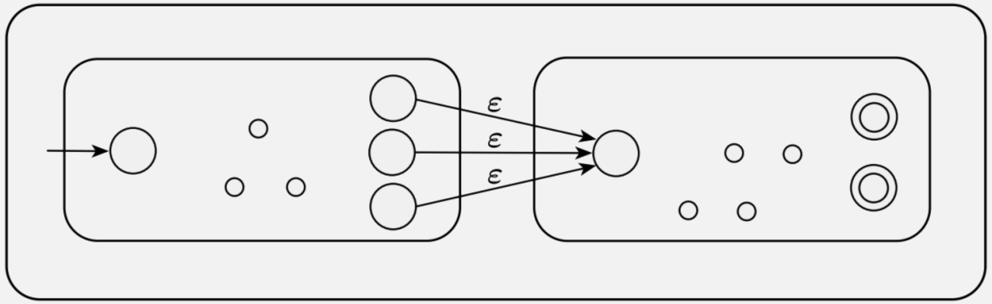
CS 622

# Regular Languages Are Closed Under Concatenation

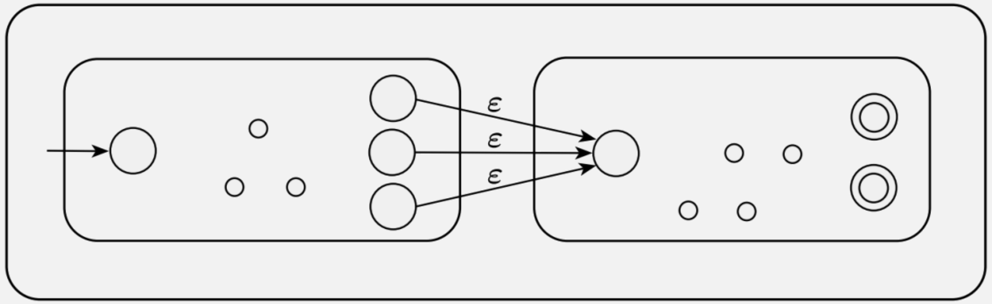
Friday, February 23, 2024

UMass Boston CS



# Announcements

- HW 3 out
  - Due Mon 3/4 12pm EST (noon)



Previously

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

# DFA Extended Transition Function

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

- <u>Domain</u> (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- <u>Range</u> (output):
  - state $q \in Q$ (doesn't have to be an accept state)

Previously

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

# NFA Extended Transition Function

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

Result is set of states

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):

states $qs \subseteq Q$

Previously

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

# NFA Extended Transition Function

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

Result is set of states

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

Recursively Defined Input
needs
Recursive Function

Base case

A **String** is either:

- the **empty string** (ε), or
- *xa* (**non-empty string**) where
  - *x* is a **string**
  - *a* is a "**char**" in Σ

(Defined recursively)

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

Previously

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

# NFA Extended Transition Function

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

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

Recursively Defined Input
needs
Recursive Function

A **String** is either:
- the **empty string** (ε), or
- *xa* (non-empty string) where
  - *x* is a **string**
  - *a* is a "char" in Σ

(Defined recursively)

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

Recursive Case $\hat{\delta}(q, w'w_n) =$

where $w' = w_1 \cdots w_{n-1}$

Recursive case

Recursive part

Recursion on recursive part

"second to last" set of states

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

Previously

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

# NFA Extended Transition Function

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

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

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

Recursively Defined Input
needs
Recursive Function

A **String** is either:

- the **empty string** (ε), or
- *xa* (**non-empty string**) where
  - *x* is a **string**
  - *a* is a "**char**" in Σ

Last char

(Defined recursively)

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

For each "second to last" state, take single step on last char

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

where $w' = w_1 \cdots w_{n-1}$

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

*Previously*

# 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

Handling ε transitions now!

# NFA Extended Transition Function

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

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

where $w' = w_1 \cdots w_{n-1}$

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

$$\bigcup_{i=1}^{k} \delta(q_i, w_n) = \{r_1, \ldots, r_\ell\}$$

(Defined recursively)

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

Recursive Case $\hat{\delta}(q, w'w_n) =$

Handling ε transitions now!

# NFA Extended Transition Function

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

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

(Defined recursively)

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

Recursive Case $\hat{\delta}(q, w' w_n) = \bigcup_{j=1}^{\ell} \varepsilon\text{-REACHABLE}(r_j)$

All chars except last

"second to last" set of states

where $w' = w_1 \cdots w_{n-1}$

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

$\bigcup_{i=1}^{k} \delta(q_i, w_n) = \{r_1, \ldots, r_\ell\}$

"last" set of states (no ε)

# Summary: NFA vs DFA Computation

**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
  - Has empty transitions
- Acceptance:
  - If one of final states is accept state

*Previously*

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

# Is Concatenation Closed?

**THEOREM**

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

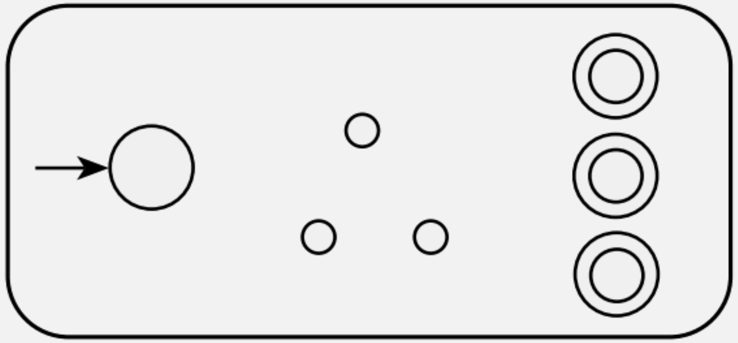
In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

*Proof requires*: Constructing new machine

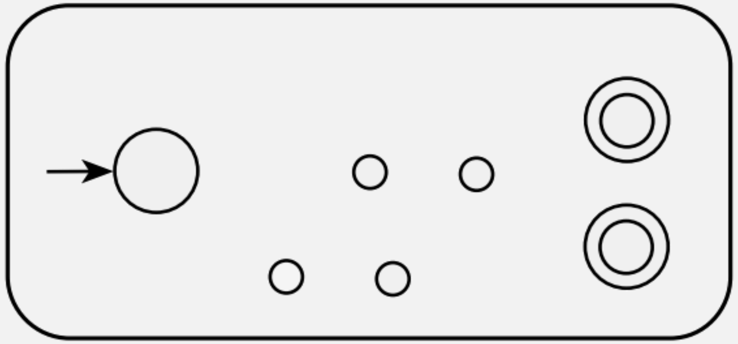
- How does it know when to switch machines?
  - Can only read input once

## Concatentation

$M_1$



$M_2$



Let $M_1$ recognize $A_1$, and $M_2$ recognize $A_2$.

