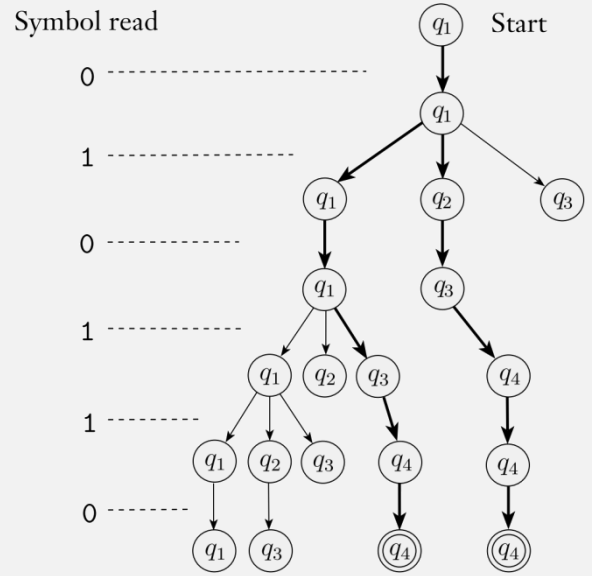
CS420

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?

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 $A =$ {**fort**, **south**} $B =$ {**point**, **boston**}

$A \circ B =$ {**fortpoint**, **fortboston**, **southpoint**, **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

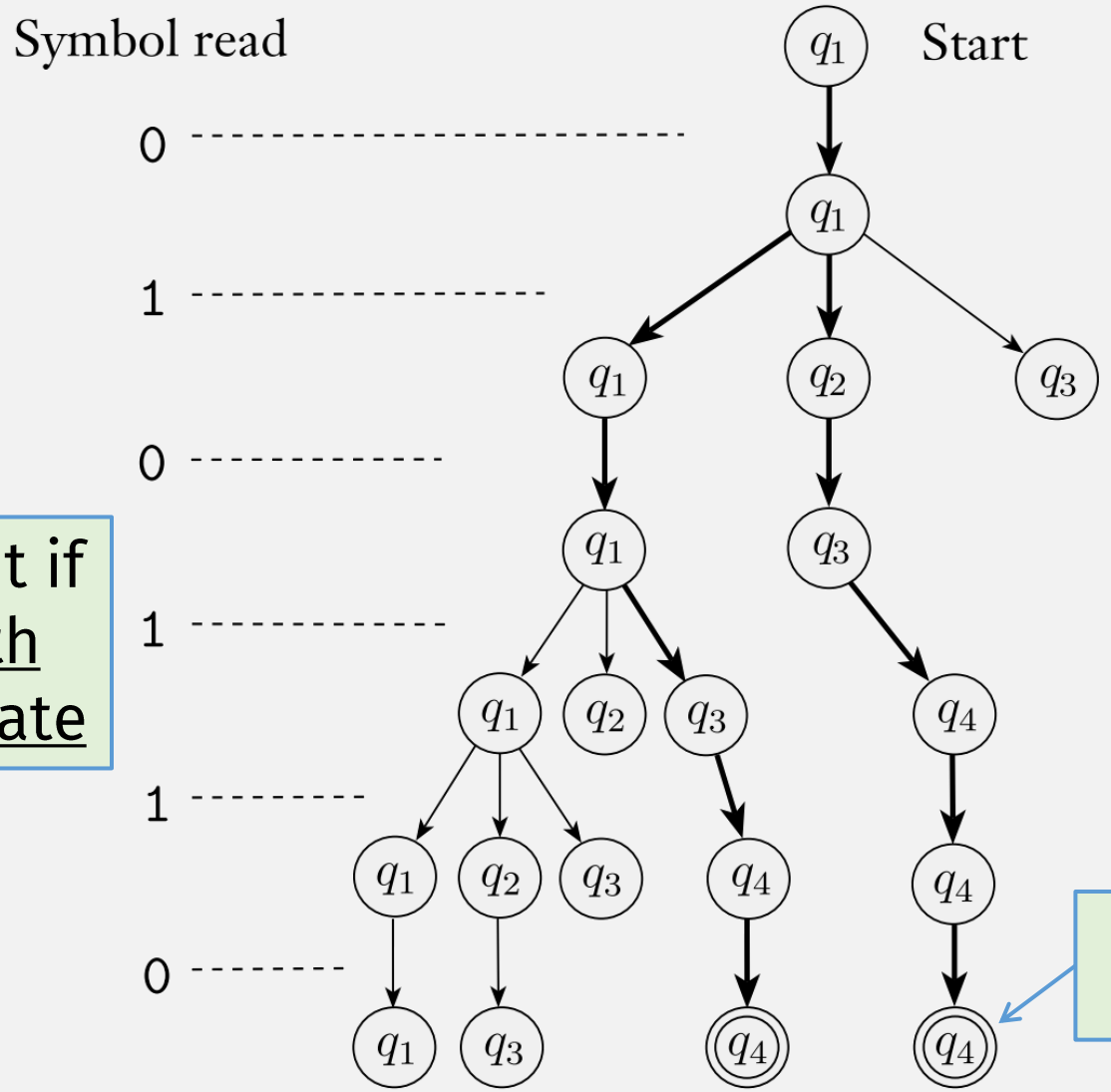
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ 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 may not read input, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Transition results in a set of states

Last Time: NFA Computation Sequence



NFA **accepts** input if at least <u>one path ends in accept state</u>

Each step can branch into <u>multiple states at the same time</u>!

