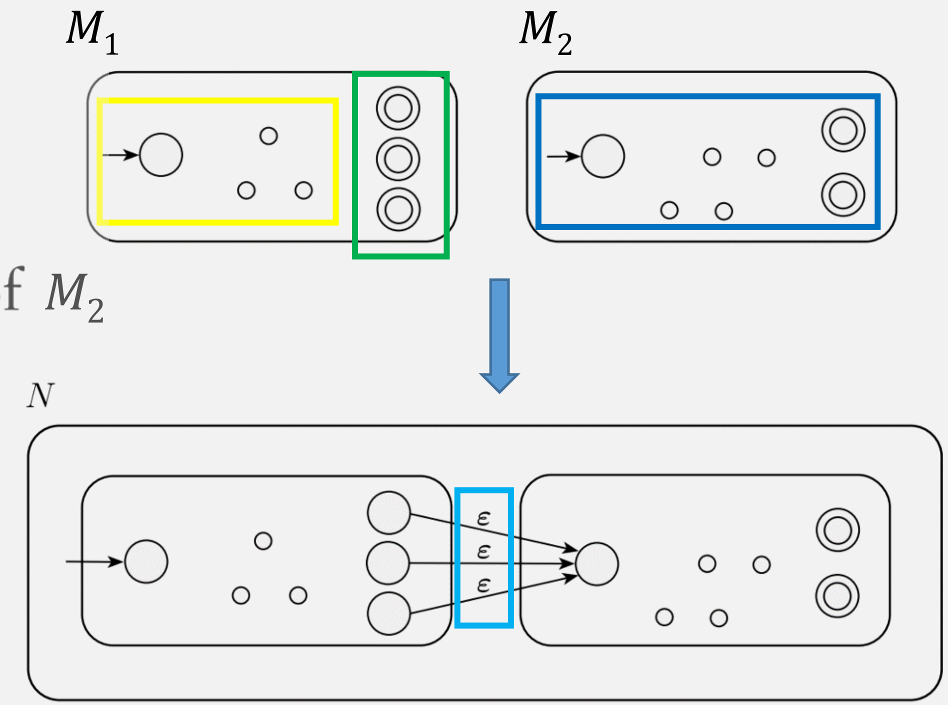
PROOF

Let DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ 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 M_1
3. The accept states F_2 are the same as the accept states of M_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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



??? ■

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

Definition: A DFA's language is a **regular language**

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., accept if final states contain <u>at least one </u>accept state

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

<u>Q</u>: What kind of languages do NFAs recognize?

Concatenation Closed for Reg Langs?

- **Combining DFAs to recognize concatenation of languages …**

  … produces an NFA

- So **to prove concatenation is closed …**

  … we must prove that NFAs *also* recognize regular languages.

Specifically, we must prove:

NFAs ⇔ regular languages

“If and only if” Statements

$X \Leftrightarrow Y$ = “X if and only if Y” = X iff Y = X <=> Y

Represents two statements:

1. ⇒ if X, then Y
   - “**forward**” direction

2. ⇐ if Y, then X
   - “**reverse**” direction

How to Prove an “iff” Statement

$X \Leftrightarrow Y$ = “X if and only if Y” = X iff Y = X <=> Y

Proof <u>at minimum</u> has 2 (If-Then proof) parts:

1. ⇒ if X, then Y
   - “**forward**” direction
   - assume X, then use it to prove Y
2. ⇐ if Y, then X
   - “**reverse**” direction
   - assume Y, then use it to prove X

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

⇒ If L is regular, then some NFA N recognizes it.

(Easier)

- We know: if L is **regular**, then a **DFA** exists that recognizes it.
- So to prove this part: Convert that DFA → an equivalent NFA! (see HW 2)

⇐ If an NFA N recognizes L, then L is regular.

Statements
&
Justifications?

"equivalent" =
"recognizes the same language"

⇒ If L is regular, then some NFA N recognizes it

Statements	**Justifications**
1. L is a regular language	1. Assumption
2. A DFA M recognizes L	2. Def of Regular language
3. Construct NFA N equiv to M	3. See hw 2
4. An NFA N recognizes L	4. ???
5. If L is a regular language, then some NFA N recognizes it	5. By Stmts #1 and #4

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

⇒ If L is regular, then some NFA N recognizes it.

(Easier)

- We know: if L is **regular**, then a **DFA** exists that recognizes it.
- So to prove this part: Convert that DFA → an equivalent NFA! (see HW 2)