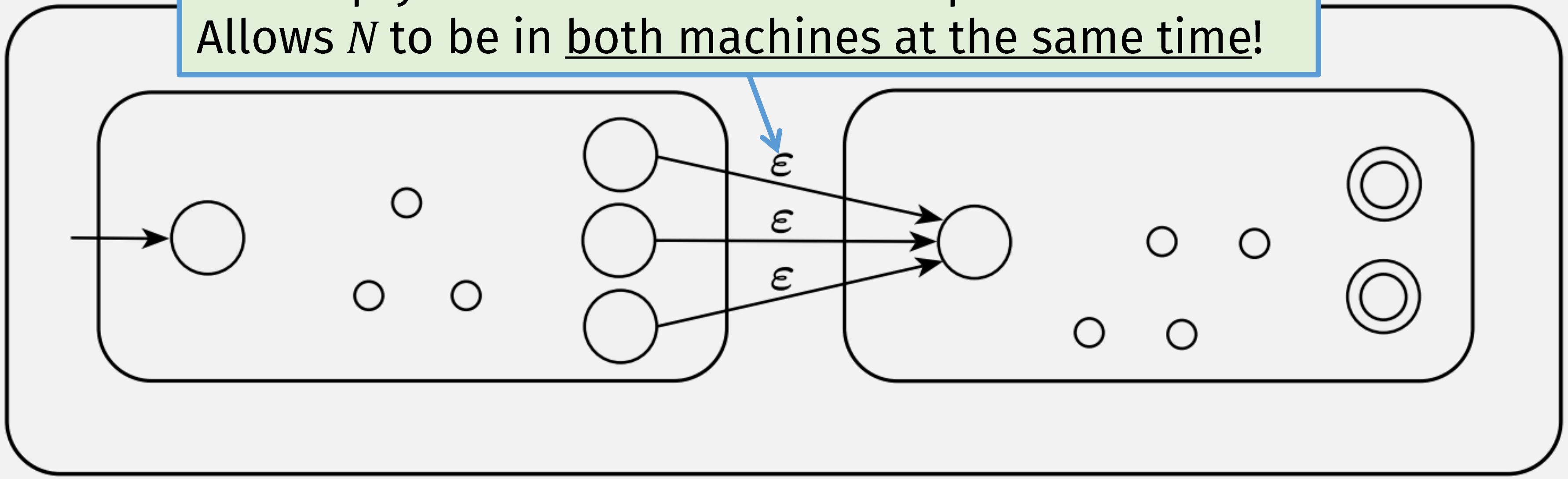
Want: Construction of $N$ to recognize $A_1 \circ A_2$

$N$ is an **NFA**! It can:
- Keep checking 1st part with $M_1$

  and
- Move to $M_2$ to check 2nd part

$N$

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



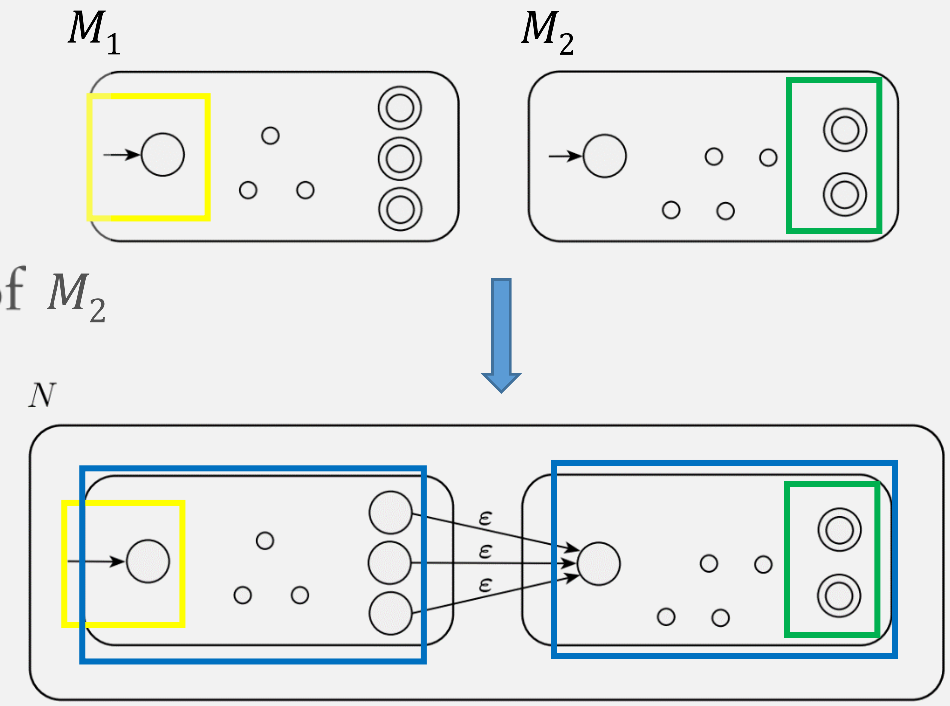
# Concatenation is Closed for Regular Langs

**PROOF** (part of)

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 $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