So this is an **accepting computation**

Flashback: DFA Computation Model

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

- Machine = a DFA
- 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:

M ***accepts*** w if
sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with

- Start in "start state" → $r_0 = q_0$
- Repeat: → $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$
  - Read 1 char;
  - Change state according to the transition table
- Result =
  - Last state is "Accept" state → $r_n \in F$

NFA

Flashbacks: ~~DFA~~ Computation Model

Informally

- Machine = ~~a DFA~~ an NFA
- 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 $i = 1, \ldots, n$
  $r_i \in \delta(r_{i-1}, w_i)$

Next states 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. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ 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:** $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

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)

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

Empty string

nonEmpty string

First char

Remaining chars
("smaller argument")

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)

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

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

First chars ("smaller argument")

last char

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

Result is set of states

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

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

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

Result is set of states

(Defined recursively, on length of input string)

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

Empty string

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

nonEmpty string

Single transition steps for last char

Recursive call on first chars (smaller argument)

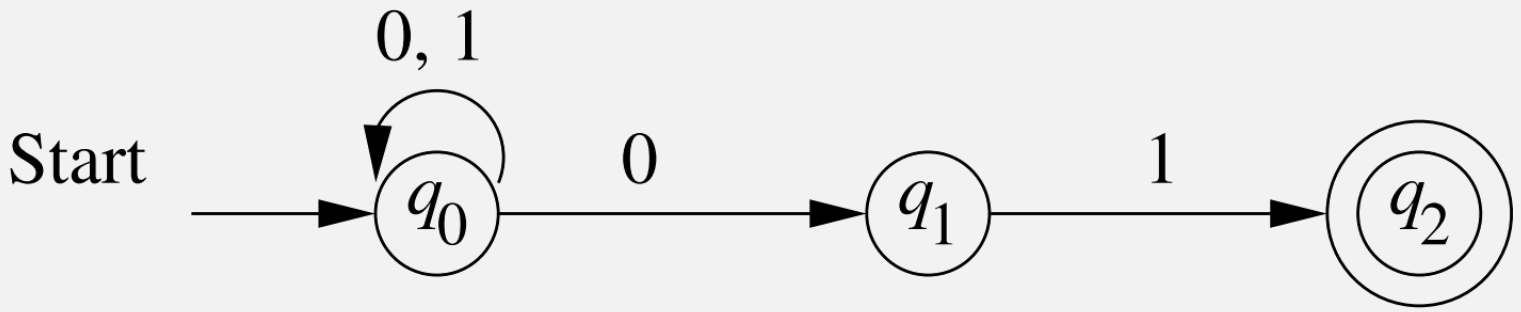
$\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$

NFA Extended δ Example

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

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

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



- $\hat{\delta}(q_0, \epsilon) = \{q_0\}$ Stay in start state
- $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$ Same as single step δ
- $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

Combine result of recursive call with “last step”

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

Adding Empty Transitions

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

(Defined recursively)

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

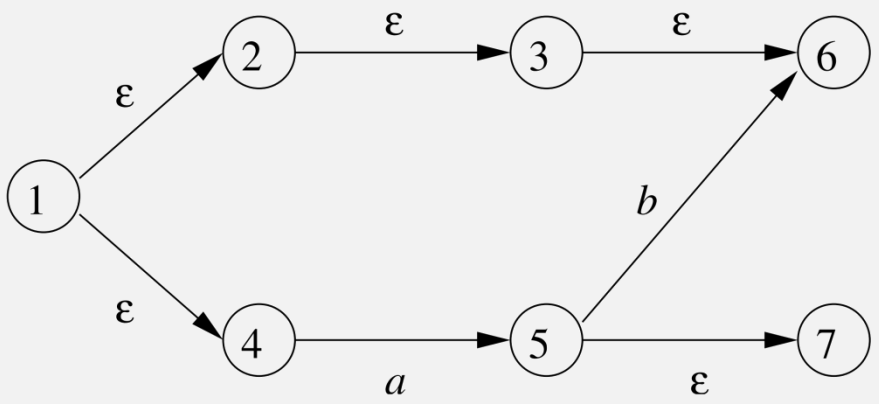
- Inductive case:

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

A state is in the reachable set if …

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

ε-REACHABLE Example



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

No empty transitions

NFA Extended Transition Function

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

Domain:

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

Range:

- Ending set of states

(Defined recursively, on length of input string)

- Base case: $\hat{\delta}(q, \epsilon) = \{q\}$
- 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:** $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$

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)

"Take single step, then follow all empty transitions"

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

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

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

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

Last Time: Concatenation is Closed?

THEOREM

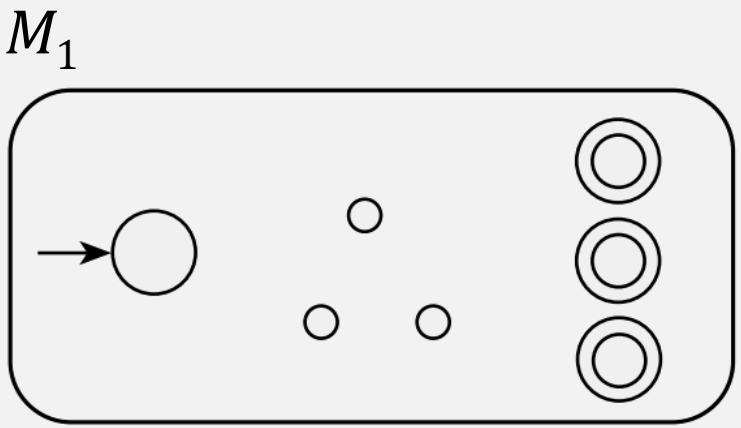
The class of regular languages is closed under the concatenation operation.