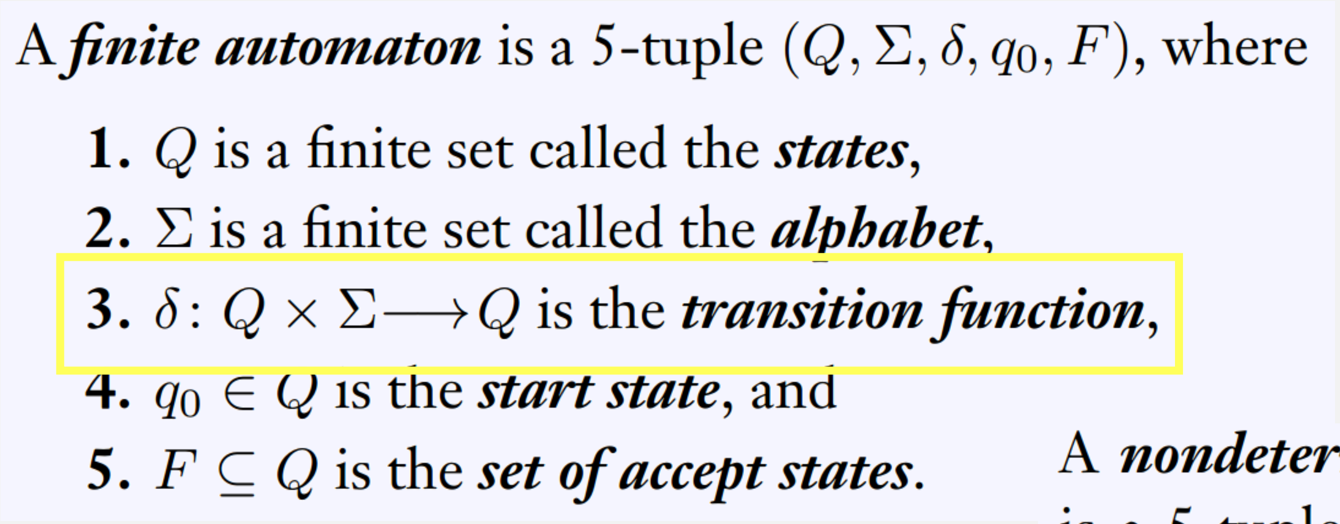
⇐ If an NFA N recognizes L, then L is regular.

(Harder)

- We know: for L to be **regular**, there must be a **DFA** recognizing it
- Proof Idea for this part: Convert given NFA N → an equivalent DFA

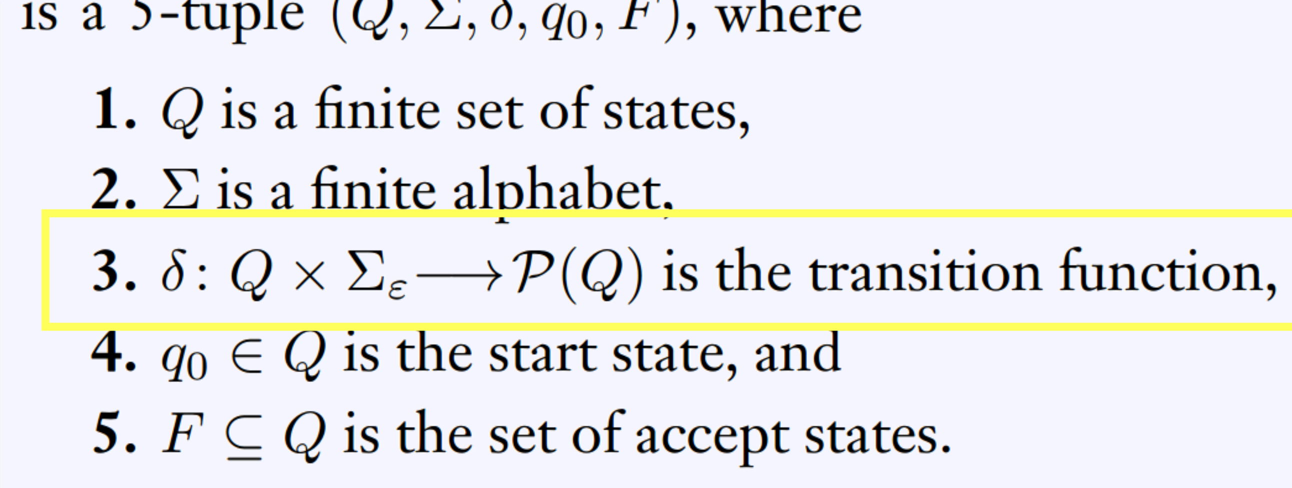
"equivalent" =
"recognizes the same language"

How to convert **NFA→DFA**?