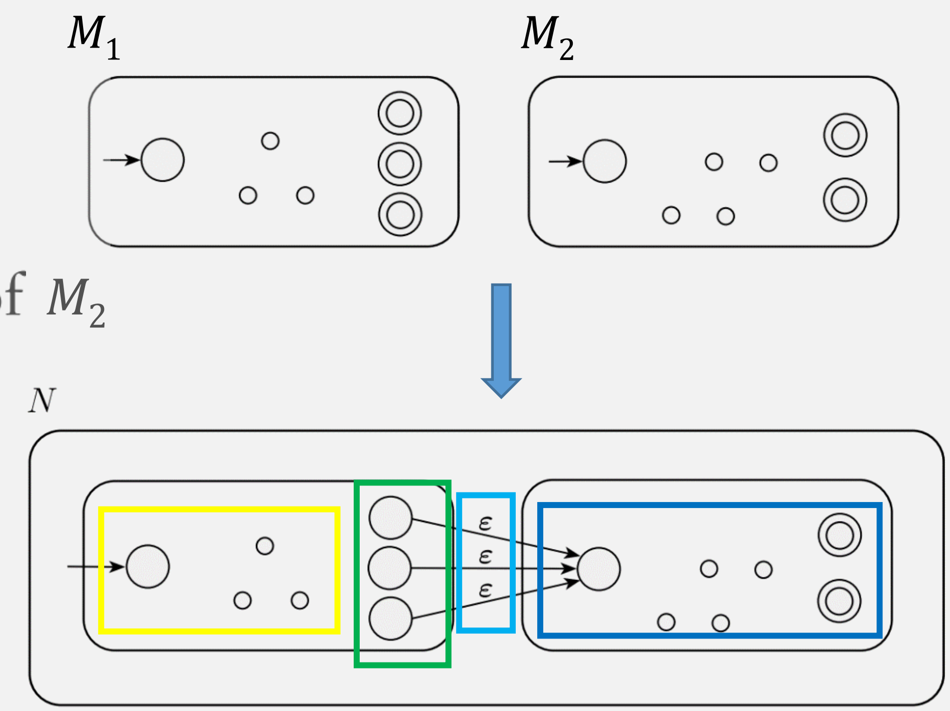


# Concatenation is Closed for Regular Langs

**PROOF** (part of)

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 $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

NFA def says $\delta$ must map every state and $\varepsilon$ to set of states

And: $\delta(q, \varepsilon) = \emptyset$, for $q \in Q, q \notin F_1$

# Concatenation is Closed for Regular Langs

Wait, is this true?

**PROOF** (part of)

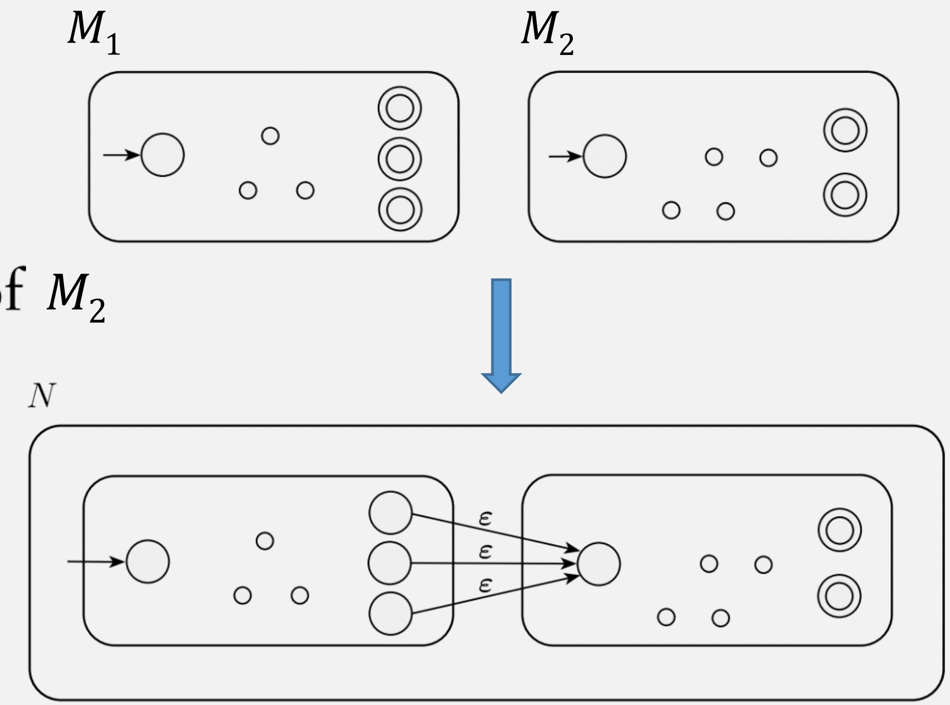
Let DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$
DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$

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 $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

And: $\delta(q, \varepsilon) = \emptyset$, for $q \in Q, q \notin F_1$




??? ∎

*Previously*

# Is Union Closed For Regular Langs?

Proof

| **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 Reg Lang (Coro) |
| 3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes $A_2$ | 3. Def of Reg Lang (Coro) |
| 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 Table |
| *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 |

Q.E.D. ■

# Is Concat Closed For Regular Langs?

Proof?

| 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 Reg Lang (Coro) |
| 3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes $A_2$ | 3. Def of Reg Lang (Coro) |
| 4. Construct NFA $M = (Q, \Sigma, \delta, q_0, F)$ | 4. Def of NFA |
| 5. $M$ recognizes ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$ | 5. See Examples Table |
| 6. ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$ is a regular language | 6. ??? Does NFA recognize reg langs? |
| 7. The class of regular languages is closed under t 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 |

Q.E.D.?

