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

Previously

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

Is Concatenation Closed?

THEOREM

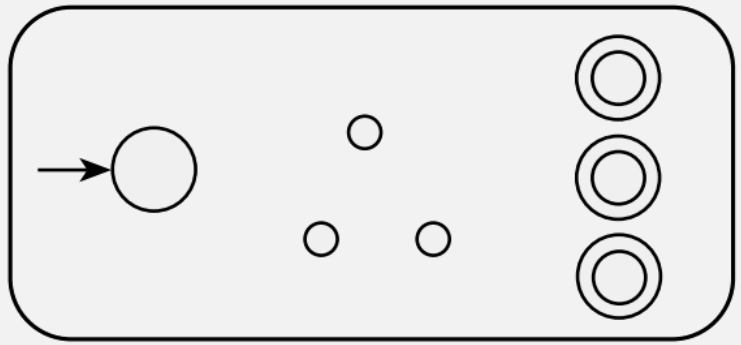
The class of regular languages is closed under the concatenation operation.

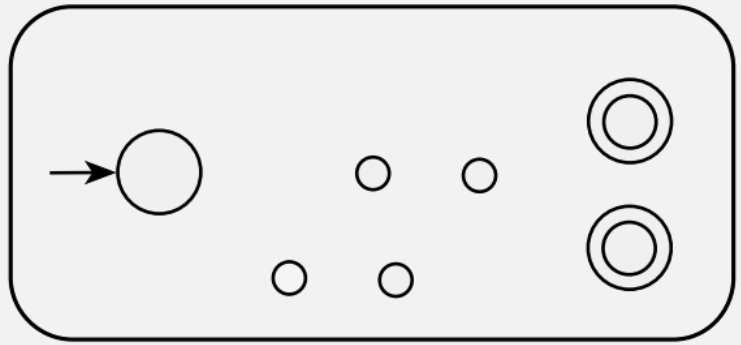
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

Proof requires: Constructing new machine

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

Concatentation





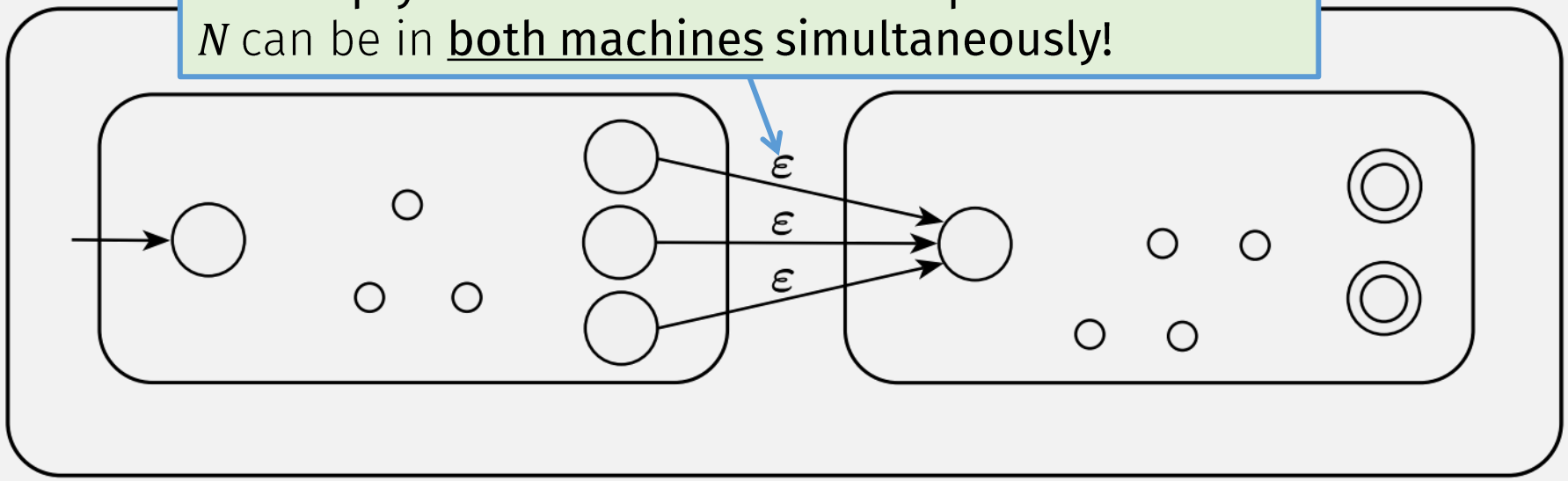
Let M_1 recognize A_1, and M_2 recognize A_2.

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 <u>both machines</u> simultaneously!



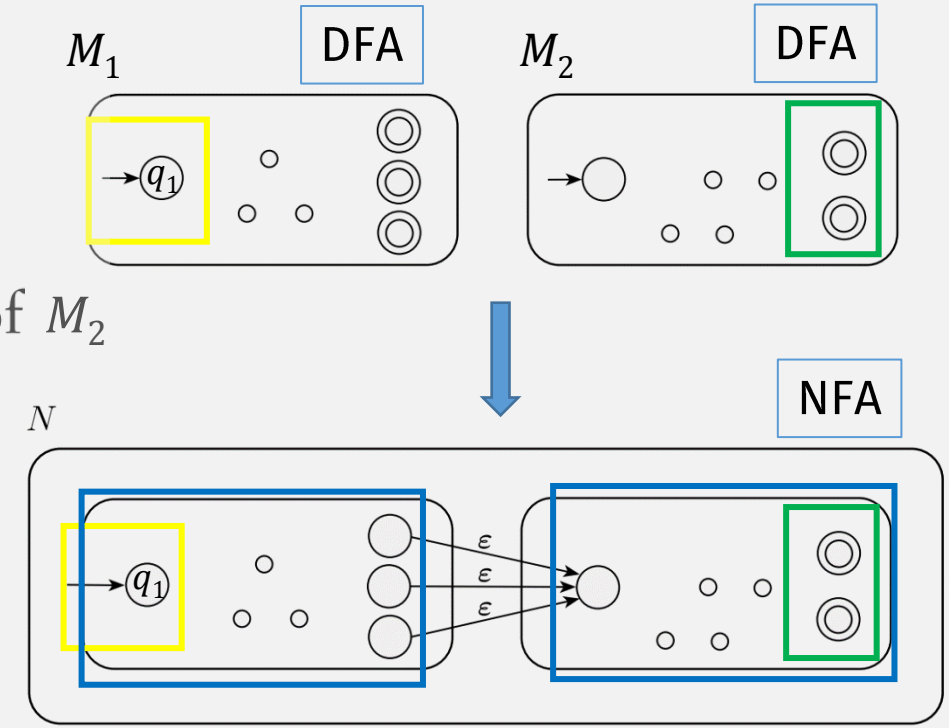
Concatenation is Closed for Regular Langs

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:

