CS 420 / CS 620

NFA ⇔ DFA

Wednesday, October 1, 2025

UMass Boston Computer Science

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.

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***.

Announcements

- HW 4
  - Out: **Mon 9/29 12pm (noon)**
  - Due: **Mon 10/6 12pm (noon)**

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.

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***.

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.

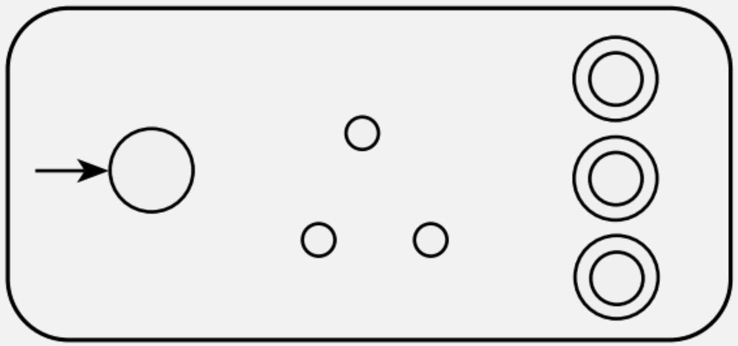
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

Proof requires: Constructing new machine

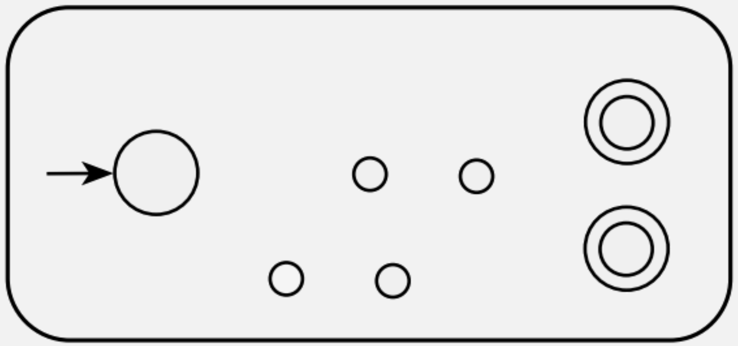
Key step: **When to switch machines?** (can only read input once)

Concatentation

M_1



M_2



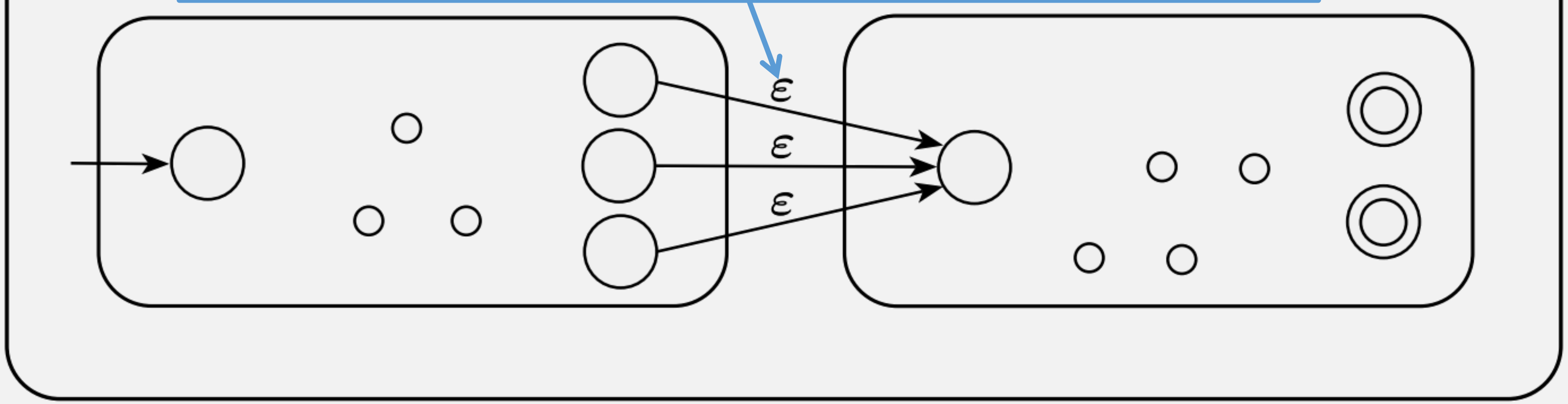
Let M_1 recognize A_1, and M_2 recognize A_2.

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
N can be in both machines simultaneously!



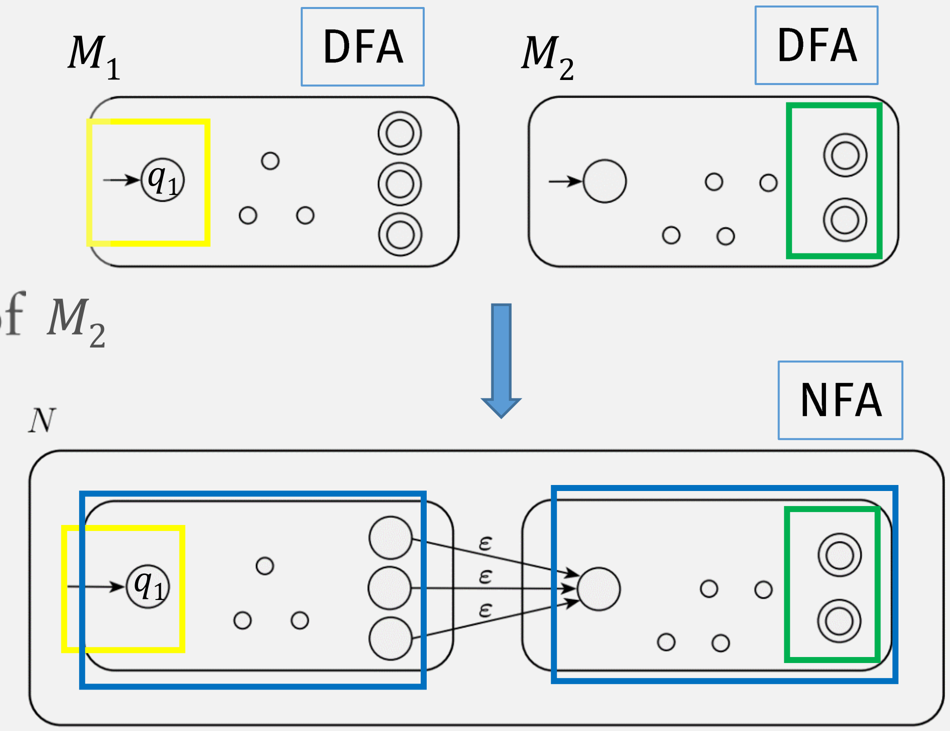
Concatenation is Closed for Regular Langs