*Previously*

# 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** $N = (Q, \Sigma, \delta, q_0, F)$
- $N$ ***accepts*** $w$ if $\hat{\delta}(q_0, w) \cap F \neq \emptyset$

  Intersection … … with accept states … … is not empty set

  - i.e., accept if final states contains at least one accept state
- Language of $N$ = $L(N)$ = $\left\{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$

Q: 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 <u>two</u> statements:

1. $\Rightarrow$ if $X$, then $Y$
   - “**forward**” direction

2. $\Leftarrow$ 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 has <u>two</u> (If-Then proof) parts:

1. $\Rightarrow$ if $X$, then $Y$
   - "**forward**" direction
   - assume $X$, then use it to prove $Y$
2. $\Leftarrow$ if $Y$, then $X$
   - "**reverse**" direction
   - assume $Y$, then use it to prove $X$

# NFA <-> DFA

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

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

# Proving NFAs Recognize Regular Langs

Theorem:

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

Proof: 2 parts

⇒ 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 3)

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

Full 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 lang (Coro) |
| 3. Construct NFA $N = \texttt{convert}(M)$ | 3. See hw ~~2~~ 3! |
| 4. DFA $M$ is equivalent to NFA $N$ | 4. See Equiv. table! |
| 5. An NFA $N$ recognizes $L$ | 5. ??? |
| 6. If $L$ is a regular language, then some NFA $N$ recognizes it | 6. By Stmts #1 and # 5 |

Assume the "if" part …

… use it to prove "then" part

# "Proving" Machine Equivalence (Table)

Let: DFA $M = (Q, \Sigma, \delta, q_0, F)$
NFA $N$ = convert($M$)
$\hat{\delta}(q_0, w) \in F$ for some string $w$

**Note:**
new required column

| String | $M$ accepts? | $N$ accepts? | $N$ accepts? Justification |
|---|---|---|---|
| $w$ | Yes | ??? | See justification #1 |
| $w'$ | No | ??? | See justification #2? |
| … | | | |

If $M$ accepts $w$ …

Then we know …

There is some sequence of states: $r_1 \ldots r_n$, where $r_i \in Q$ and

$r_1 = q_0$ and $r_n \in F$

Then $N$ accepts?/rejects? $w$ because …

Justification #1?
There is an accepting sequence of set of states in $N$ … for string $w$

# "Proving" Machine Equivalence (Table)

Let: DFA $M = (Q, \Sigma, \delta, q_0, F)$
NFA $N = \text{convert}(M)$
$\hat{\delta}(q_0, w) \in F$ for some string $w$
$\hat{\delta}(q_0, w') \in F$ for some string $w'$

| String | $M$ accepts? | $N$ accepts? | $N$ accepts? Justification |
|---|---|---|---|
| $w$ | Yes | ??? | See justification #1 |
| $w'$ | No | ??? | See justification #2? |
| … | | | |

**If $M$ accepts $w'$ …**

Then we know …

Then $N$ accepts?/rejects? $w'$ because …

Justification #2?

# 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 3)

⇐ 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**?

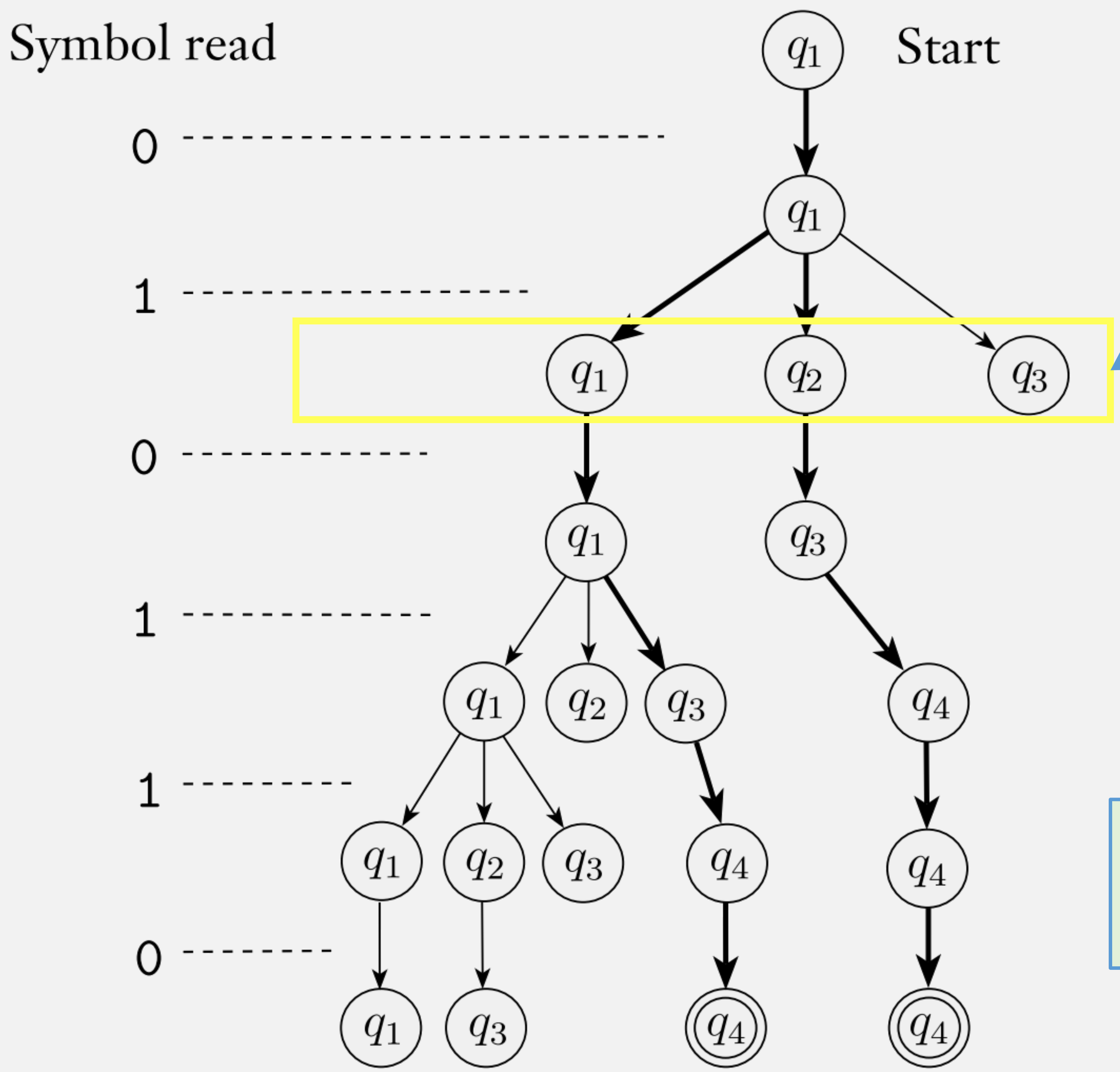
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the ***states***,
2. $\Sigma$ 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. $\Sigma$ 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