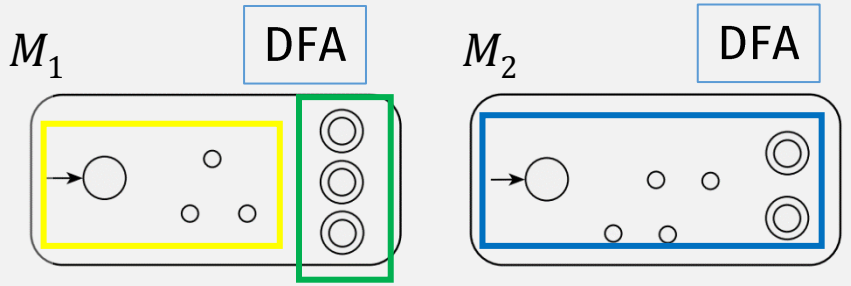
$\text{CONCAT}_{\text{DFA-NFA}}(M_1, M_2) = N =$ $(Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

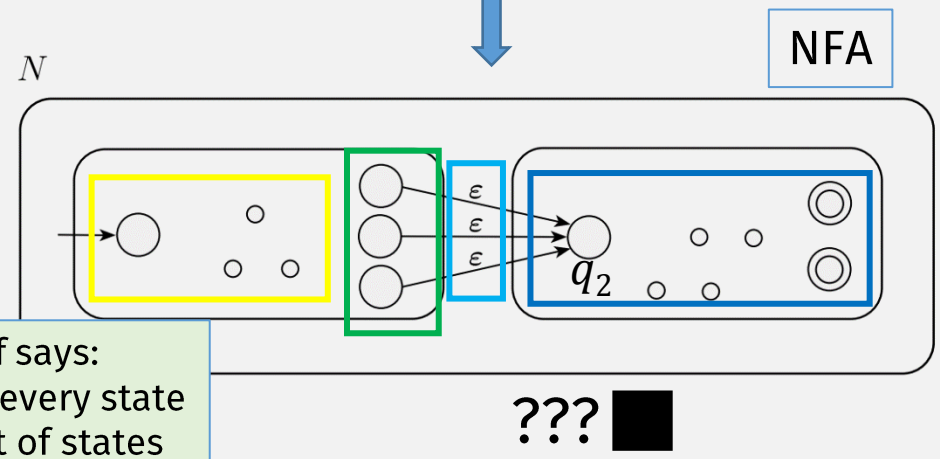
1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of M_1
3. The accept states F_2 are the same as the accept states of M_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

(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 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 ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$
6. ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$ 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. ??? Does NFA recognize reg langs?
7. From stmt #1 and #6

Q.E.D.?

Previously

Definition of Regular Language

If a **DFA** recognizes a language L, then L is a **regular language**

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 ...

... 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 **NFA**

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

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

Specifically, we will 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

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

⇒ 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.

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 = $\text{CONVERT}_{\text{DFA-NFA}}(M)$	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: $r_1 \ldots r_n$, where $r_i \in Q$ and

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

Then N accepts?/rejects? w because …

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

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)

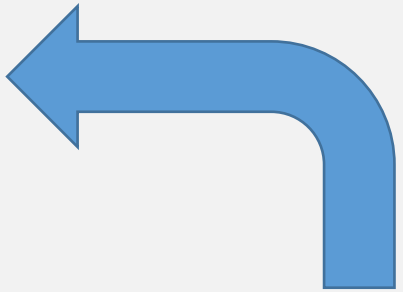
- We know: for L to be **regular**, there must be a **DFA** recognizing it
- Proof Idea for this part: Convert given NFA N → an equivalent DFA

"equivalent" =
"recognizes the same language"

How to convert **NFA→DFA**?

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

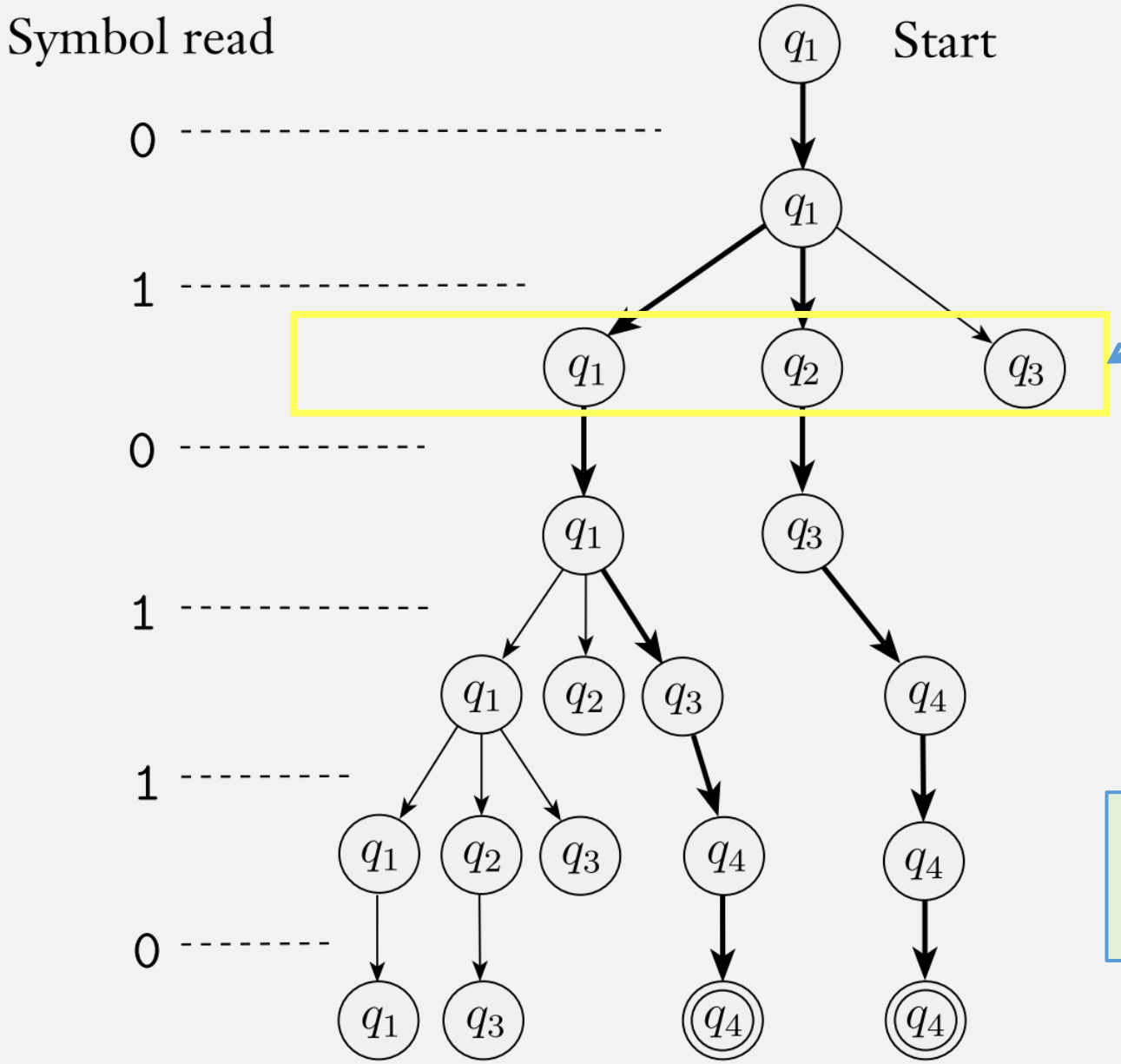
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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>