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

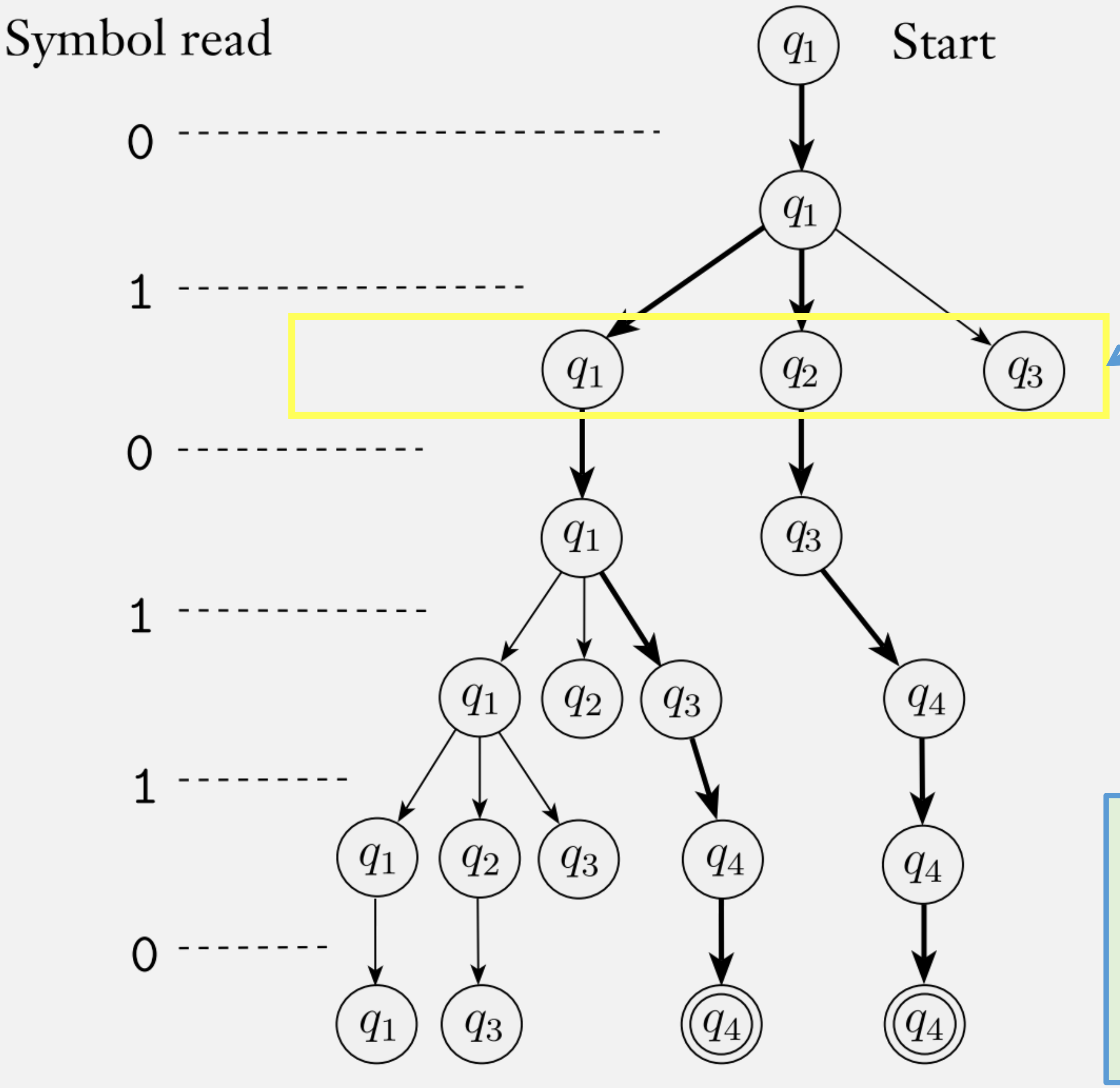
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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.

Proof idea:
Let each "state" of the DFA
= set of states in the NFA



NFA computation can be in multiple states

DFA computation can only be in one state

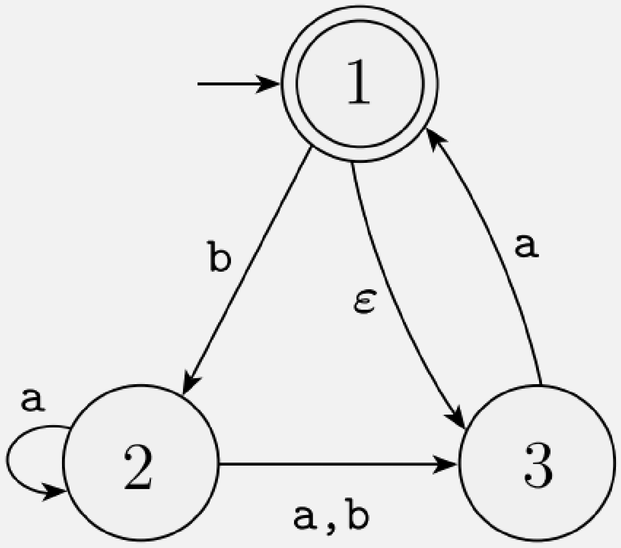
So encode:
a set of NFA states
as one DFA state

This is similar to the proof strategy from **"Closure of union"** where: **a state = a pair of states**