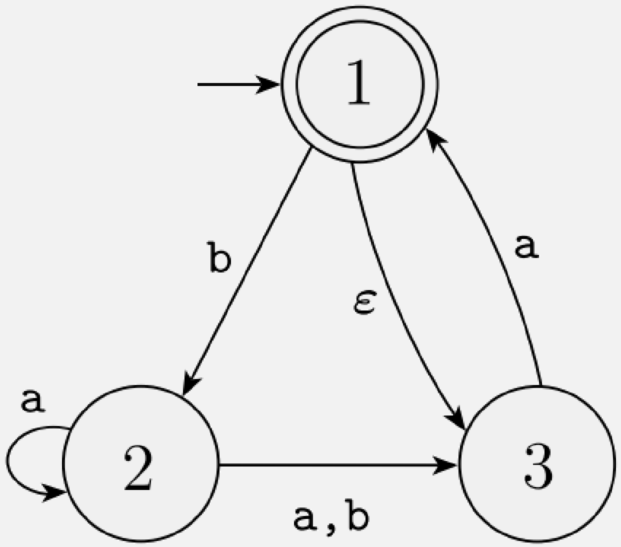
So **encode:**
a set of NFA states
as one DFA state

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

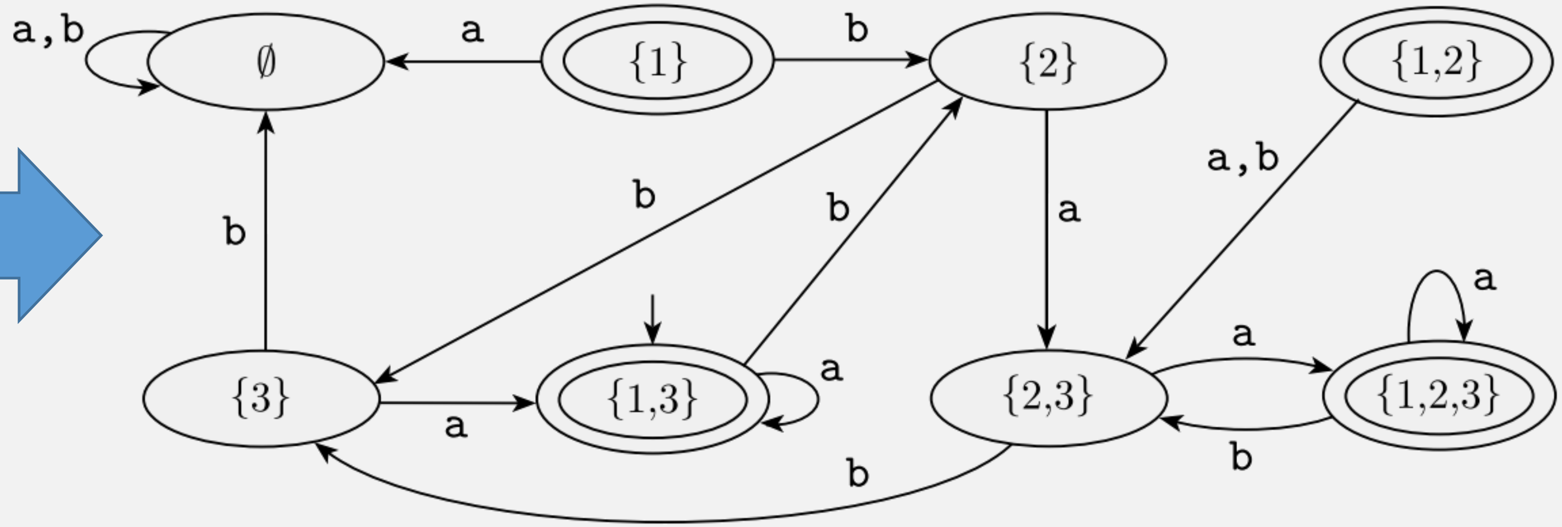
# Convert **NFA→DFA**, Formally

- 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 $M = (Q', \Sigma, \delta', {q_0}', F')$

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

A DFA state = a set of NFA states

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

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

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

- Recursive 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

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 $a \in \Sigma$,

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

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

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

Statements & Justifications?

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 3)