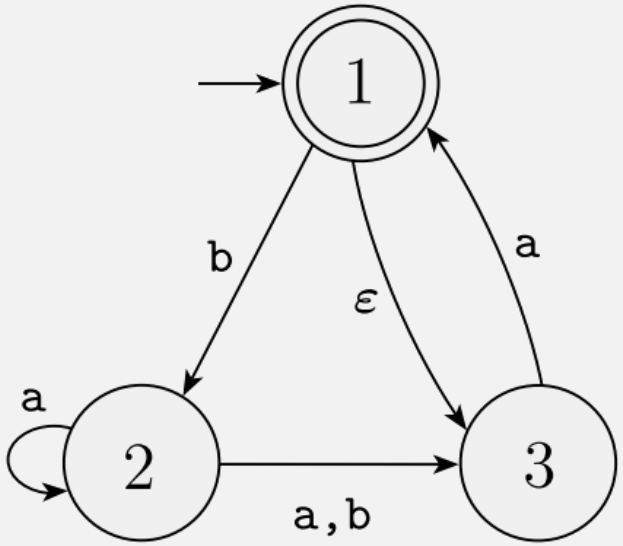
This is similar to the proof strategy from **"Closure of union"** where:
a state = a pair of states

Convert **NFA→DFA**, Formally

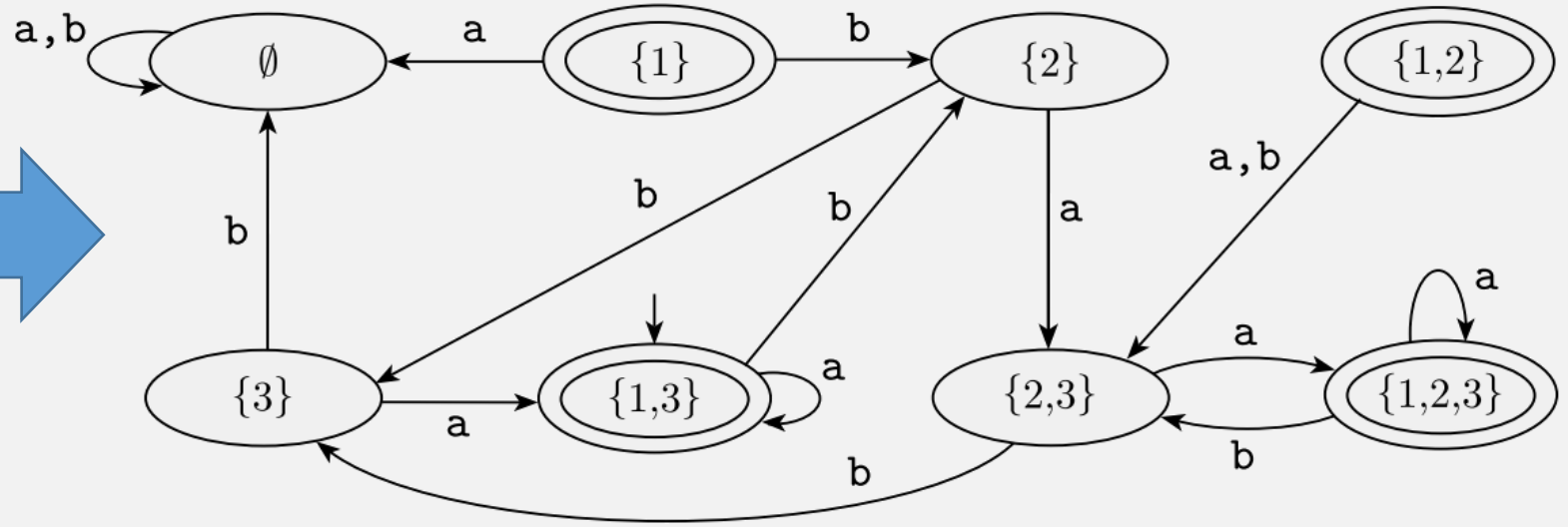
- Let **NFA** N = $(Q, \Sigma, \delta, q_0, F)$
- **An equivalent DFA** M **has states** Q' = $\mathcal{P}(Q)$ (power set of Q)

Example:

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

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

1. $Q_{DFA} = \mathcal{P}(Q_{NFA})$ A DFA state = a set of NFA states

qs = DFA state = set of NFA states

2. For $qs \in Q_{DFA}$ and $a \in \Sigma$
   - $\delta_{DFA}(qs, a) = \bigcup_{q \in qs} \delta_{NFA}(q, a)$ A DFA step = an NFA step for all states in the set

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

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

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

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

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

Almost the same, except …

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

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

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

4. $F_{DFA} = \{ qs \in Q_{DFA} \mid qs \text{ contains accept state of } N \}$

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.
(Harder)

Examples table?

- 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 ☑

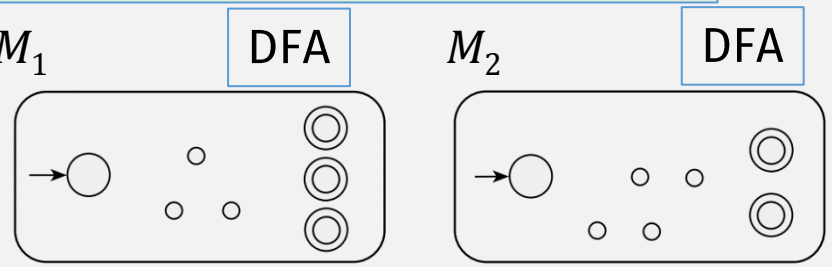
PROOF

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

Wait, is this true?

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

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



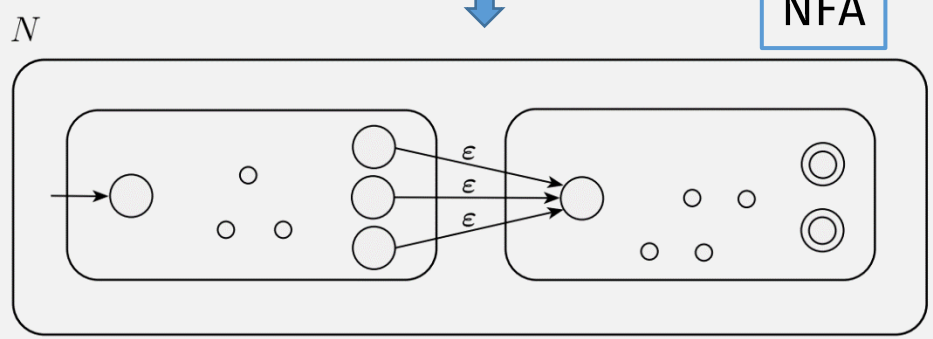
1. $Q = Q_1 \cup Q_2$

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 $a \in \Sigma_\varepsilon$,

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



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

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 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 $A_1 \circ A_2$	5. See Examples Table Thm1.40⇐
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

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 (Thm1.40⇒)
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

New possible proof strategy!