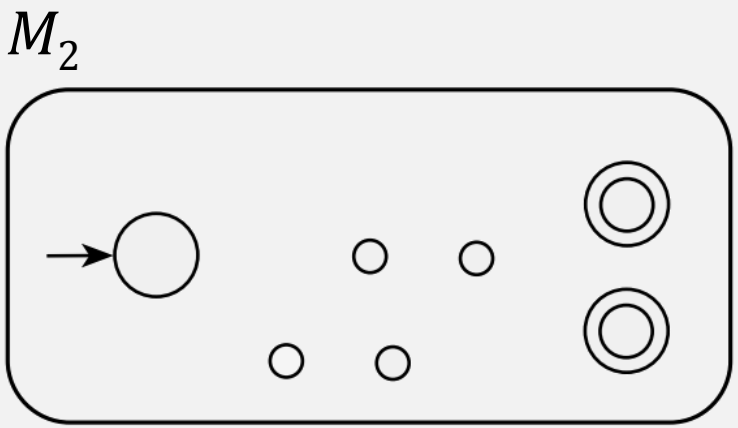
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

Proof: Construct a new machine

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

Concatentation





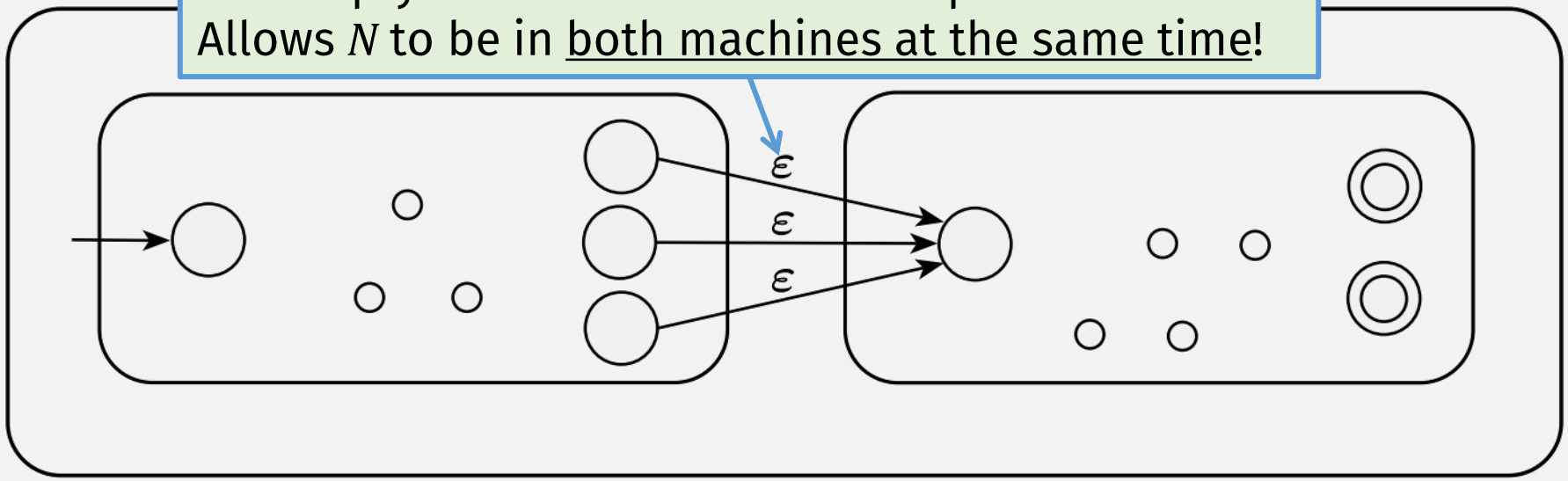
Let M_1 recognize A_1, and M_2 recognize A_2.

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

N is an **NFA**! Now is 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 <u>both machines at the same time</u>!



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 ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$	5. See examples
6. $A_1 \circ A_2$ ~~$A_1 \cup A_2$~~ is a regular language	6. ???
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

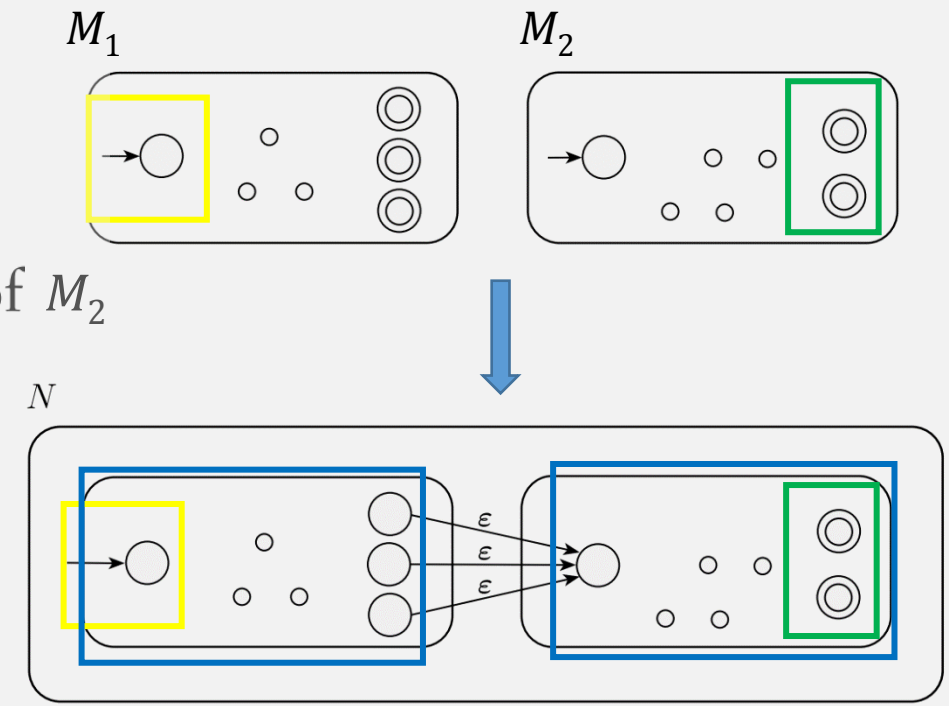
Concatenation is Closed for Regular Langs

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?