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 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 (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

Define the function:

$\text{CONCAT}_{\text{DFA-NFA}}(M_1, M_2) = N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

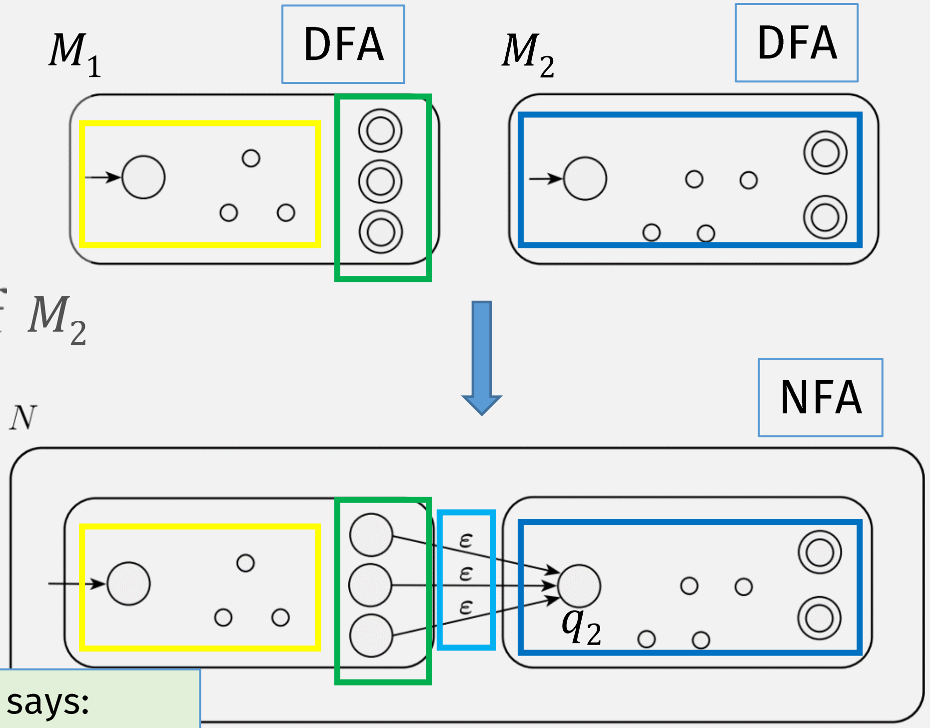
1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of M_1
3. The accept states F_2 are the same as the accept states of M_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

(no other empty transitions)

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

NFA def says:
δ must map every state and ε to set of states



??? ∎

Previously

Is Union Closed For Regular Langs?

Proof

Statements	**Justifications**
1. A_1 and A_2 are regular languages	1. Assumption of If part of If-Then
2. A DFA M_1 recognizes A_1	2. Def of Reg Lang (Coro)
3. A DFA M_2 recognizes A_2	3. Def of Reg Lang (Coro)
4. Construct DFA $M =$ $\text{UNION}_{\text{DFA}}$ (M_1,M_2)	4. Def of DFA and $\text{UNION}_{\text{DFA}}$
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 $\boxed{\text{Concat}}$ Closed For Regular Langs?

Proof?

Statements	Justifications
1. A_1 and A_2 are regular languages	1. Assumption of If part of If-Then
2. A DFA M_1 recognizes A_1	2. Def of Reg Lang (Coro)
3. A DFA M_2 recognizes A_2	3. Def of Reg Lang (Coro)
4. Construct NFA $N = \text{CONCAT}_{\text{DFA-NFA}}(M_1, M_2)$ ☑	4. Def of NFA and $\text{CONCAT}_{\text{DFA-NFA}}$
5. N recognizes $\cancel{A_1 \cup A_2}$ $A_1 \circ A_2$	5. See Examples Table
6. $\cancel{A_1 \cup A_2}$ $A_1 \circ A_2$ is a regular language	6. ~~Def~~ ??? Does NFA recognize reg langs?
7. The class of regular languages is closed under 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

Definition of Regular Language

If a **DFA recognizes** a language L, then L is a **regular 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 … … with accept states …

- N ***accepts*** w if $\hat{\delta}(q_0, w) \cap F \neq \emptyset$ … is not empty set
  - i.e., accept if final states contains <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 <u>NFA</u>

- So to **prove regular languages closed under concatenation …**

  … must **prove** that <u>NFAs *also* recognize regular languages</u>.

Specifically, we will <u>prove</u>:

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 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

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof: 2 parts

Assume

Prove

$\Rightarrow$ 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 $\rightarrow$ an equivalent NFA! (see HW 4)

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

Full Statements & Justifications?

"equivalent" = "recognizes the same language"

$\Rightarrow$ 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 $N = \text{CONVERT}_{\text{DFA-NFA}}(M)$	3. See hw 4!
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 $N = \text{CONVERT}_{\text{DFA-NFA}}(M)$

$\hat{\delta}(q_0, w) \in F$ for some string w

Note:
extra 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 … r_n, where $r_i \in Q$ and

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

Exercise left for HW
Show that you know how an NFA computes

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 $N = \text{CONVERT}_{\text{DFA-NFA}}(M)$

$\hat{\delta}(q_0, w) \in F$ for some string w

$\hat{\delta}(q_0, w') \notin F$ for some string w'

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

If M rejects w' …

Then we know …

Exercise left for HW

Show that you know how an NFA computes

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 p Assume vert that DFA → Prove uivalent NFA! (see HW 4)

⇐ 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**?