Concat Closed for Reg Langs: Use NFAs Only

PROOF

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

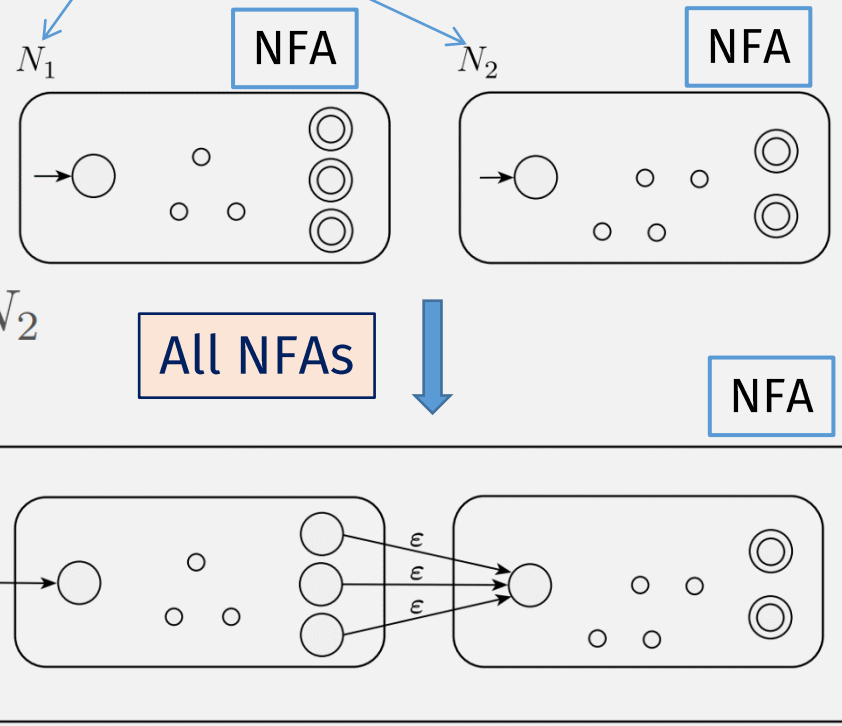
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
NFAs $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

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

????

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



New possible proof strategy!

Sipser Thm 1.47

Concat Closed for Reg Langs: Use NFAs Only

PROOF

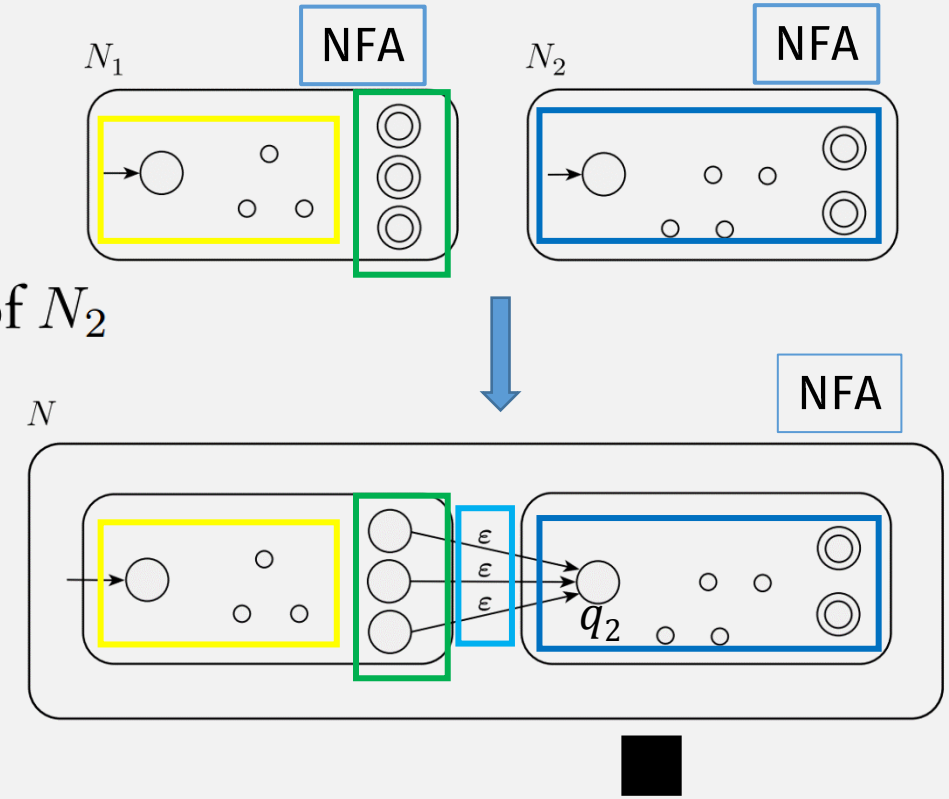
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
NFAs $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

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

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

F_1 states might already have empty transitions!



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

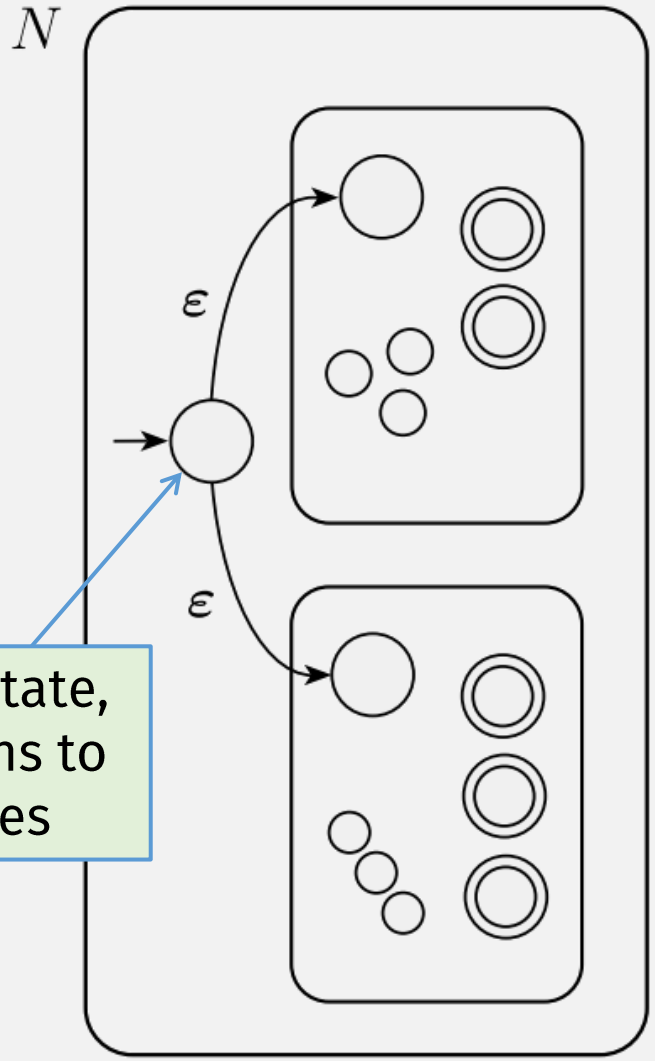
- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: $\text{UNION}_{\text{DFA}}(M_1, M_2) = M = (Q, \Sigma, \delta, q_0, F)$ using M_1 and M_2
- states of M: $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in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