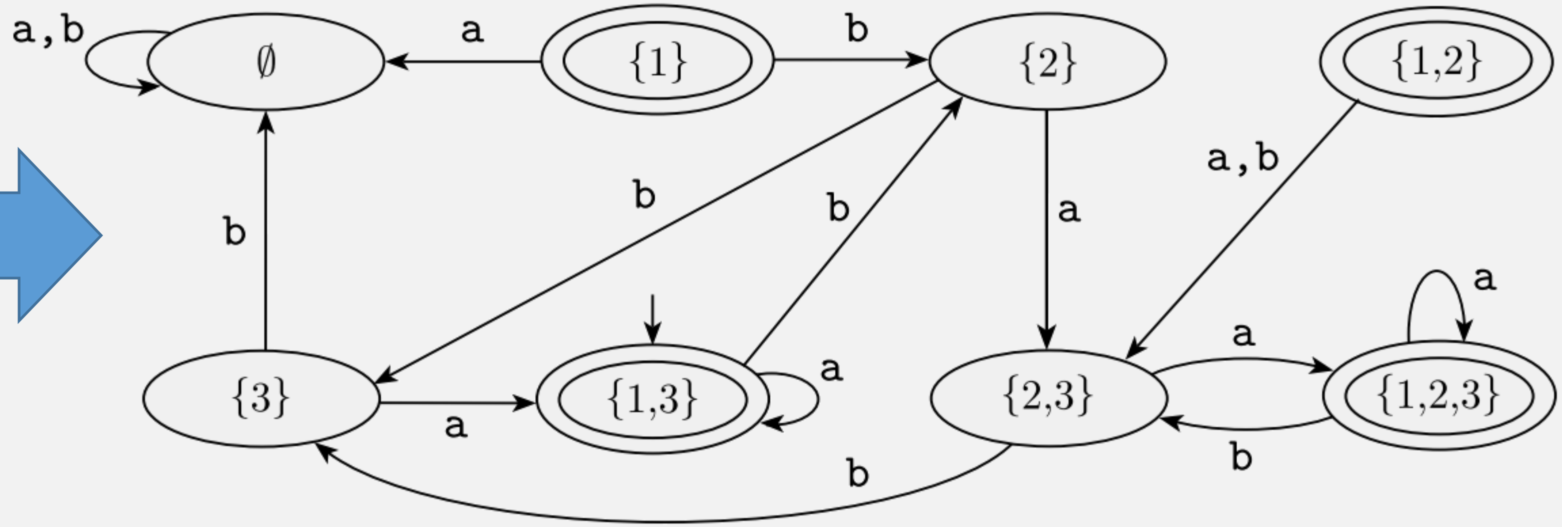
Convert **NFA→DFA**, Formally

- 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

No empty transitions

NFA→DFA

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

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

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

A DFA state = a set of NFA states

$\mathbf{2.}$ **For** $R \in Q'$ **and** $a \in \Sigma$**,**

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

A DFA step = an NFA step for all states in the set

R = DFA state = set of NFA states

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

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

Flashback: 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:

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

With empty transitions

NFA→DFA

Almost the same, except …

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

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

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

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

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

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

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

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

⇒ If L is regular, then some NFA N recognizes it.

(Easier)

- We know: if L is **regular**, then a **DFA** exists that recognizes it.
- So to prove this part: Convert that DFA → an equivalent NFA! (see HW 2)

⇐ If an NFA N recognizes L, then L is regular.

(Harder)

- We know: for L to be **regular**, there must be a **DFA** recognizing it
- Proof Idea for this part: Convert given NFA N → an equivalent DFA …

… using our NFA to DFA algorithm! ■

Concatenation is Closed for Regular Langs

PROOF

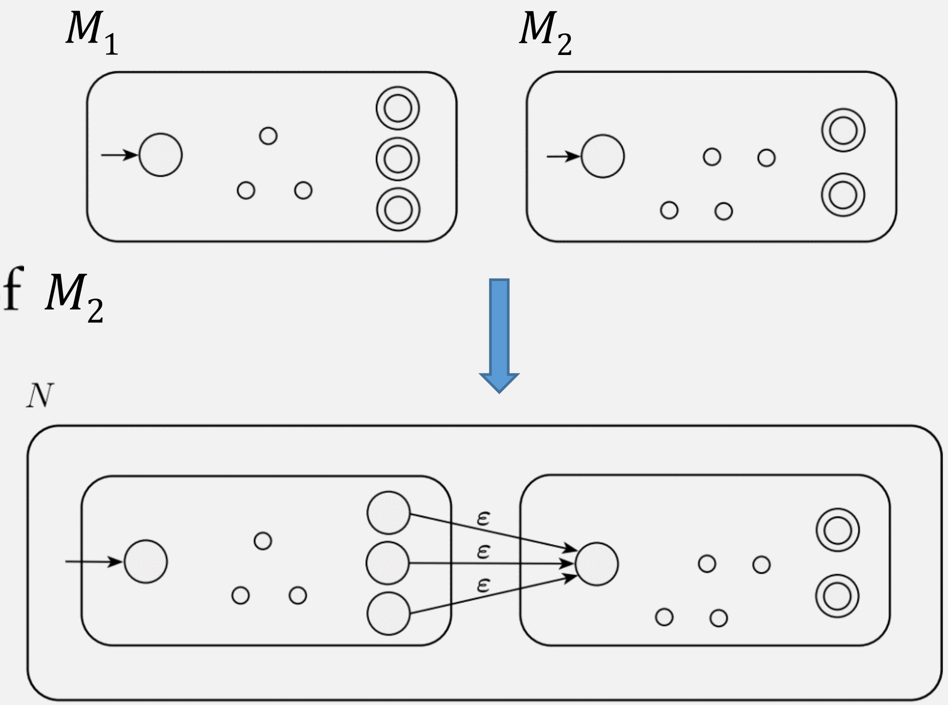
Let DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

If a language has an NFA recognizing it, then it is a regular language

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 M_1
3. The accept states F_2 are the same as the accept states of M_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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