⇐ 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 ☑

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 $A_2$

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 $M_1$

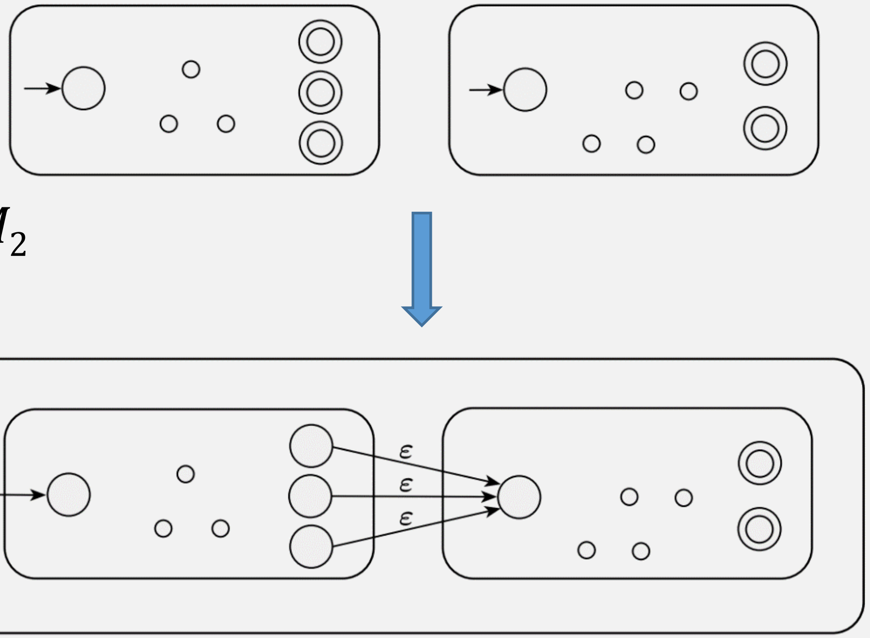
**3.** The accept states $F_2$ are the same as the accept states of $M_2$

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

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

And: $\delta(q, \varepsilon) = \emptyset$, for $q \in Q$, $q \notin F_1$ ~~???~~ ∎ ☑

$M_1$ $M_2$ $N$



# Concat Closed for Reg Langs: Use NFAs Only

**PROOF**

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

NFAs

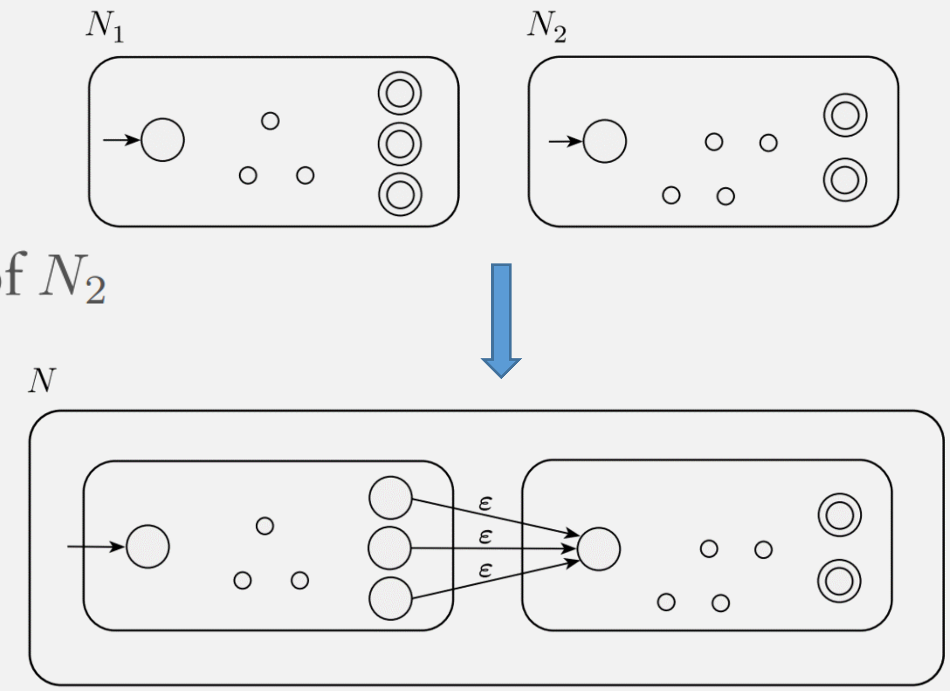
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

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 $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

????

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



**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?

# *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** $M_1$ **and** $M_2$
- 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 $Q_2$