  M step = a step in M_1 + a step in M_2
- M start state: (q_1, q_2)

  Accept if <u>either</u> M_1 or M_2 accept
- M accept states: $F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Union is Closed for Regular Languages



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

Alternate Proof, with NFAs

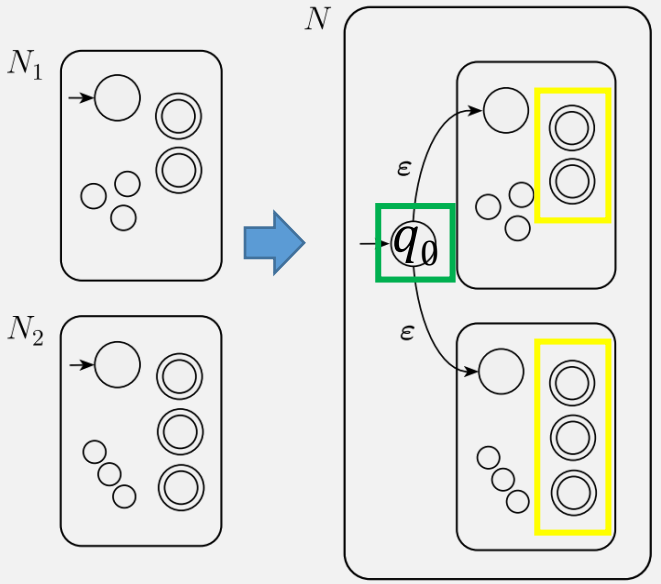
Union is Closed for Regular Languages

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.

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



Sipser Thm 1.45

Alternate Proof, with NFAs

Union is Closed for Regular Languages

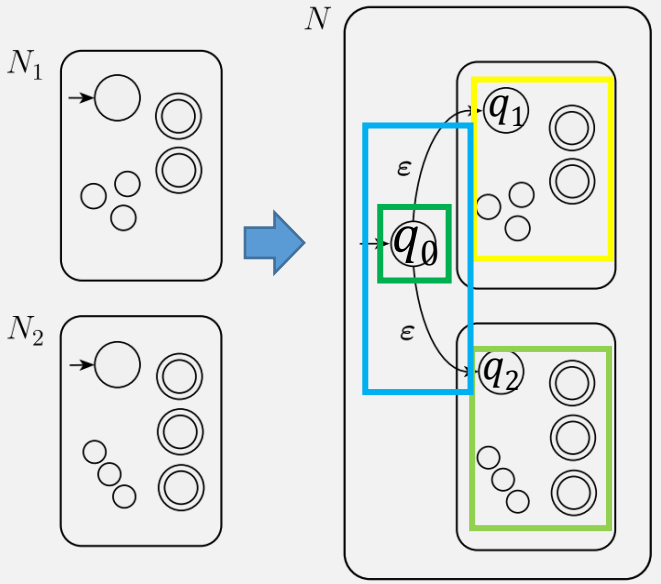
PROOF

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.

$\text{UNION}_{\text{NFA}}(N_1, N_2) = N =$ $(Q, \Sigma, \delta, q_0, F)$ 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: $A^* = \{x_1 x_2 \ldots x_k | \; k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star Example

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

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

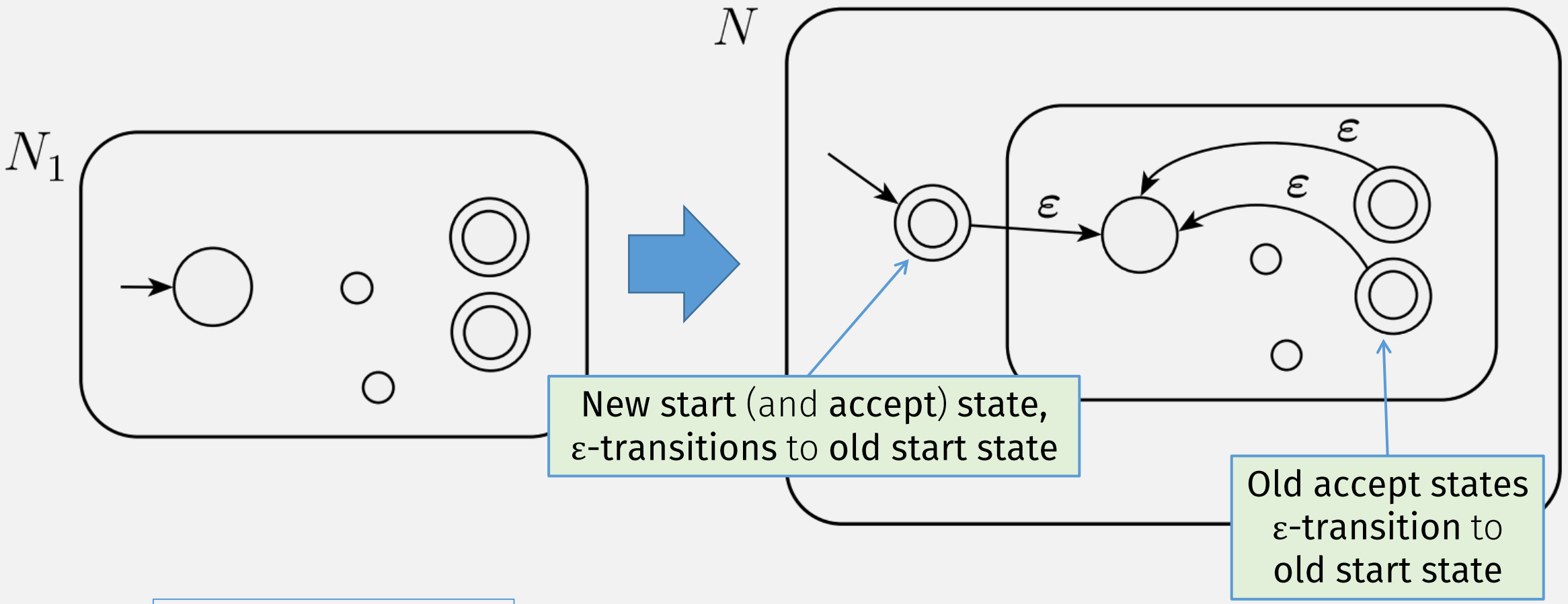
$$A^* = \{\varepsilon, \mathtt{good}, \mathtt{bad}, \mathtt{goodgood}, \mathtt{goodbad}, \mathtt{badgood}, \mathtt{badbad}, \mathtt{goodgoodgood}, \mathtt{goodgoodbad}, \mathtt{goodbadgood}, \mathtt{goodbadbad}, \ldots\}$$

Note: repeat zero or more times

(this is an infinite language!)