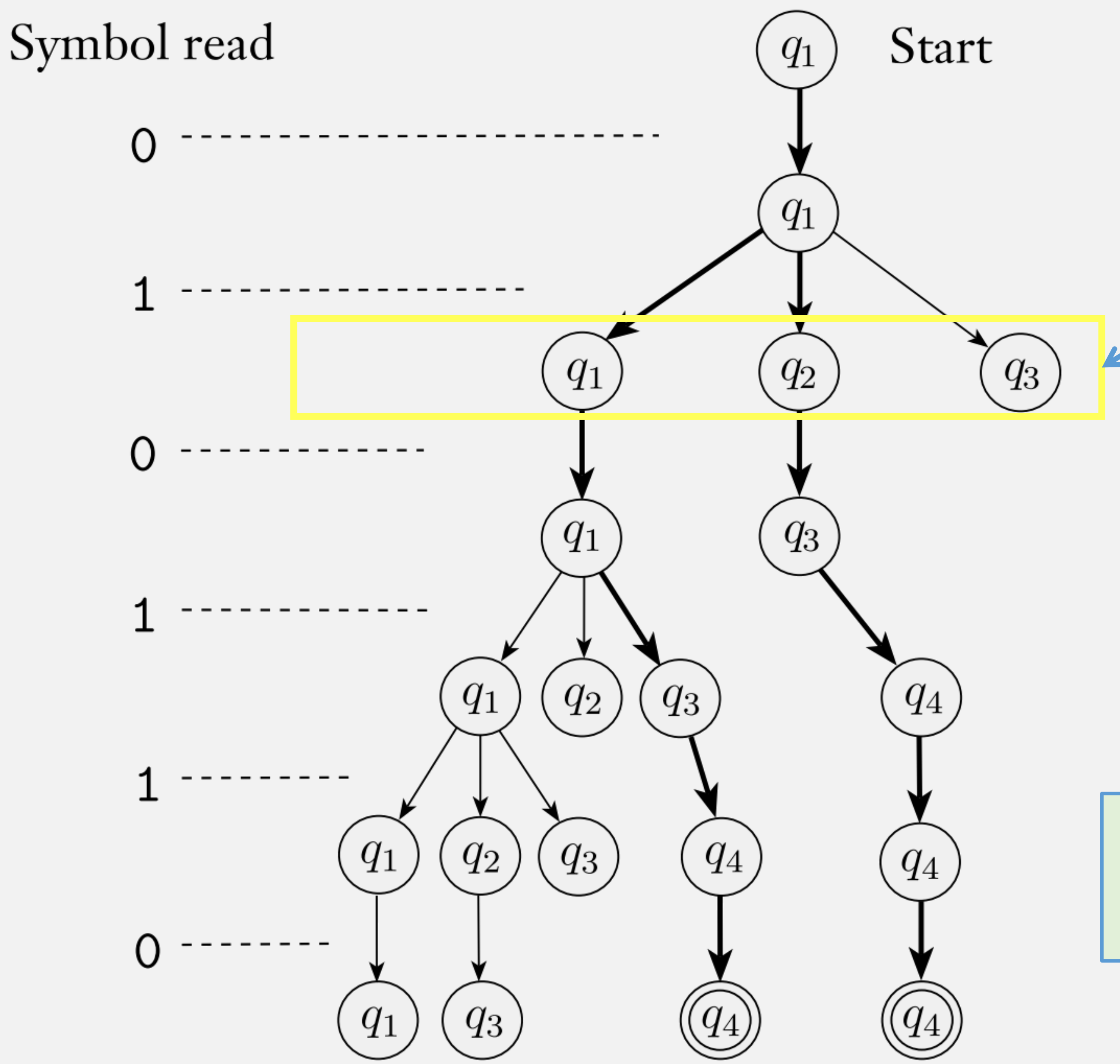
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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**

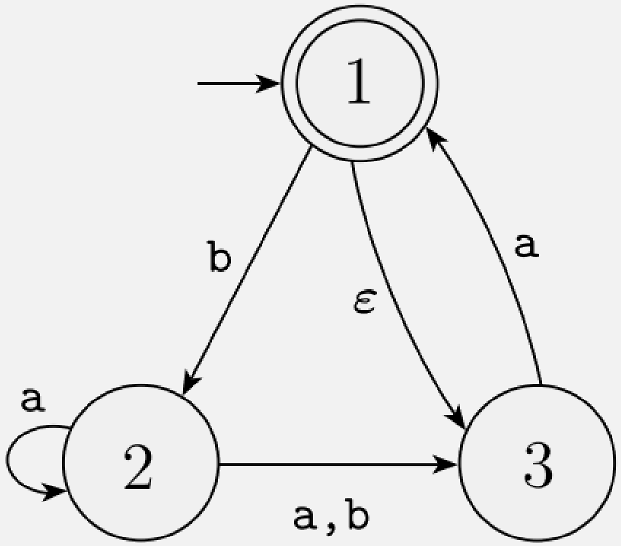
This is similar to the proof strategy from **"Closure of union"** where:
a state = a pair of states

Convert **NFA→DFA**, Formally

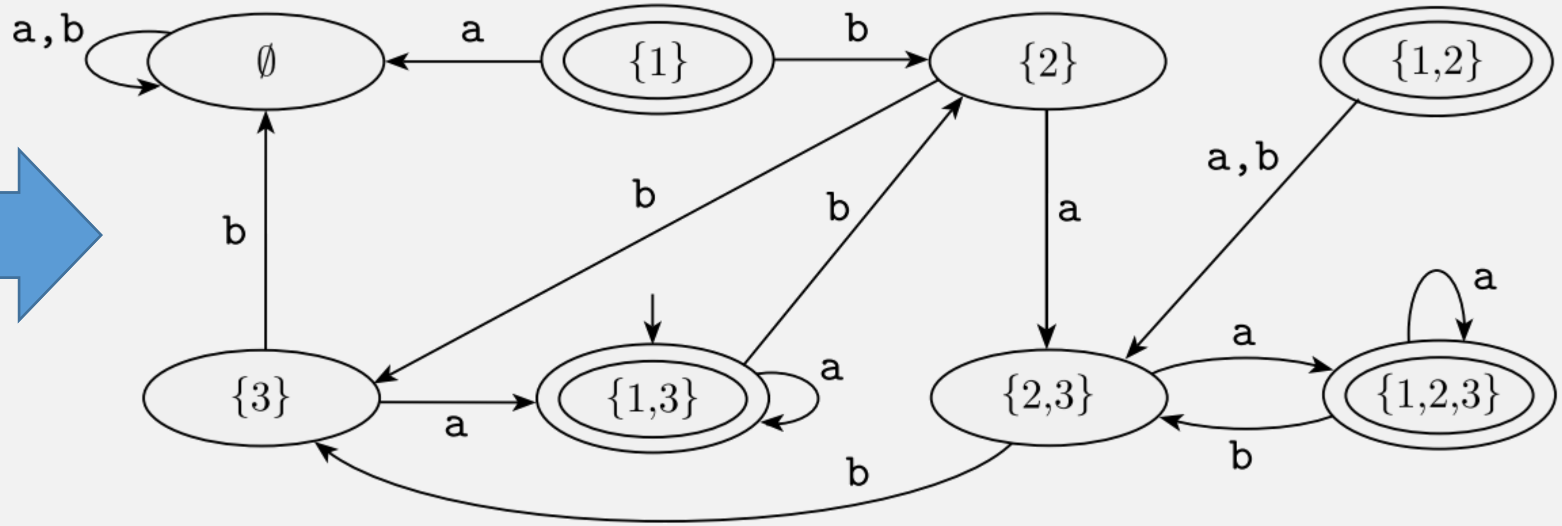
- Let **NFA** N = $(Q, \Sigma, \delta, q_0, F)$
- **An equivalent DFA** M **has states** Q' = $\mathcal{P}(Q)$ (power set of Q)