☑ ~~???~~ ■

Concat Closed for Reg Langs: Use NFAs Only

PROOF

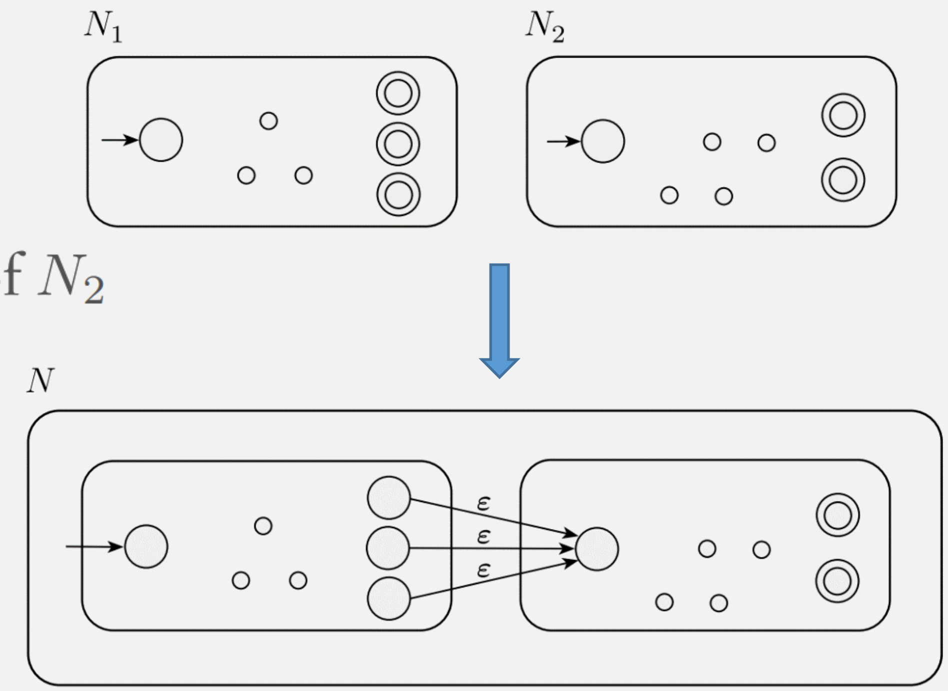
If language is regular,
then it has an **NFA** recognizing it …

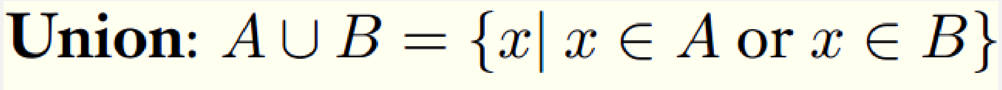
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

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 $a \in \Sigma_\varepsilon$,

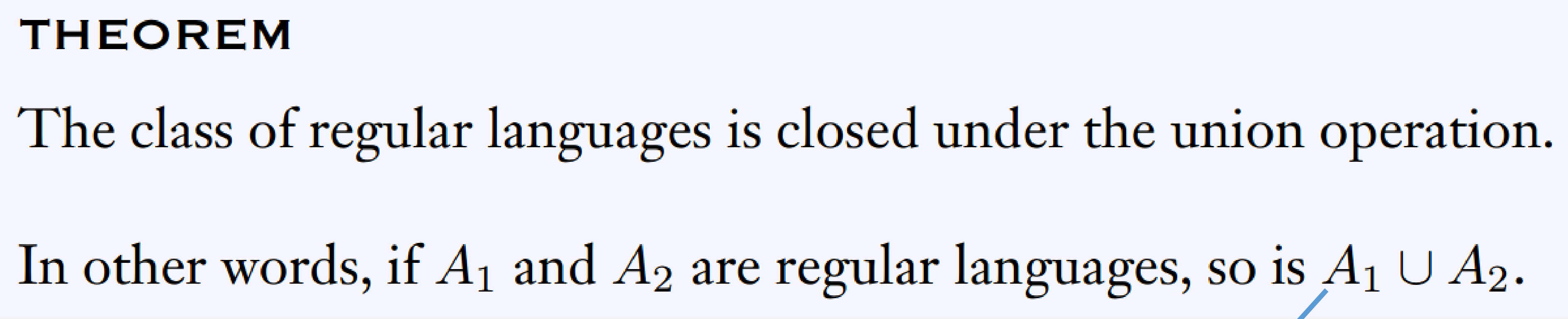
$$\delta(q,a) = \begin{cases} ? \;\; \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ ? \;\; \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ ? \;\; \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ ? \;\; \delta_2(q,a) & q \in Q_2. \end{cases}$$





Union: $A \cup B = \{x | \, x \in A \text{ or } x \in B\}$

Flashback: Union is Closed For Regular Langs



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 $A_1 \cup A_2$.

Proof:

- How do we prove that a language is regular?
  - Create a DFA or NFA recognizing it!
- Combine the machines recognizing A_1 and A_2
  - Should we create a DFA or NFA?