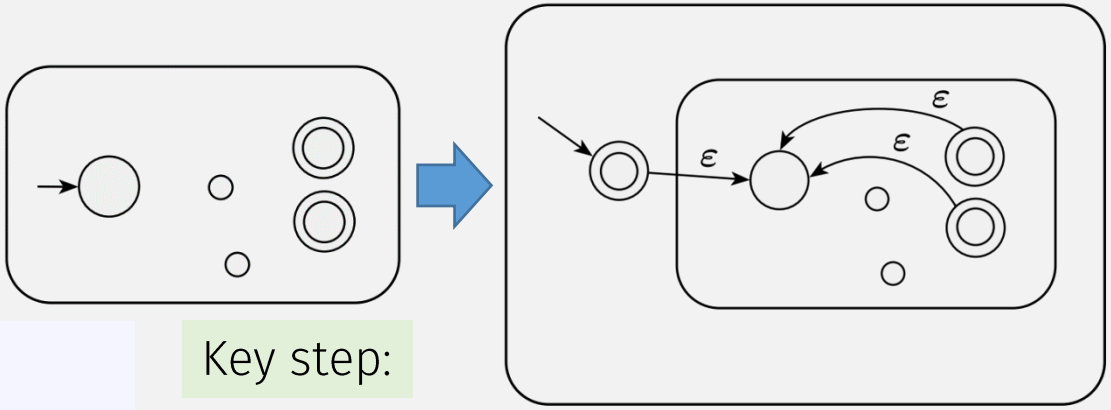
Star: $A^* = \{x_1 x_2 \ldots x_k | \; k \geq 0 \text{ and each } x_i \in A\}$

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:

$\text{STAR}_{\text{NFA}} : \text{NFA} \rightarrow \text{NFA}$

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

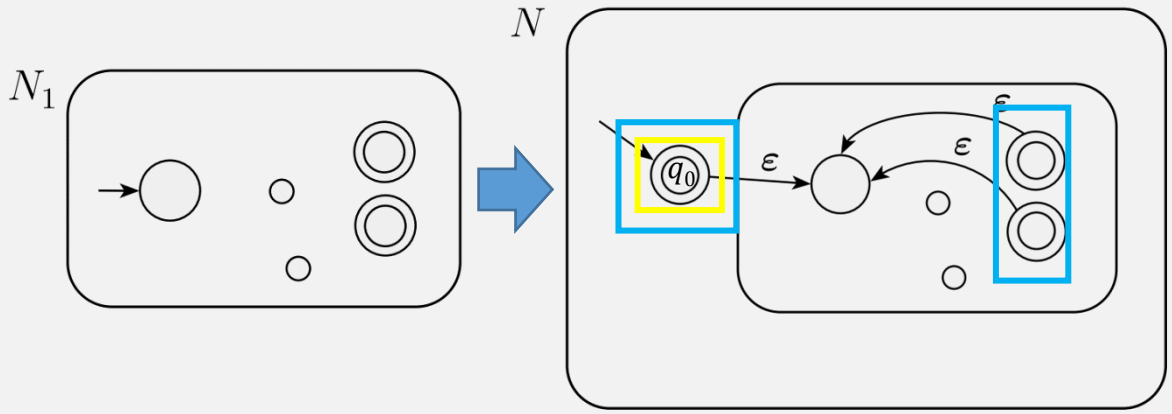
In-class solution

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.

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



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

Kleene star of a language must accept the empty string!

In-class solution

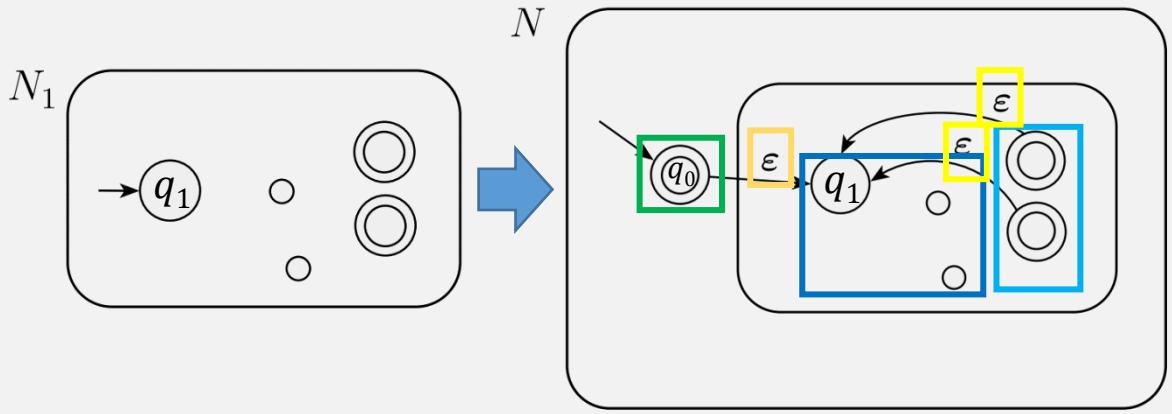
Sipser Thm 1.49

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 $a \in \Sigma_\varepsilon$,

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

F_1 states might already have empty transitions!

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

Why These (Closed) Operations?

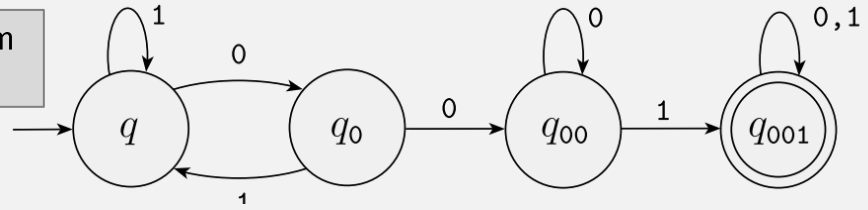
- **Union** $A \cup B = \{x | \, x \in A \text{ or } x \in B\}$
- **Concatenation** $A \circ B = \{xy | \, x \in A \text{ and } y \in B\}$
- **Kleene star** (repetition) $A^* = \{x_1 x_2 \ldots x_k | \, k \geq 0 \text{ and each } x_i \in A\}$

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

- (language of) **single-char strings** (from some alphabet), and
- these **three closed operations!**