Example:

- Let **NFA** N_4 = $(Q, \Sigma, \delta, q_0, F)$
- An equivalent DFA D has states = $\mathcal{P}(Q)$ (power set of Q)



The NFA N_4



A DFA D that is equivalent to the NFA N_4

No empty transitions

NFA→DFA

Have: NFA $N = (Q_{\text{NFA}}, \Sigma, \delta_{\text{NFA}}, q_{0\text{NFA}}, F_{\text{NFA}})$

Want: DFA $D = (Q_{\text{DFA}}, \Sigma, \delta_{\text{DFA}}, q_{0\text{DFA}}, F_{\text{DFA}})$

1. $Q_{\text{DFA}} = \mathcal{P}(Q_{\text{NFA}})$

A DFA state = a set of NFA states

qs = DFA state = set of NFA states

2. For $qs \in Q_{\text{DFA}}$ and $a \in \Sigma$
   - $\delta_{\text{DFA}}(qs, a) = \bigcup_{q \in qs} \delta_{\text{NFA}}(q, a)$

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

3. $q_{0\text{DFA}} = \{q_{0\text{NFA}}\}$

4. $F_{\text{DFA}} = \{\, qs \in Q_{\text{DFA}} \mid qs$ contains 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:

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

Have: NFA $N = (Q_{\text{NFA}}, \Sigma, \delta_{\text{NFA}}, q_{0\text{NFA}}, F_{\text{NFA}})$

Want: DFA $D = (Q_{\text{DFA}}, \Sigma, \delta_{\text{DFA}}, q_{0\text{DFA}}, F_{\text{DFA}})$

Almost the same, except ...

1. $Q_{\text{DFA}} = \mathcal{P}(Q_{\text{NFA}})$

2. For $qs \in Q_{\text{DFA}}$ and $a \in \Sigma$
   - $\delta_{\text{DFA}}(qs, a) =$ $S = \cancel{\bigcup_{q \in qs} \delta_{\text{NFA}}(q, a)}$

   $$\bigcup_{s \in S} \varepsilon\text{-REACHABLE}(s)$$

3. $q_{0\text{DFA}} = \cancel{\{q_{0\text{NFA}}\}}\ \varepsilon\text{-REACHABLE}(q_{0\text{NFA}})$

4. $F_{\text{DFA}} = \{\, qs \in Q_{\text{DFA}} \mid qs$ contains 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 4)

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

Examples table?

(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

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

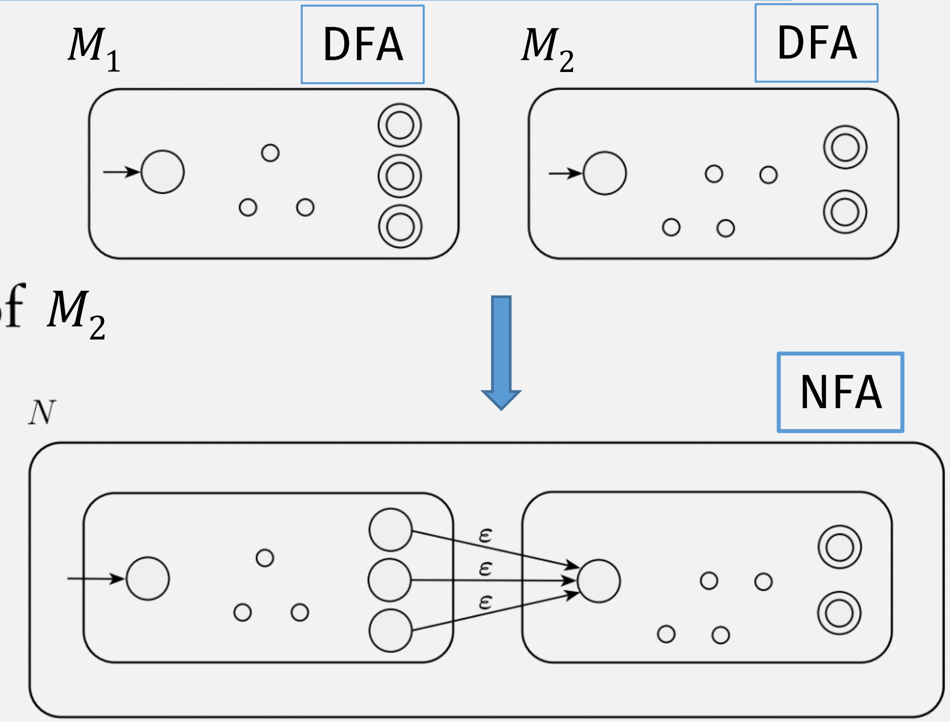
Let DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

CONCAT$_{\text{DFA-NFA}}$ $(M_1, M_2) = N = (Q, \Sigma, \delta, q_1, F_2)$ 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 $a \in \Sigma_\varepsilon$,

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

<u>And</u>: $\delta(q, \varepsilon) = \emptyset$, for $q \in Q, q \notin F_1$ ~~???~~ ■ ☑



Is Concat Closed For Regular Langs?

Proof?

Statements

1. A_1 and A_2 are regular languages
2. A DFA M_1 recognizes A_1
3. A DFA M_2 recognizes A_2
4. Construct NFA $N = \text{CONCAT}_{\text{DFA-NFA}}\ (M_1, M_2)$
5. N recognizes $A_1 \circ A_2$
6. $A_1 \circ A_2$ is a regular language
7. The class of regular languages is closed under concatenation operation.

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

Justifications

1. Assumption of If part of If-Then
2. Def of Reg Lang (Coro)
3. Def of Reg Lang (Coro)
4. Def of NFA and $\text{CONCAT}_{\text{DFA-NFA}}$
5. See Examples Table
6. Thm1.40⇐ If NFA recognizes lang, then it's Regular
7. From stmt #1 and #6

Q.E.D.? ■

New possible proof strategy!

Use NFAs Only

Is Concat Closed For Regular Langs?

Proof?