Proof, with DFA

Flashback: 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,
- Construct: **a new machine** $M = (Q, \Sigma, \delta, q_0, F)$ **using** ***M₁*** **and** ***M₂***
- **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)
- ***M* accept states:** $F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}$

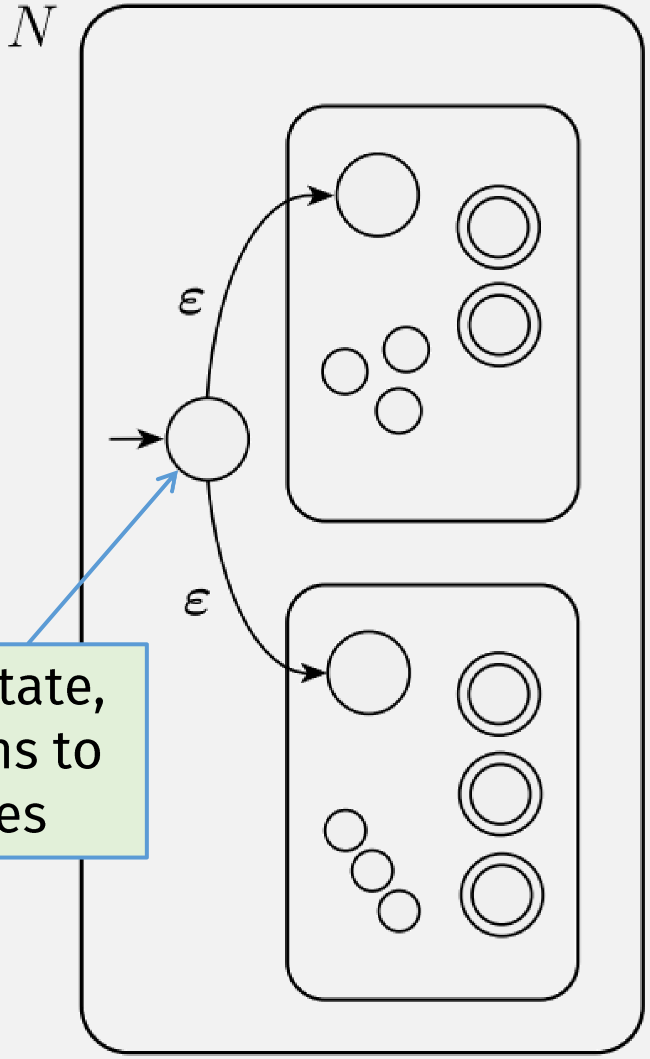
State in *M* =
M_1 state +
M_2 state

M step =
a step in M_1 + a step in M_2

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

Alternate Proof, with NFAs

Union is Closed for Regular Languages



Add new start state, and ε-transitions to old start states

Alternate Proof, with NFAs

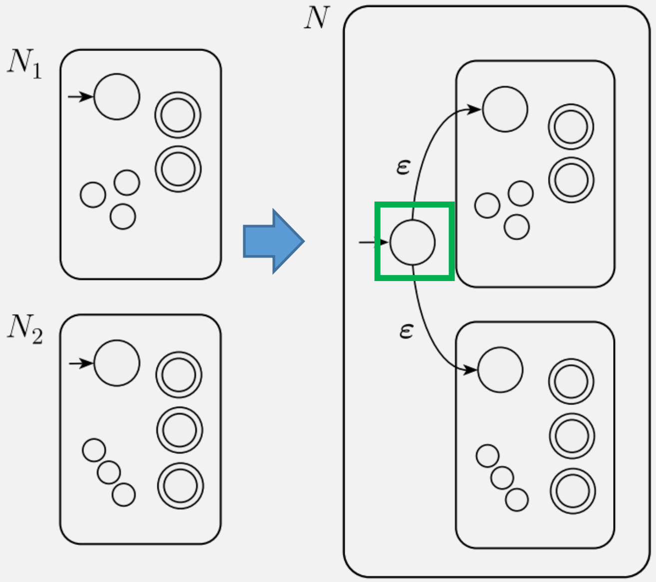
Union is Closed for Regular Languages

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.

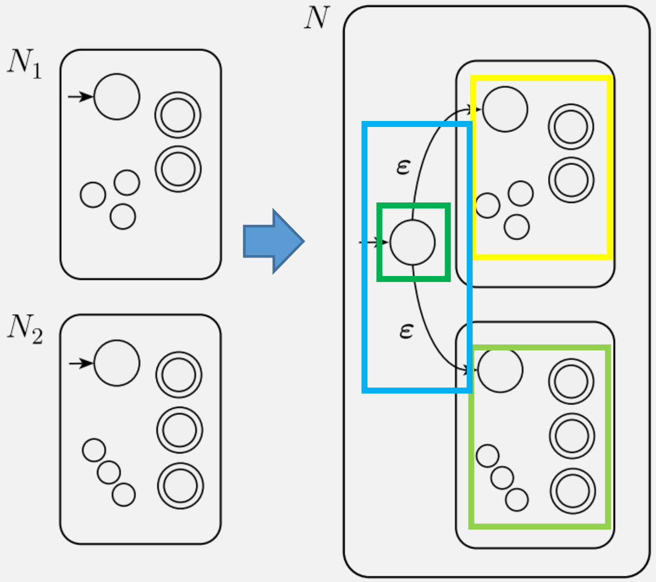
Construct $N = (Q, \Sigma, \delta, \boxed{q_0,} F)$ to recognize $A_1 \cup A_2$.

1. $Q = \boxed{\{q_0\}} \cup Q_1 \cup Q_2$.
2. The state $\boxed{q_0}$ is the start state of N.
3. The set of accept states $F = F_1 \cup F_2$.



Alternate Proof, with NFAs

Union is Closed for Regular Languages



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 $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.