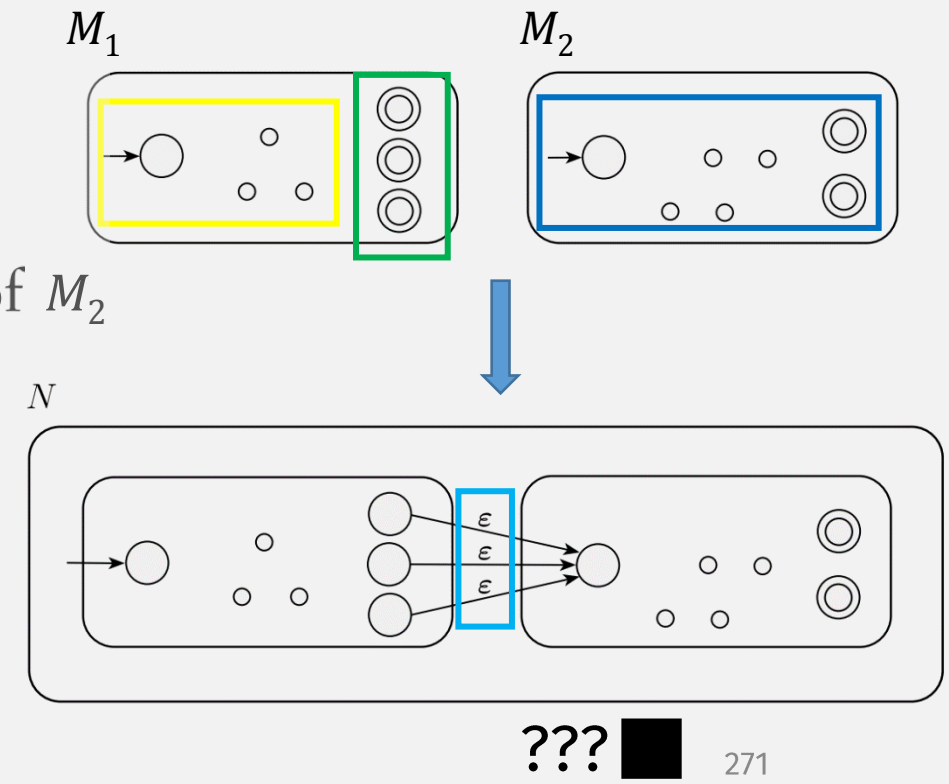
PROOF

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 $a \in \Sigma_\varepsilon$,

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



??? ■

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

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. $\Rightarrow$ 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 <u>at minimum</u> has 2 required 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

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"

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

Statements

1. L is a regular language
2. A DFA M recognizes L
3. Construct NFA N equiv to M
4. An NFA N recognizes L
5. If L is a regular language, then some NFA N recognizes it

Justifications

1. Assumption
2. Def of Regular language
3. See hw 2
4. ???
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)

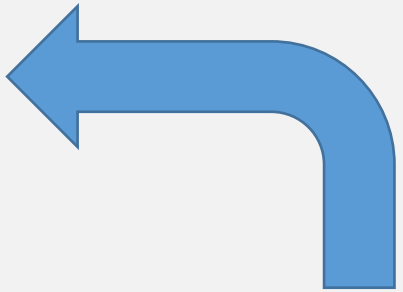
- We know: for L to be **regular**, there must be a **DFA** recognizing it
- Proof Idea for this part: Convert given NFA N → an equivalent DFA

"equivalent" =
"recognizes the same language"

How to convert **NFA→DFA**?

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

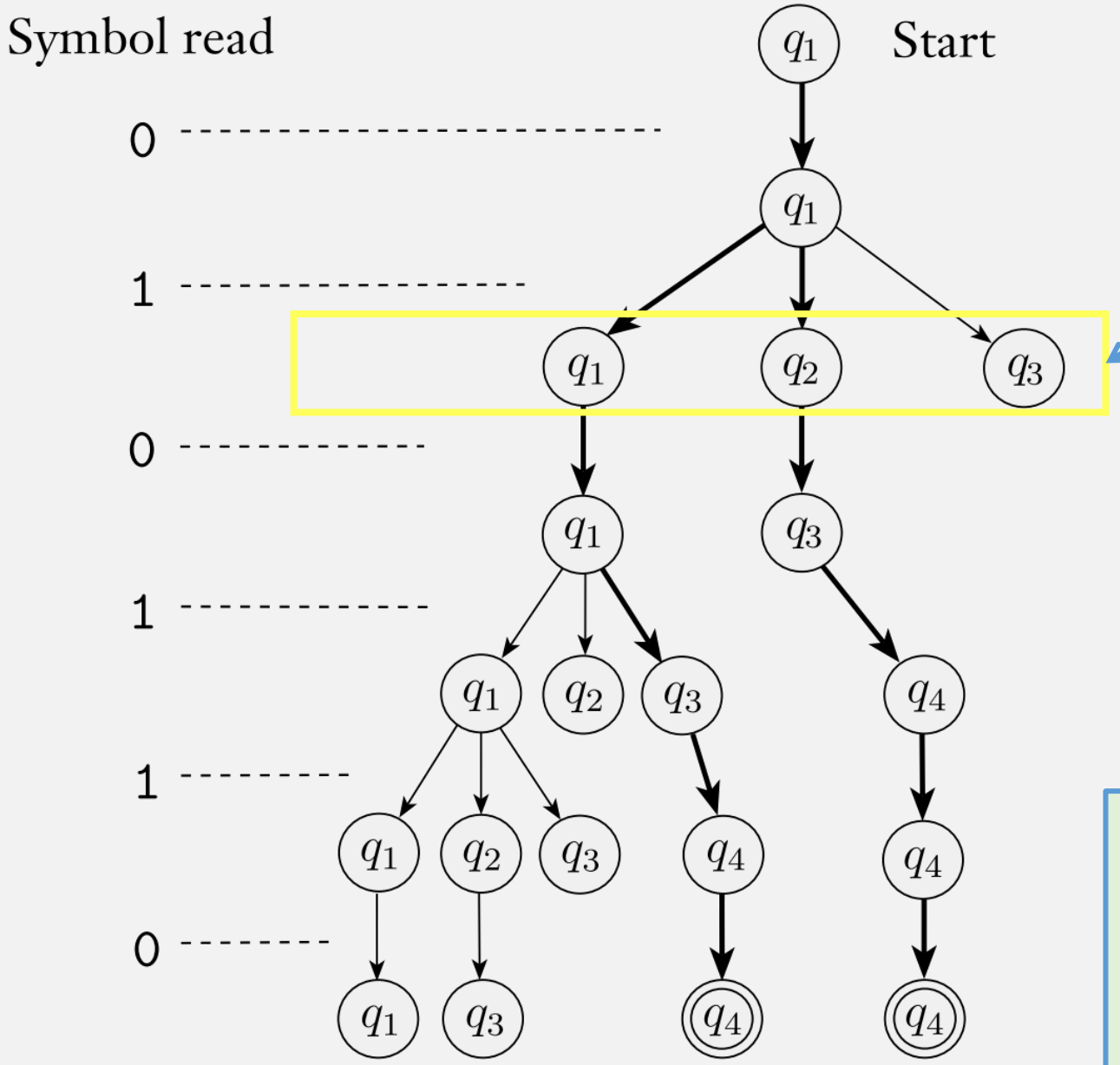
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ 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 <u>multiple</u> states