Statements	Justifications
1. A_1 and A_2 are regular languages	1. Assumption of If part of If-Then
2. A NFA N_1 recognizes A_1	2. If a lang is Regular, then it has an NFA
3. A NFA N_2 recognizes A_2	3. If a lang is Regular, then it has an NFA
4. Construct NFA $N = \text{CONCAT}_{\text{NFA}}{}^{???} (N_1, N_2)$	4. Def of NFA and $\text{CONCAT}_{\text{NFA}}$
5. N recognizes $A_1 \circ A_2$	5. See Examples Table
6. $A_1 \circ A_2$ is a regular language	6. If NFA recognizes lang, then it's Regular
7. The class of regular languages is closed under 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

Thm1.40⇒

New possible proof strategy!

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

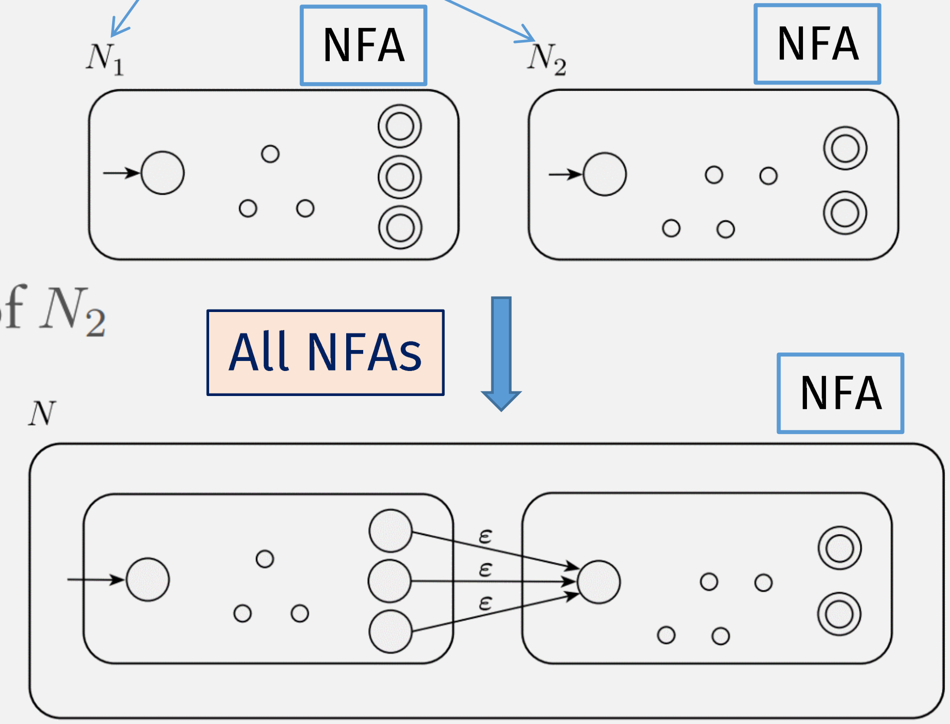
NFAs $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

CONCAT$_{\text{NFA}}$ $(N_1, N_2) = N =$ $(Q, \Sigma, \delta, q_1, F_2)$ 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 $a \in \Sigma_\varepsilon$,

????

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



New possible proof strategy!

Sipser Thm 1.47

Concat Closed for Reg Langs: Use NFAs Only

PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and

NFAs $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

CONCAT$_{\text{NFA}}$ $(N_1, N_2) = N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$

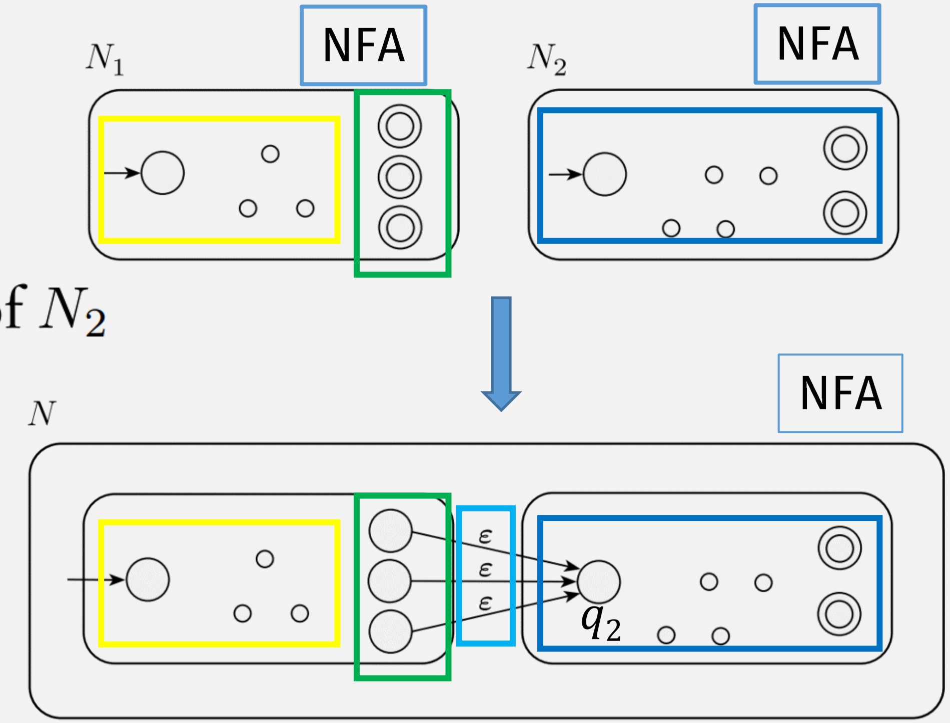
2. The state q_1 is the same as the start state of N_1

3. The accept states F_2 are the same as the accept states of N_2

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

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

F_1 states might already have empty transitions!