State in $M$ = $M_1$ state + $M_2$ state

- $M$ transition fn: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$

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

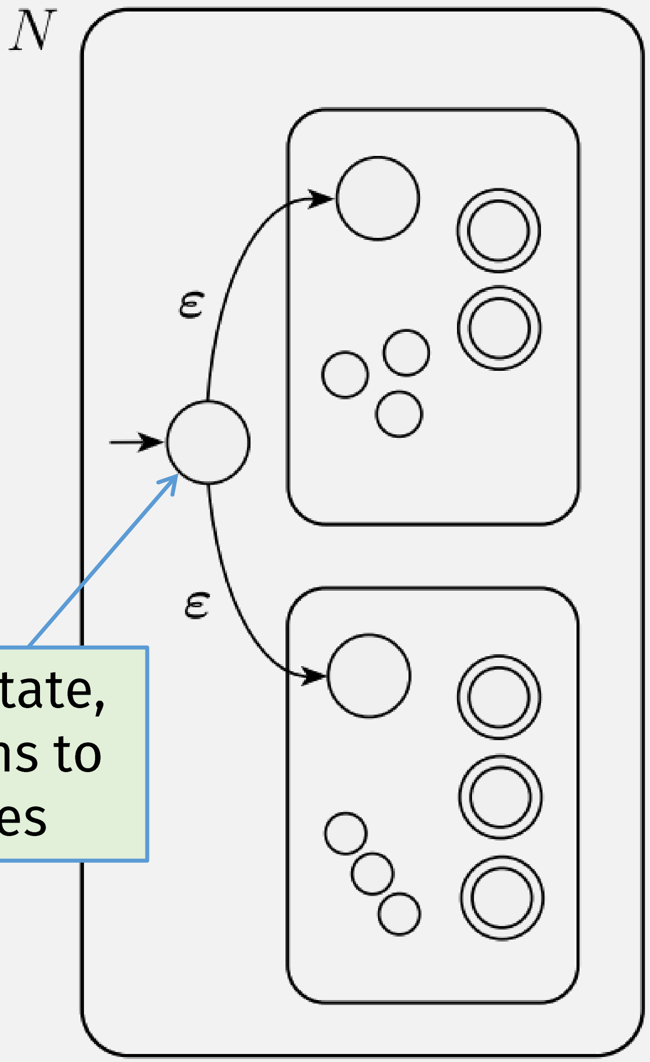
- $M$ start state: $(q_1, q_2)$

Accept if either $M_1$ or $M_2$ accept

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

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

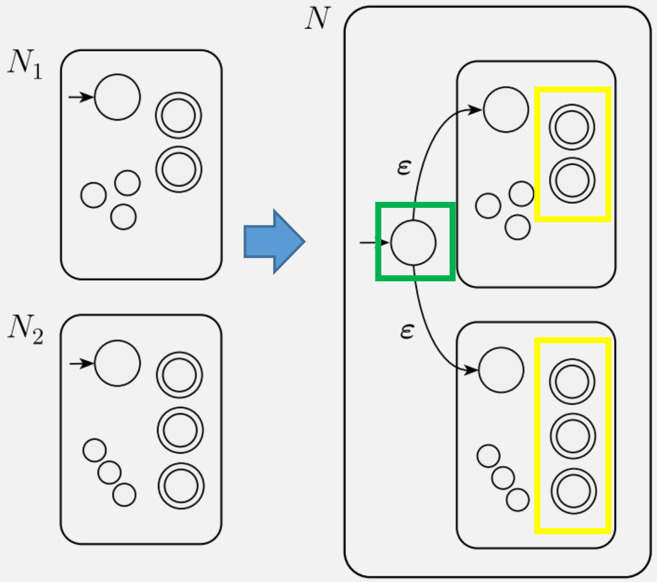
# Union is Closed for Regular Languages

**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 $N = (Q, \Sigma, \delta, q_0, F)$ 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 $F = F_1 \cup F_2$.



# 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 $A_1 \cup A_2$.

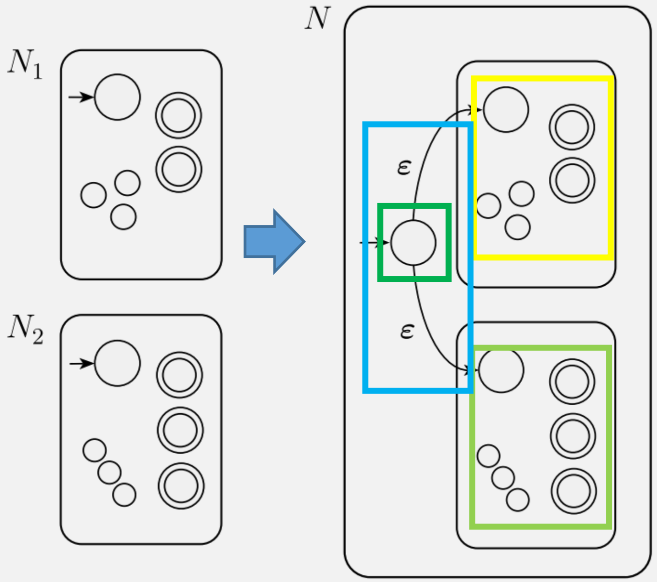
**1.** $Q = \{q_0\} \cup Q_1 \cup Q_2$.

**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 $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} ? & q \in Q_1 \\ ? & q \in Q_2 \\ ? & q = q_0 \text{ and } a = \varepsilon \\ ? & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$




Don't forget Statements and Justifications!

# List of Closed Ops for Reg Langs (so far)

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

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

# Kleene Star Example

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

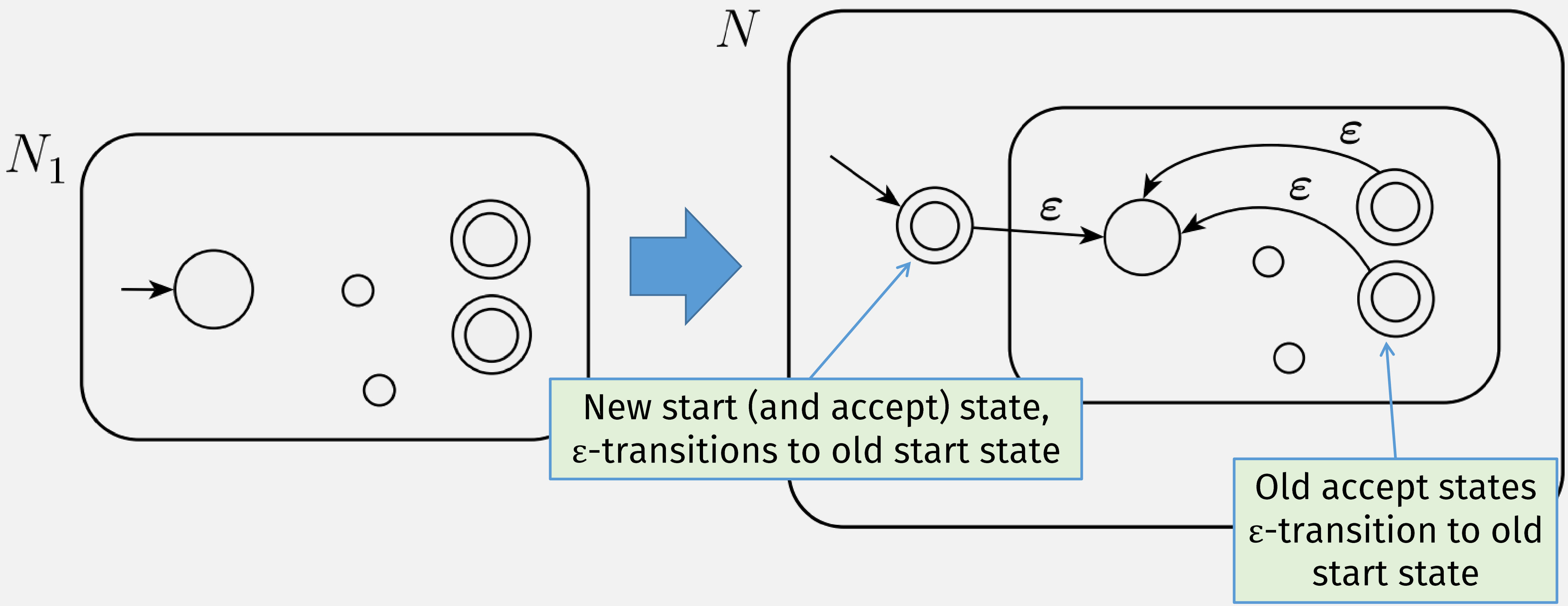
If $A = \{\mathtt{good}, \mathtt{bad}\}$

$$A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad}, \ldots\}$$

Note: repeat zero or more times

(this is an infinite language!)