DFA computation can only be in <u>one</u> state

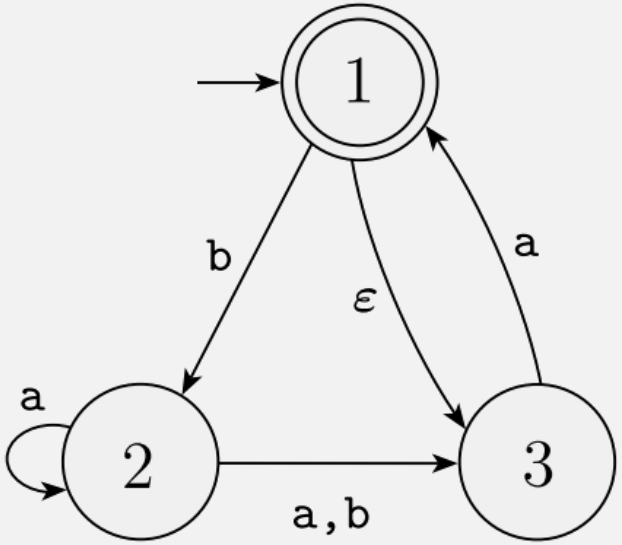
So **encode:**
a <u>set of NFA states</u>
as <u>one DFA state</u>

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

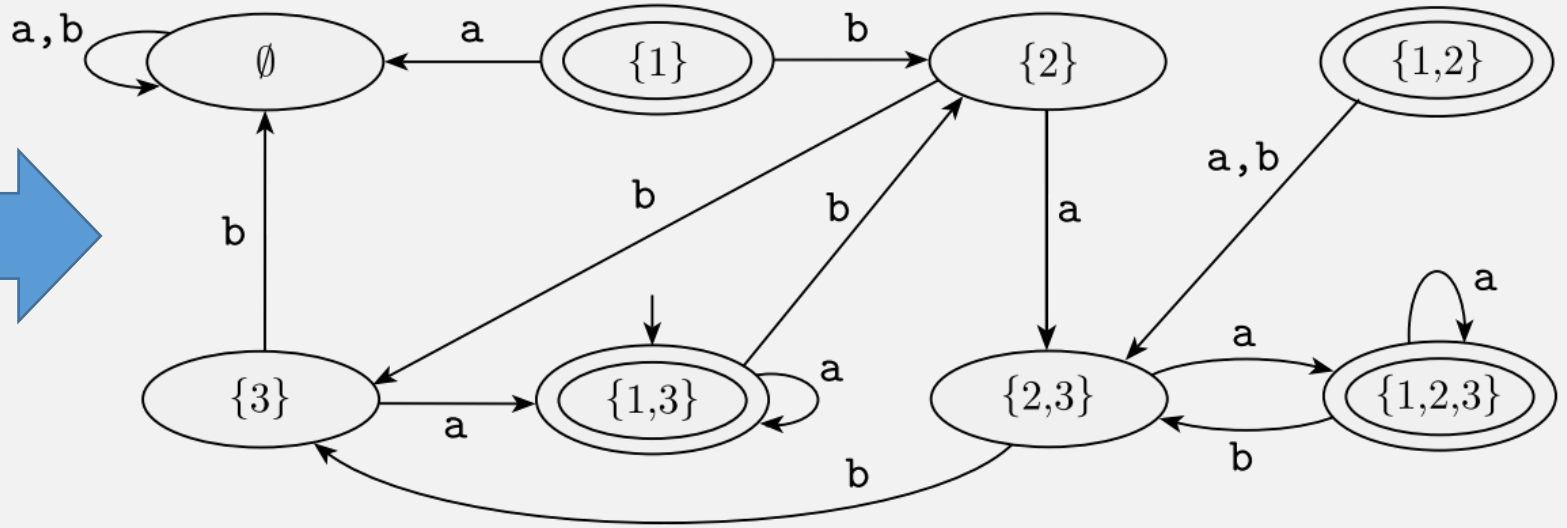
Convert **NFA→DFA**, Formally

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

- Inductive case:

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

A state is in the reachable set if …

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

With empty transitions

NFA→DFA

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

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

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

Almost the same, except …

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

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

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

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

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 $A_1 \circ A_2$

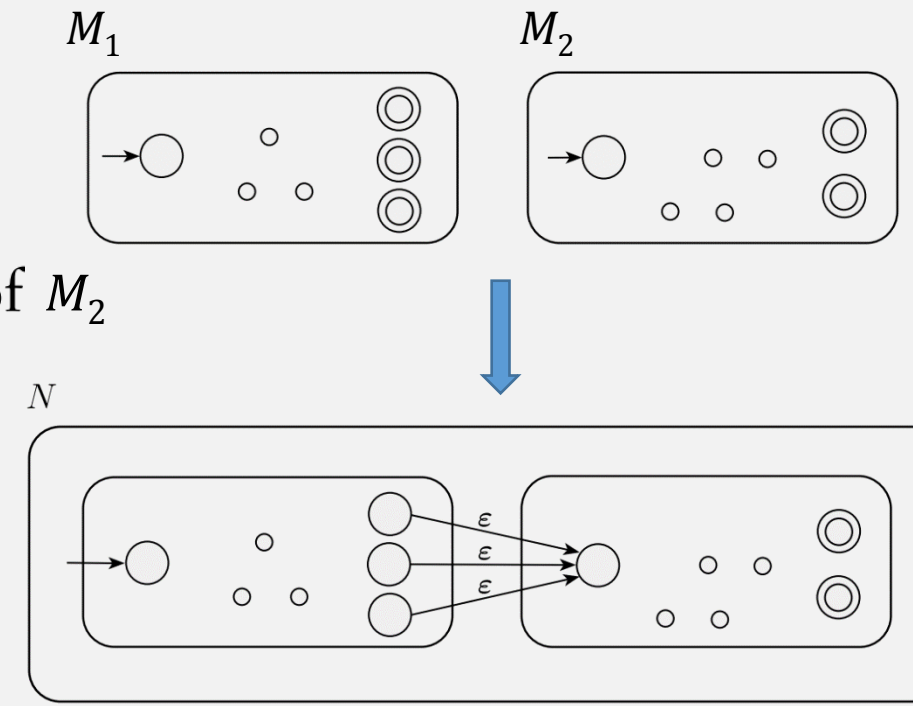
1. $Q = Q_1 \cup Q_2$

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 $a \in \Sigma_\varepsilon$,

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



☑ ??? ■

Concat Closed for Reg Langs: Use NFAs Only

PROOF

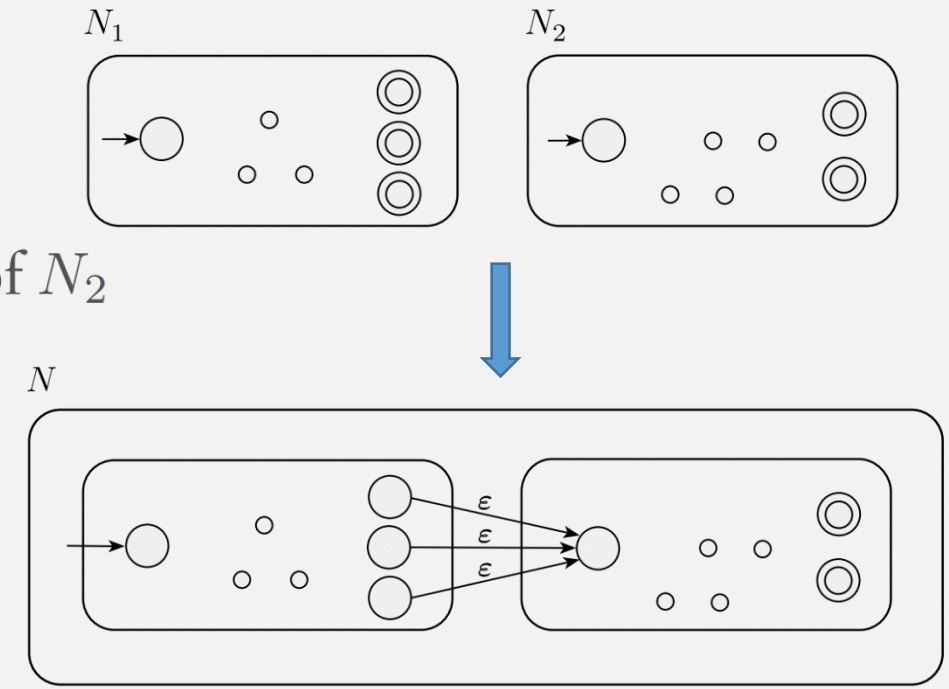
If language is regular,
then it has an **NFA** recognizing it ...

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: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

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

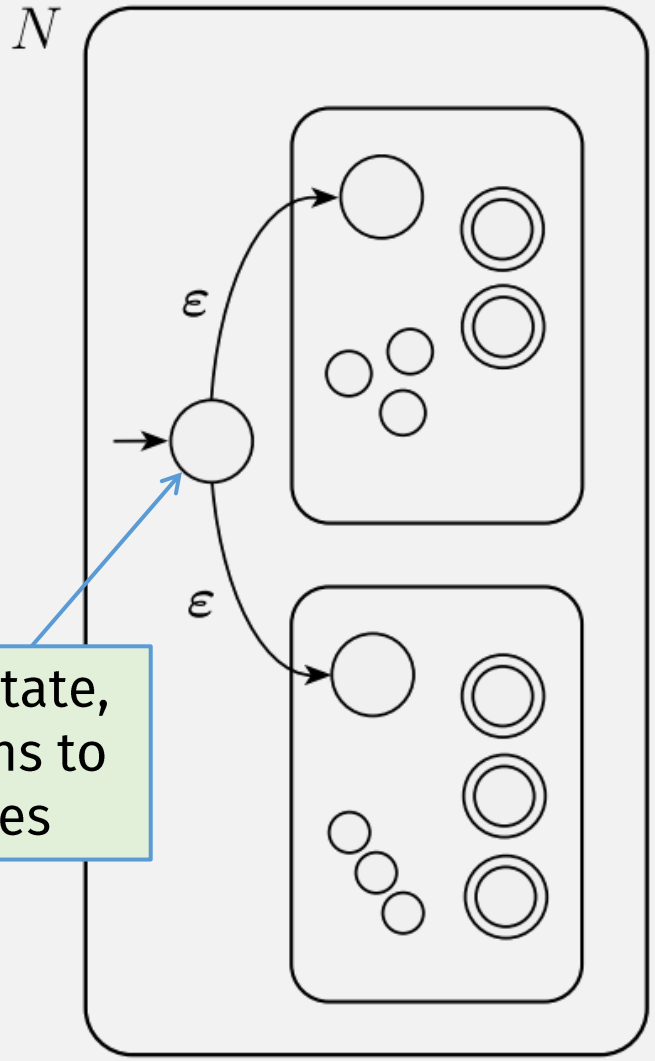
- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using M_1 and M_2
- states of M: $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$
  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: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