■

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: $\text{UNION}_{\text{DFA}}(M_1, M_2) = M = (Q, \Sigma, \delta, q_0, F)$ 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

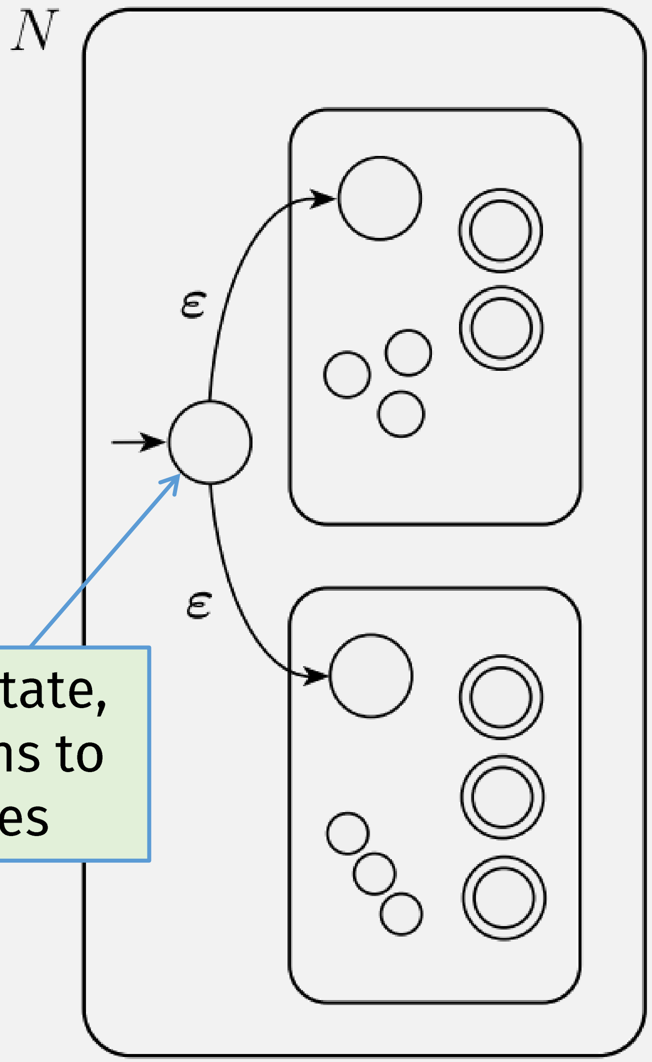
- M start state: (q_1, q_2)

Accept if <u>either</u> 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

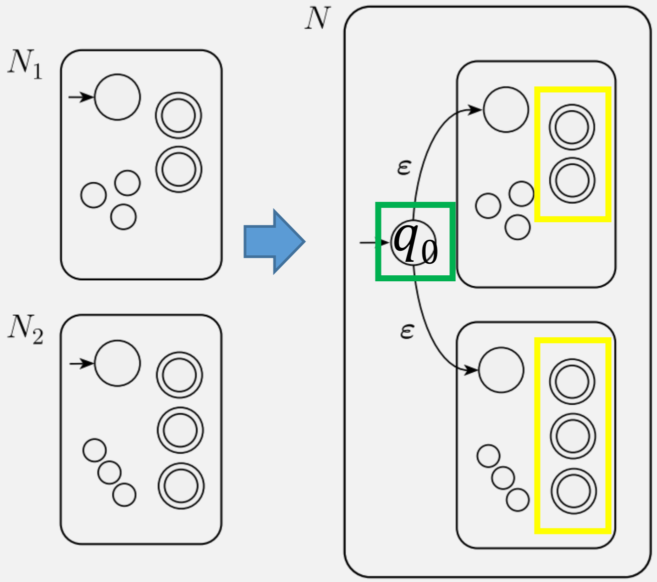
Union is Closed for Regular Languages

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.

UNION$_{\text{NFA}}$ $(N_1,N_2) = N =$ $(Q, \Sigma, \delta, \boxed{q_0,} F)$ 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 $\boxed{F} = F_1 \cup F_2$.



Sipser Thm 1.45

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.

UNION$_{\text{NFA}}$ (N_1,N_2) = N = $(Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

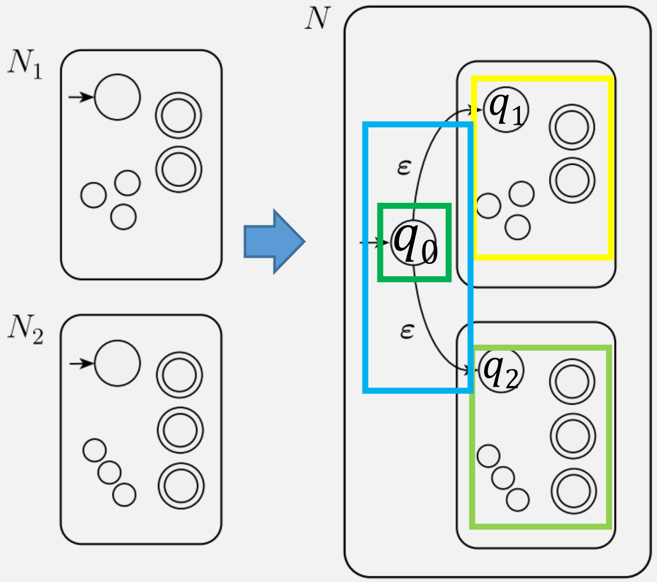
1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.

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 $a \in \Sigma_\varepsilon$,

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



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

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

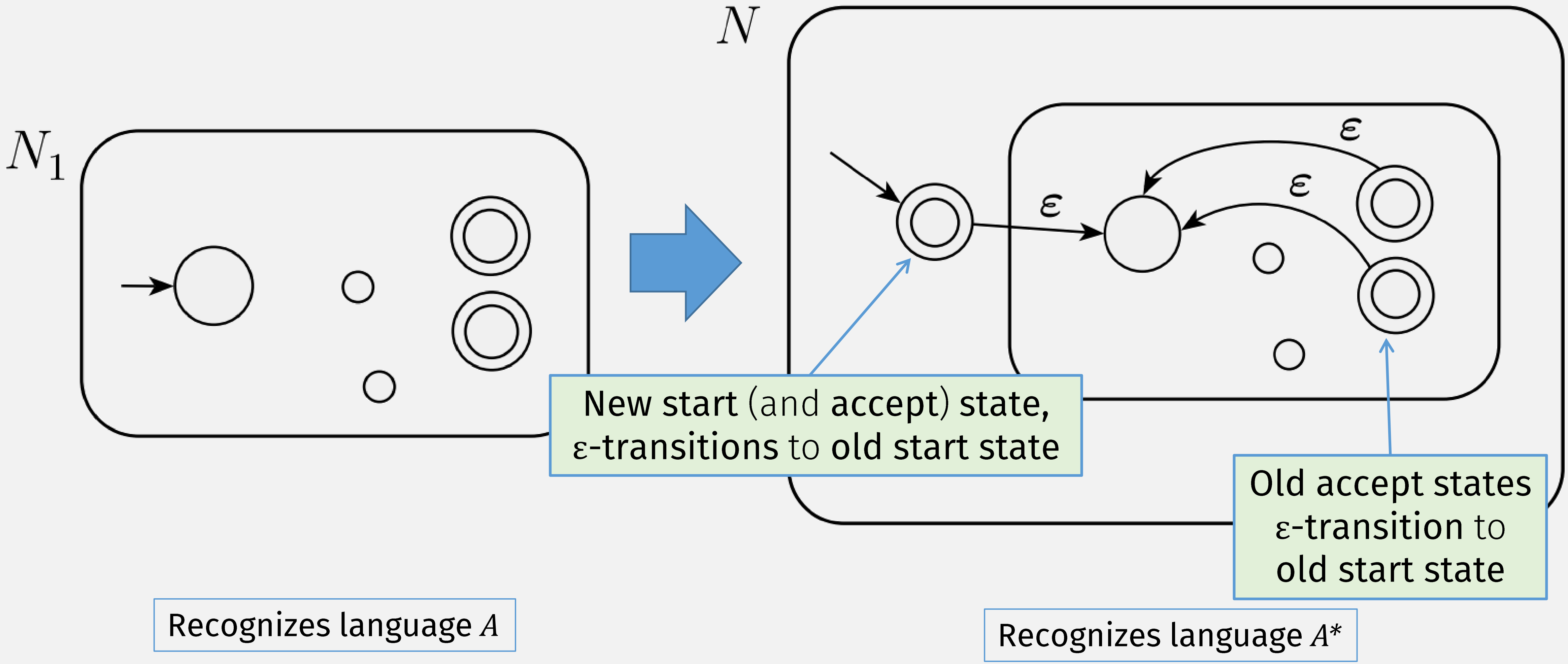
$$A^* = \begin{array}{l}\{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad,} \\ \text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, } \ldots\}\end{array}$$