2. The state q_0 is the start state of N.

3. The set of accept states $F = F_1 \cup F_2$.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(?, a) & \boxed{q \in Q_1} \\ \delta_2(?, a) & \boxed{q \in Q_2} \\ \{q_1 ? q_2\} & \boxed{q = q_0 \text{ and } a = \varepsilon} \\ \emptyset \quad ? & \boxed{q = q_0 \text{ and } a \neq \varepsilon} \end{cases}$$

List of Closed Ops for Reg Langs (so far)

- ☑ Union
- ☑ Concatentation
- Kleene Star (repetition)

Star: $A^* = \{x_1x_2\dots x_k | \; k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star Example

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

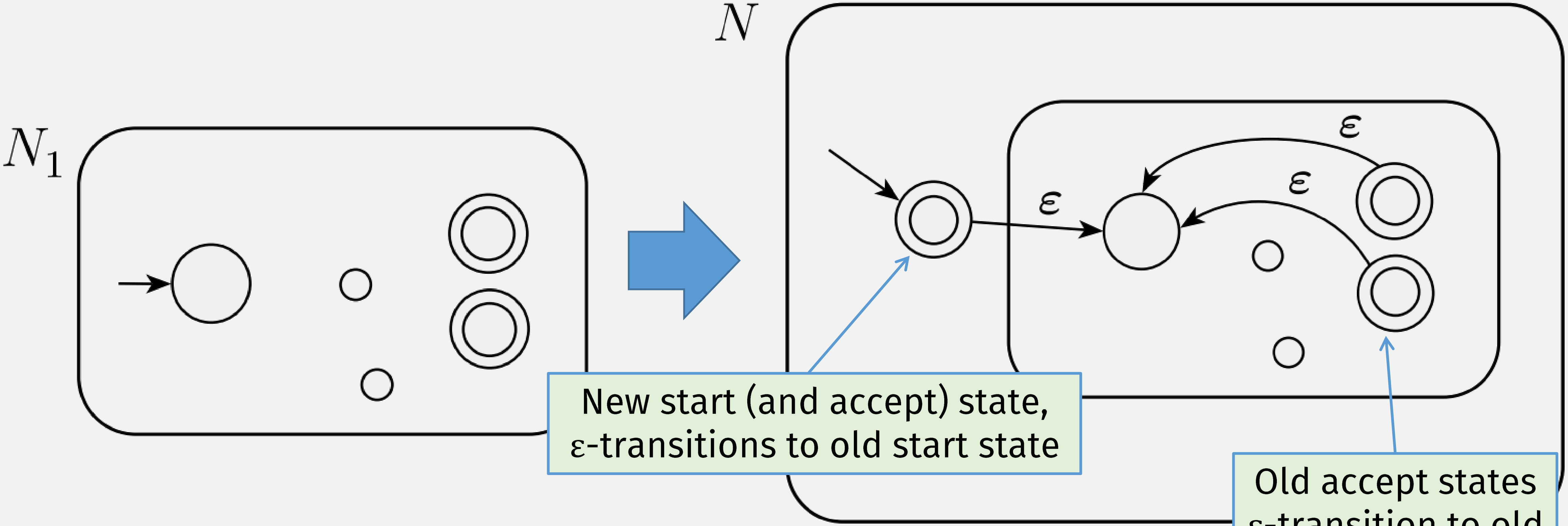
If $A = \{\mathtt{good}, \mathtt{bad}\}$

$$A^* = \begin{array}{l}\{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad,} \\ \text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, } \dots\}\end{array}$$

Note: **repeat <u>zero or more times</u>**

(this is an infinite language!)

Kleene Star

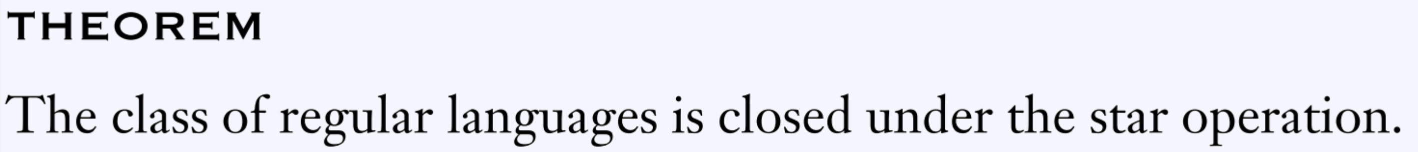


New start (and accept) state, ε-transitions to old start state

Old accept states ε-transition to old start state

In-class exercise:

Kleene Star is Closed for Regular Langs

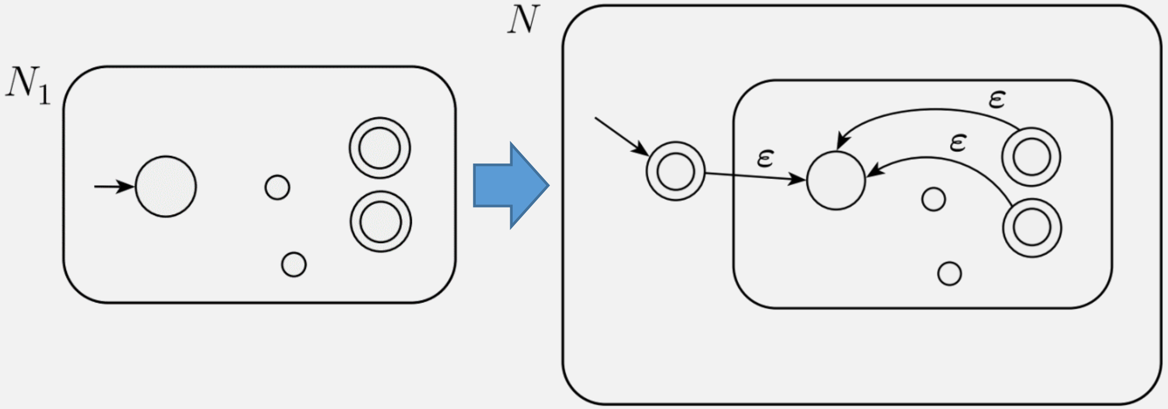


THEOREM

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

Kleene Star is Closed for Regular Langs

PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

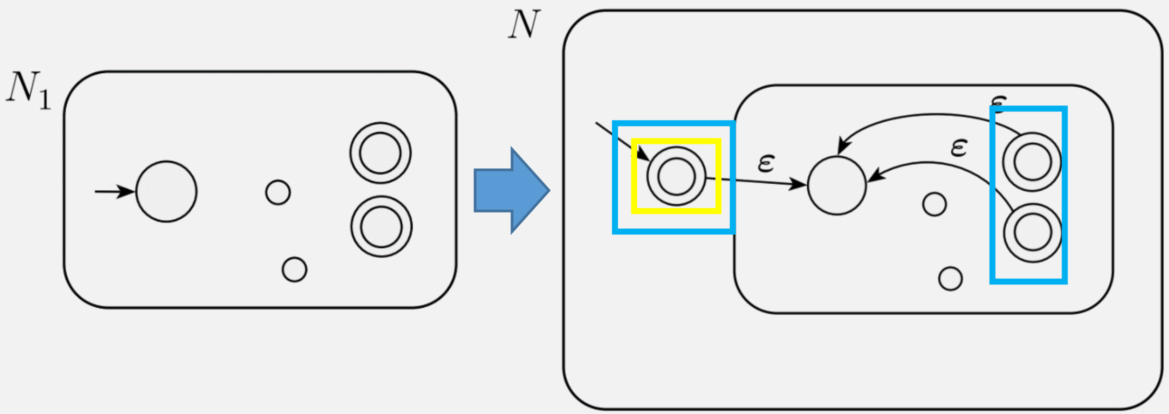


Kleene Star is Closed for Regular Langs

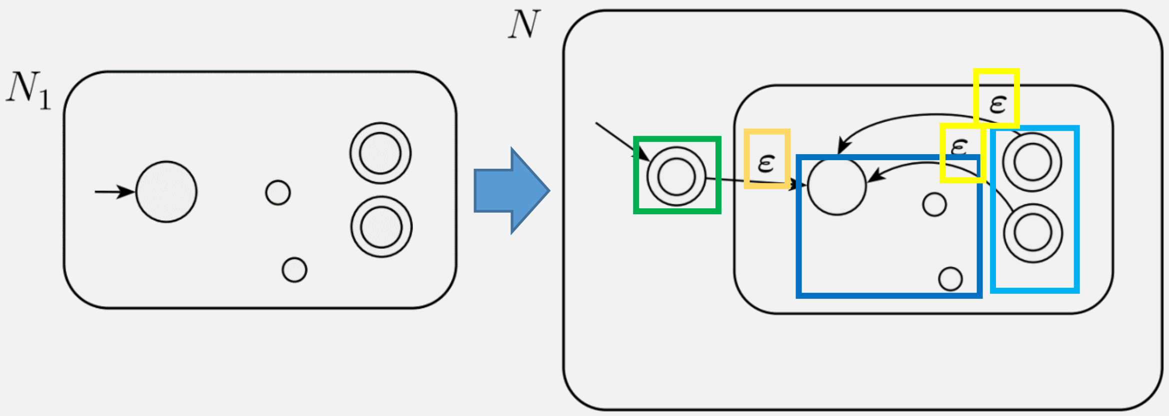
PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = \boxed{\{q_0\}} \cup Q_1$
2. The state $\boxed{q_0}$ is the new start state.
3. $F = \boxed{\{q_0\} \cup F_1}$

Kleene star of a language must accept the empty string!



Kleene Star is Closed for Regular Langs



PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = \{q_0\} \cup Q_1$
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) \,? & \boxed{q \in Q_1 \text{ and } q \notin F_1} \\ \delta_1(q,a) \,? & \boxed{q \in F_1 \text{ and } a \neq \varepsilon} \\ \delta_1(q,a) \,? \cup \{q_1\} & q \in F_1 \text{ and } \boxed{a = \varepsilon} \\ \{q_1\} \quad ? & q = q_0 \text{ and } \boxed{a = \varepsilon} \\ \emptyset \quad ? & \boxed{q = q_0 \text{ and } a \neq \varepsilon.} \end{cases}$$

Many More Closed Operations on Regular Languages!

- Complement
- Intersection
- Difference
- Reversal
- Homomorphism
- (See HW2)

Check-in Quiz 2/13

On gradescope