# Kleene Star


$N_1$
$N$
ε
ε
ε
New start (and accept) state,
ε-transitions to old start state
Old accept states
ε-transition to old
start state

*In-class exercise:*

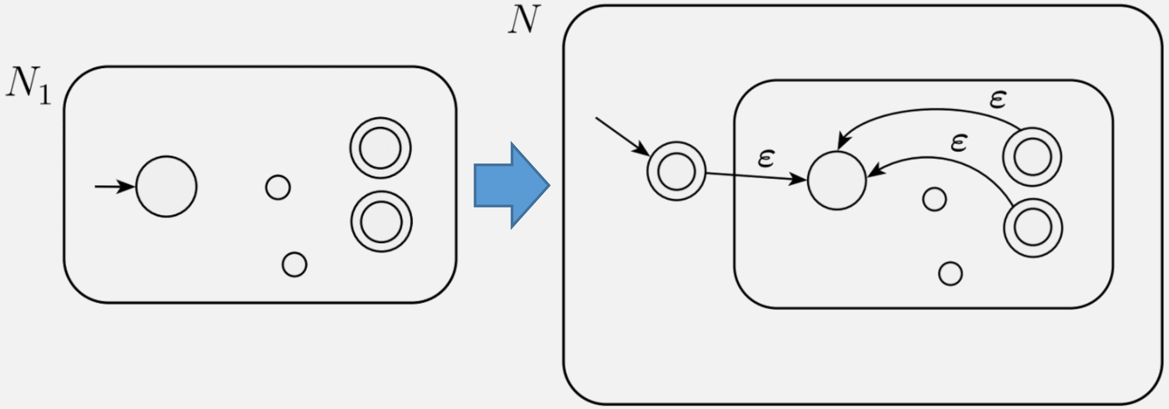
# Kleene Star is Closed for Regular Langs

**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 $A_1^*$.

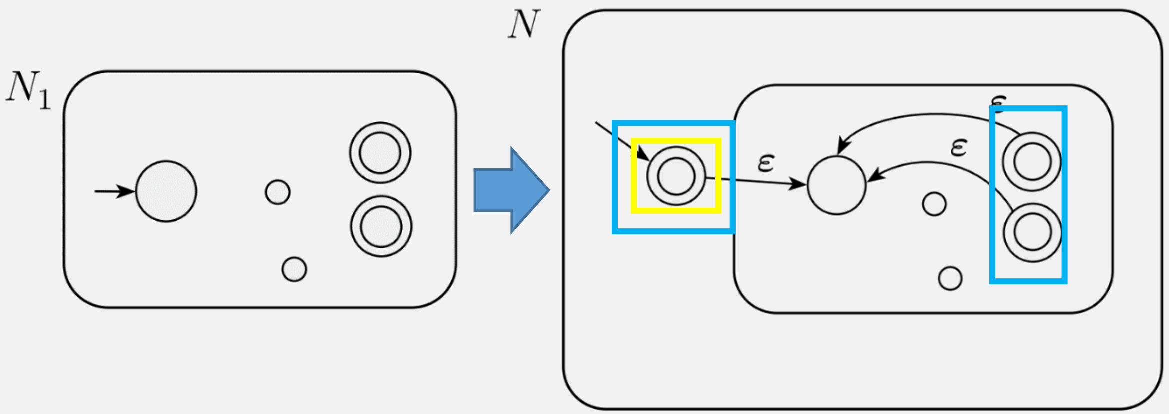


# Kleene Star is Closed for Regular Langs

**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 $A_1^*$.

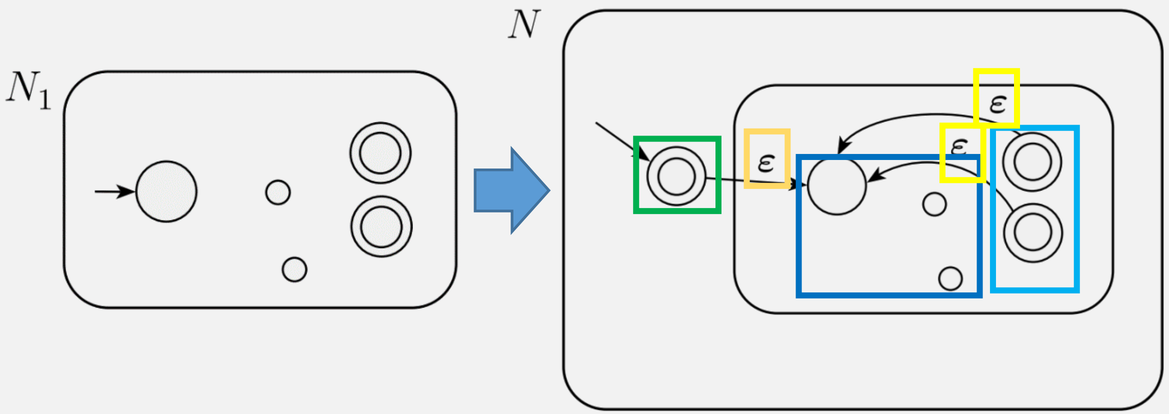
1. $Q = \{q_0\} \cup Q_1$
2. The state $q_0$ is the new start state.
3. $F = \{q_0\} \cup F_1$

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 $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

# *Next Time:* Why These Closed Operations?

- Union
- Concat
- Kleene star

<u>All</u> regular languages can be <u>constructed</u> from:

- single-char strings, and

- these three <u>combining</u> operations!

# Submit in-class work 2/26

On gradescope