Note: repeat zero or more times

(this is an infinite language!)

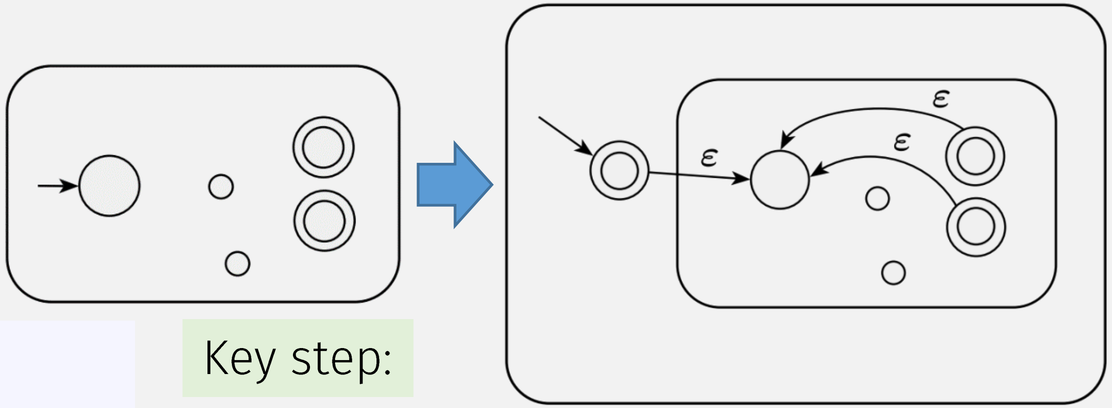
Star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star is Closed for Regular Langs?



In-class exercise

Kleene Star is Closed for Regular Langs



THEOREM

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

Key step:

$\mathrm{STAR}_{\mathrm{NFA}}$: NFA → NFA

where $L(\mathrm{STAR}_{\mathrm{NFA}}(N_1)) = L(N_1)^*$

In-class solution

Kleene Star is Closed for Regular Langs

(part of)

PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1.

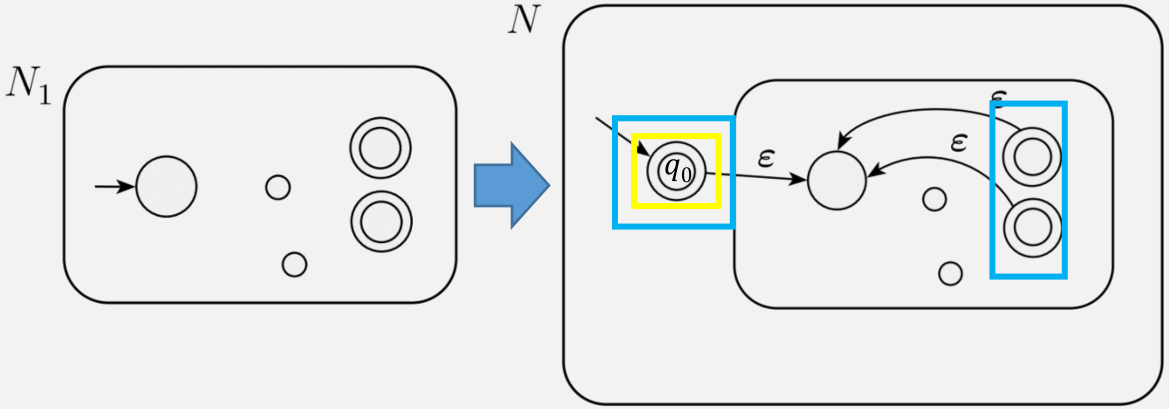
$N = \mathrm{STAR}_{\mathrm{NFA}}(N_1)$ $= (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

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

2. The state q_0 is the new start state.

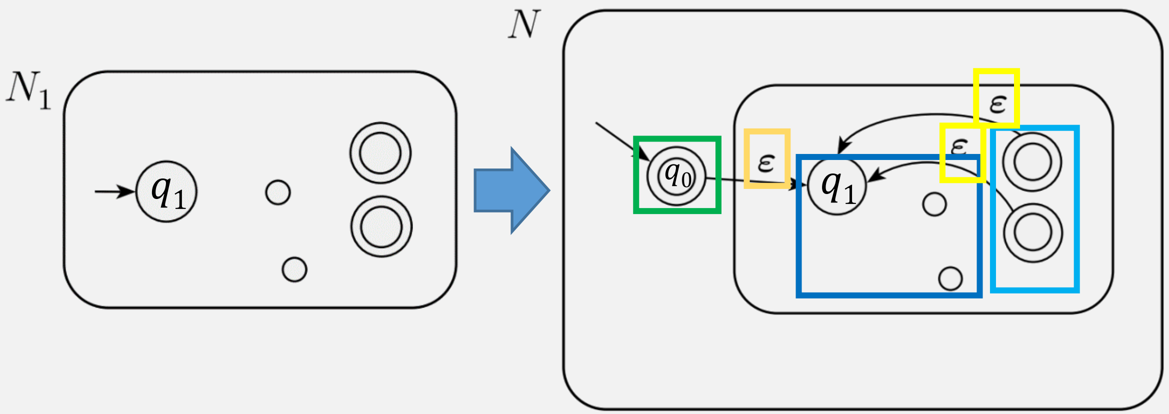
3. $F = \{q_0\} \cup F_1$

Kleene star of a language must accept the empty string!