  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: $F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Union is Closed for Regular Languages



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

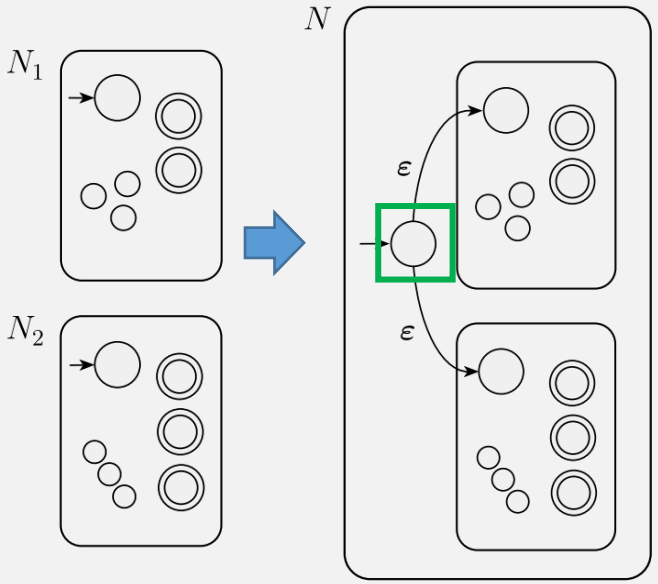
Union is Closed for Regular Languages

PROOF

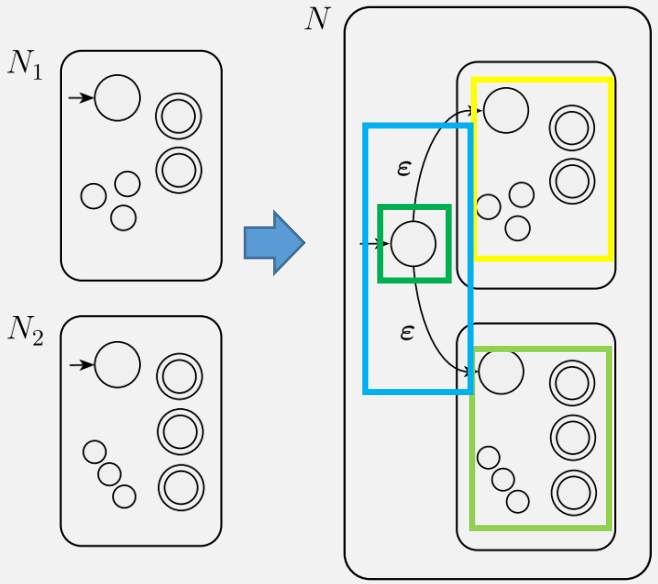
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

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

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

List of Closed Ops for Reg Langs (so far)

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

Star: $A^* = \{x_1 x_2 \ldots x_k | \; k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star Example

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

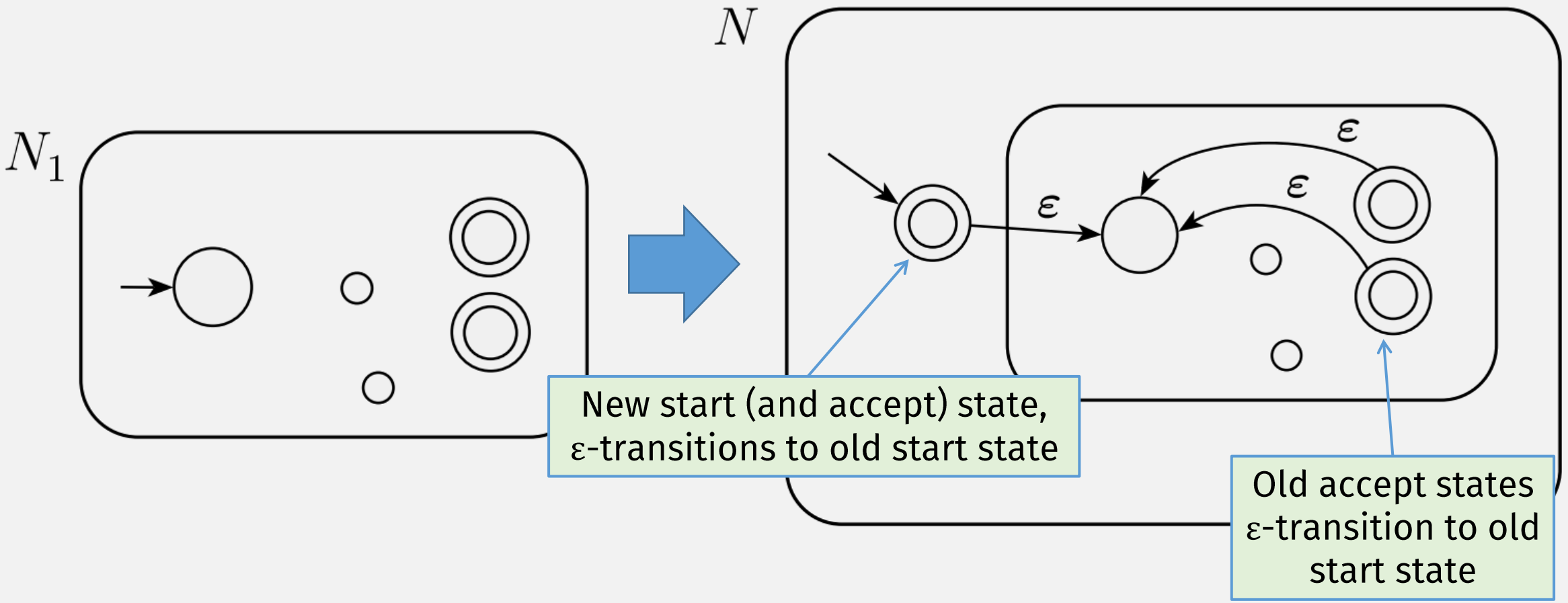
If $A = \{\mathtt{good}, \mathtt{bad}\}$

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

Note: repeat zero or more times

(this is an infinite language!)