In-class solution

Kleene Star is Closed for Regular Langs



(part of)

PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1.

$N = \text{STAR}_{\text{NFA}}(N_1)$ $= (Q, \Sigma, \delta, q_0, F)$ 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 $a \in \Sigma_\varepsilon$,

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

Old accept states ε-transition to old start state

New start state ε-transitions to old start state

New start state has only ε-transitions

F_1 states might already have empty transitions!

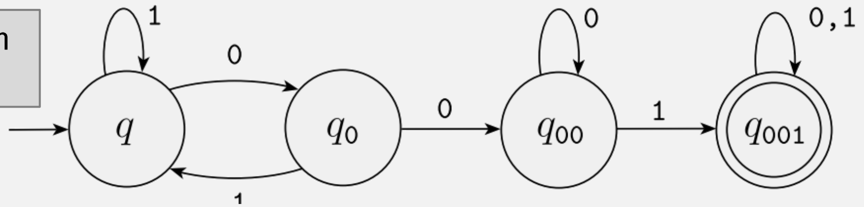
Why These (Closed) Operations?

- **Union** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Concatenation** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- **Kleene star** (repetition) $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

<u>All</u> **regular languages** can be <u>constructed</u> **from:**
- (language of) **single-char strings** (from some alphabet)**,** and
- these **three closed operations!**

So Far: Regular Language Representations

1. State diagram (NFA/DFA)



2. Formal description

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

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state
5. $F = \{q_2\}$

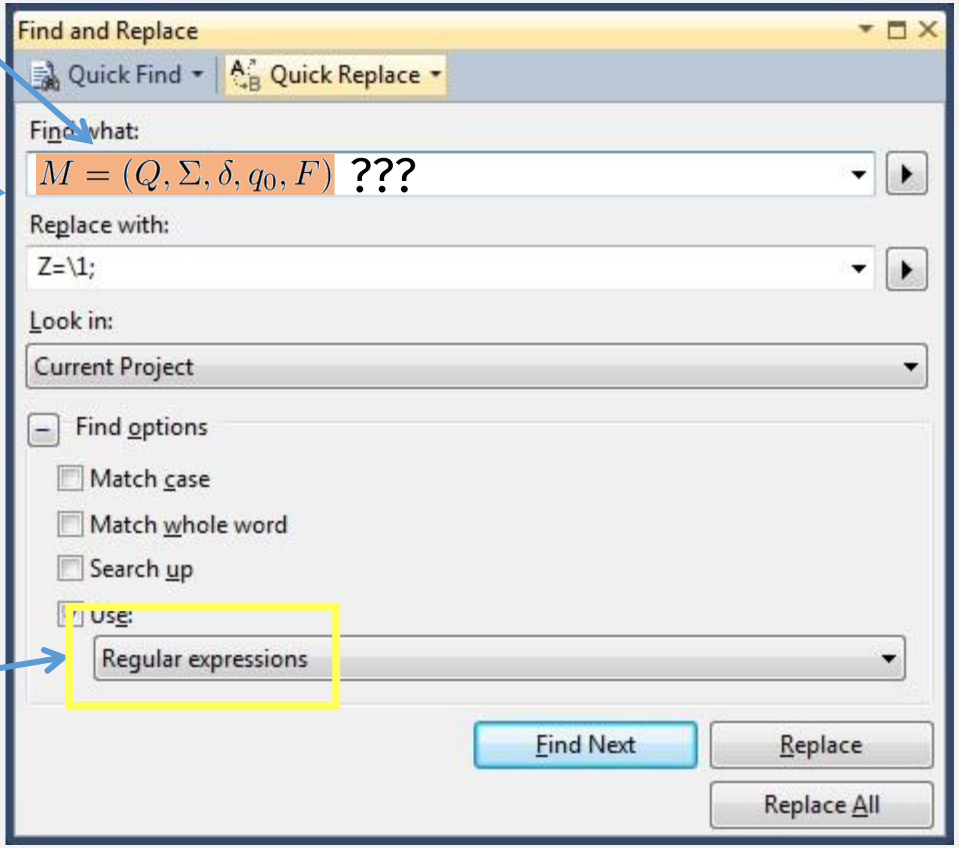
(hard to write)

Our Running Analogy:
- Set of all **regular languages** ~ a "programming language"
- One **regular language** ~ a "program"

? 3. $\Sigma^* 001 \Sigma^*$

Need a more concise (textual) notation??

Actually, **it's a real programming language, for text search / string matching** computations



Regular Expressions:
A Widely Used Programming Language
(in other tools / languages)

- Unix / Linux
- Java
- Python
- Web APIs

java.util.regex

Class Pattern

java.lang.Object
  java.util.regex.Pattern

```
GREP(1)                  General Commands Manual                  GREP(1)

NAME
       grep, egrep, fgrep, rgrep - print lines matching a pattern

SYNOPSIS
       grep [OPTIONS] PATTERN [FILE...]
       grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
       grep  searches the named input FILEs (or standard input if no files are
       named, or if a single hyphen-minus (-) is given as file name) for lines
       containing  a  match to the given PATTERN.  By default, grep prints the
       matching lines.
```

Python » English 3.8.6rc1 Documentation » The Python Standard Library » Text Processing Services »

— Regular expression operations

ce code: Lib/re.py

nodule provides regular expression matching operations similar to those found in Perl.

About regular expressions (regex)

Analytics supports regular expressions so you can create more flexible definitions for things like view filters, goals, segments, audiences, content groups, and channel groupings.

> This article covers regular expressions in both Universal Analytics and Google Analytics 4.

In the context of Analytics, regular expressions are specific sequences of characters that broadly or narrowly match patterns in your Analytics data.

For example, if you wanted to create a view filter to exclude site data generated by your own employees, you could use a regular expression to exclude any data from the entire range of IP addresses that serve your employees. Let's say those IP addresses range from 198.51.100.1 - 198.51.100.25. Rather than enter 25 different IP addresses, you could create a regular expression like **198\.51\.100\.\d*** that matches the entire range of addresses.

Why These (Closed) Operations?

- Union
- Concatenation
- Kleene star (repetition)

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

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

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

All **regular languages** can be constructed **from:**

- (language of) **single-char strings** (from some alphabet), and

- these **three closed operations!**

They are used to define **regular expressions!**

Regular Expressions: Formal Definition

R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

This is a <u>recursive definition</u>