Kleene Star


N
N_1
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:

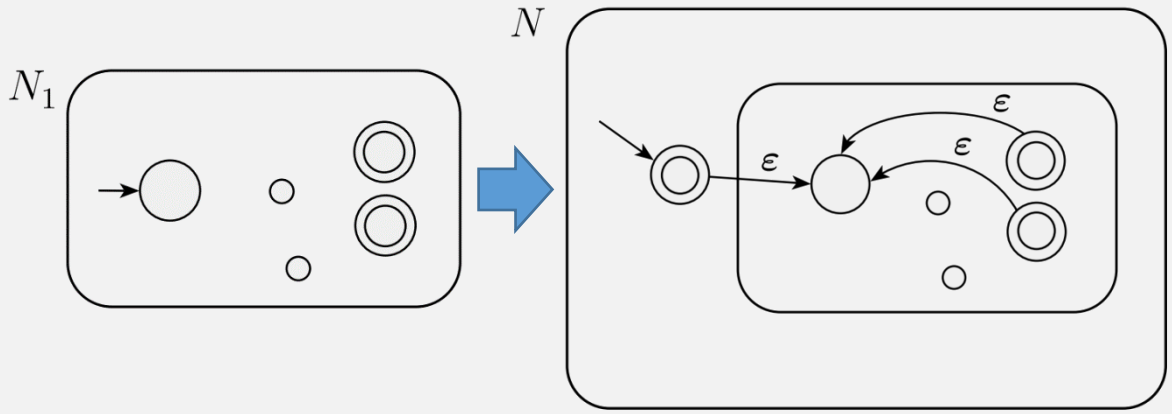
Kleene Star is Closed for Regular Langs

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^*.

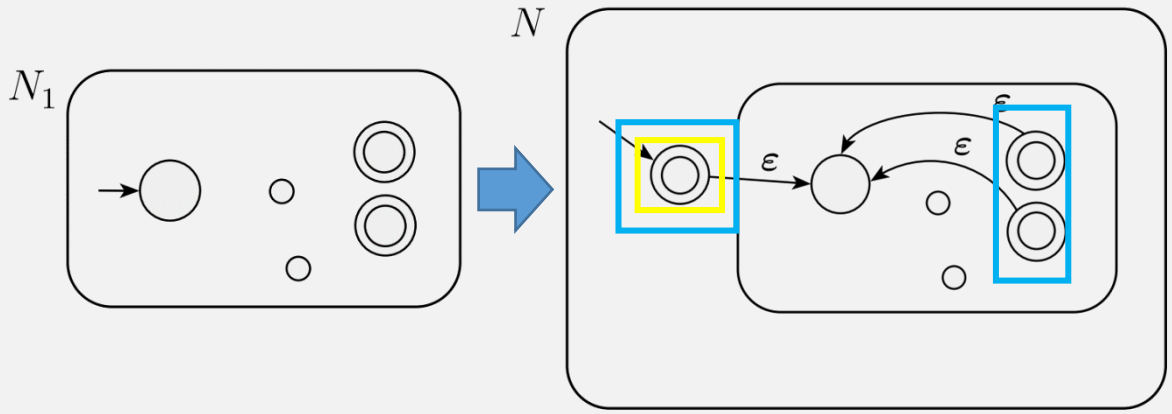


Kleene Star is Closed for Regular Langs

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^*.

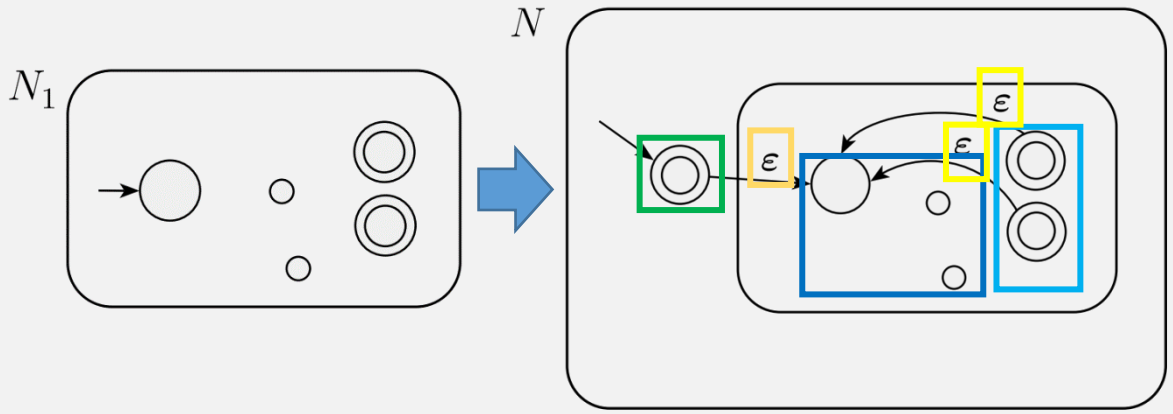
1. $Q = \{q_0\} \cup Q_1$
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$

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

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

Many More Closed Operations on Regular Languages!

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

Check-in Quiz 2/13

On gradescope