So Far: Regular Language Representations

1. State diagram (NFA/DFA)



2. Formal description

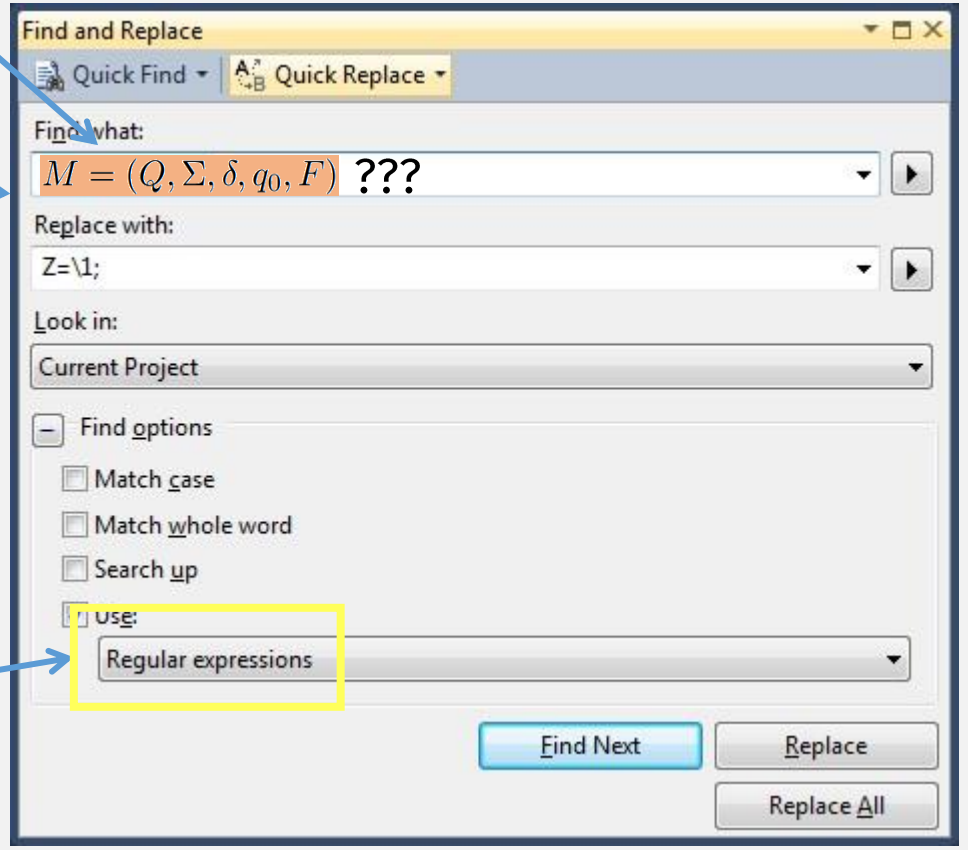
1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	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)

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



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??

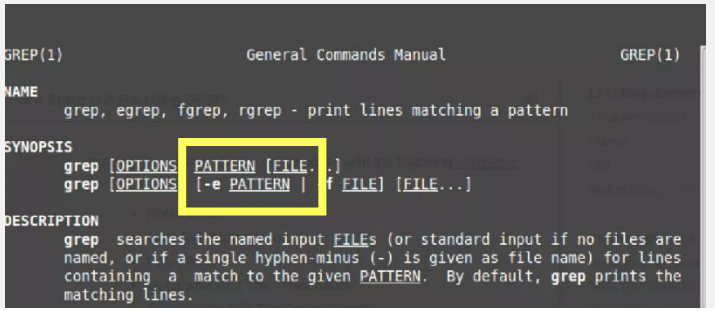
Regular Expressions:
A Widely Used Programming Language
(in other tools / languages)

- Unix / Linux
- Java
- Python
- Web APIs

java.util.regex

Class Pattern

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



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

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.

— Regular expression operations

ce code: Lib/re.py

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

Why These (Closed) Operations?

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

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.

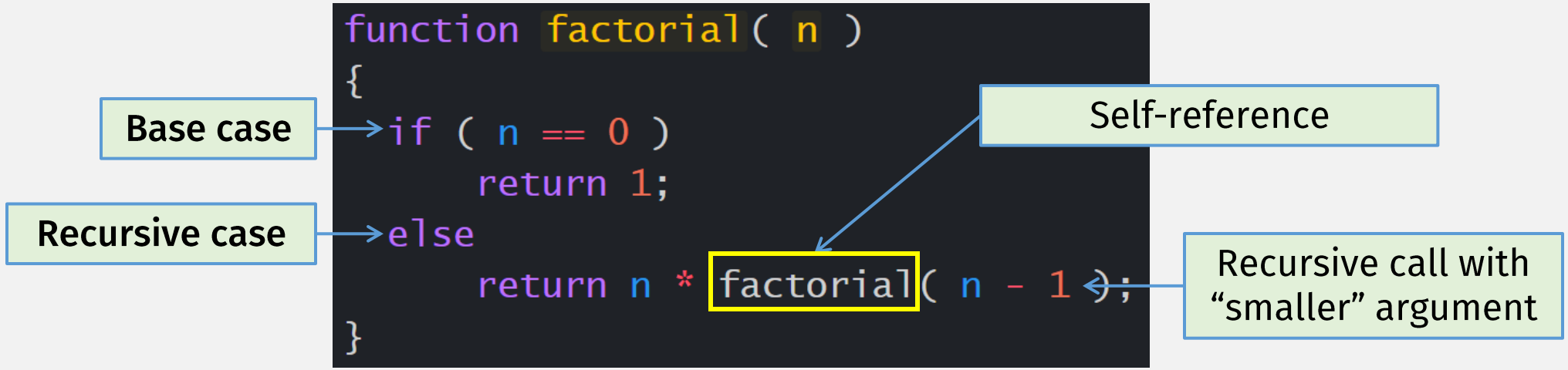
This is a recursive definition

Flashback: Recursive Definitions

Recursive definitions are definitions with a self-reference

A valid recursive definition must have:
- **base case** and
- **recursive case** (with a **"smaller" self-reference**)

Flashback: Recursive Definitions



Flashback: Recursive Definitions

A **Natural Number** is either:

- **Zero**, or
- the **Successor** of a **Natural Number**

Self-reference

Base case

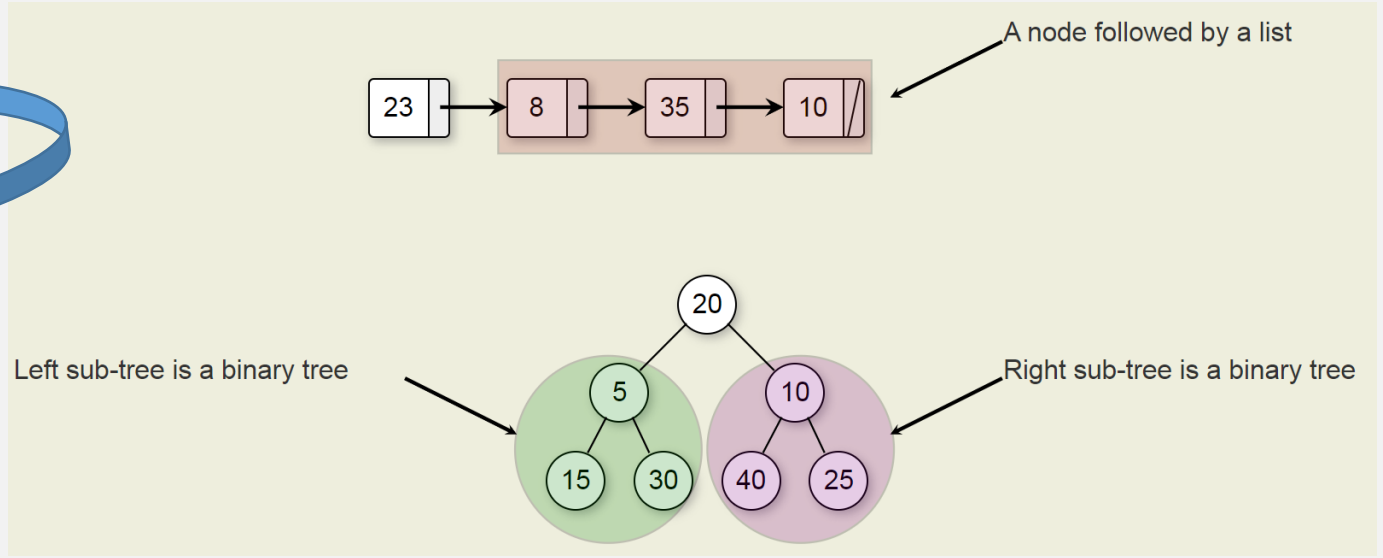
Recursive case

"smaller" argument

Flashback: Recursive Definitions

```
/* Linked list Node*/
class Node {
    int data;
    Node next;
}
```

Smaller self-reference



Q: Where's the base case??

> I call it my billion-dollar mistake. It was the invention of the null reference in 1965.
>
> — Tony Hoare —

Data structures are commonly defined recursively

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.

3 Base Cases

(A lang containing a) **length-1 string**

(A lang containing) **the empty string**

(This is the 3rd **use** of the ε **symbol!**)

The empty set (i.e., a lang containing no strings)

union → 4.

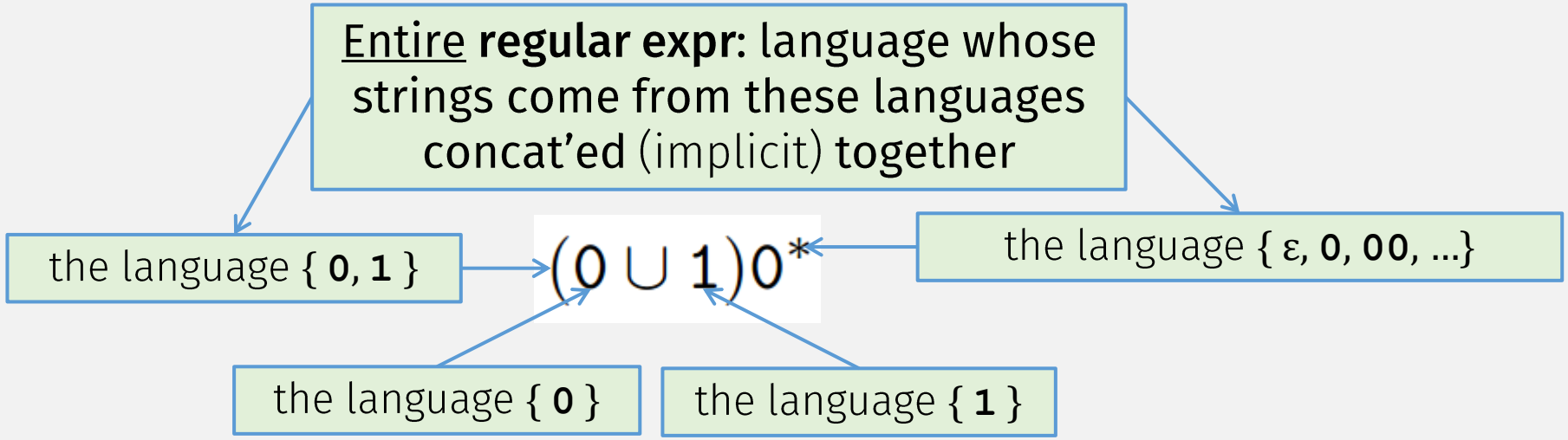
concat → 5.

star → 6.

3 Recursive Cases

Note:
- A **regular expression** represents a **language**
- The *set of all* **regular expressions** represents a *set of* **languages**

Regular Expression: Concrete Example



- Operator Precedence:
  - Parentheses
  - Kleene Star
  - Concat (sometimes use ∘, sometimes implicit)
  - Union

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.

alphabet Σ is $\{0,1\}$

Regular Expression: More Examples

$0^*10^* = \{w | \ w \text{ contains a single } 1\}$

$\Sigma^*1\Sigma^* = \{w | \ w \text{ has at least one } 1\}$ Σ in regular expression = "any char"

$1^*(01^+)^* = \{w | \ \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$ let R^+ be shorthand for RR^*

$(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$ $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$

$1^*\emptyset = \emptyset$ $A \circ B = \{xy | \ x \in A \text{ and } y \in B\}$

nothing in B = nothing in $A \circ B$

$\emptyset^* = \{\varepsilon\}$ Star of any lang has ε

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.

Regular Expressions = Regular Langs?

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.

We would like:
- A **regular expression** represents a **regular language**
- The *set of all* **regular expressions** represents the *set of all* **regular languages**

(But we have to prove it)

Thm: A Lang is Regular **iff** Some Reg Expr Describes It

⇒ If **a language is regular,** then **it's described by a reg expression**

⇐ If **a language is described by a reg expression,** then **it's regular**
(Easier)

- Key step: **convert reg expr → equivalent NFA!**
- (Hint: we mostly did this already when discussing closed ops)

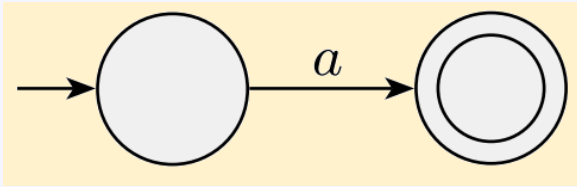
How to show that a language is regular?

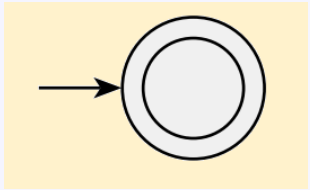
Construct a **DFA** *or* **NFA!**

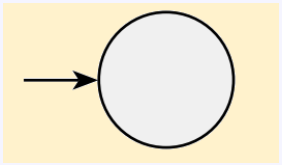
RegExpr→NFA

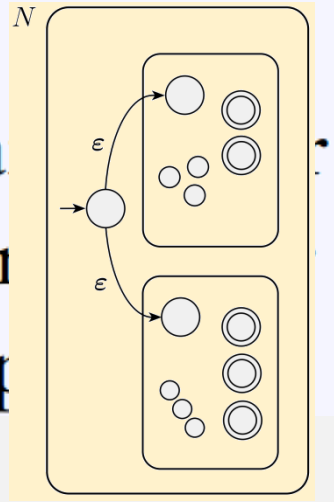
R is a ***regular expression*** if R is

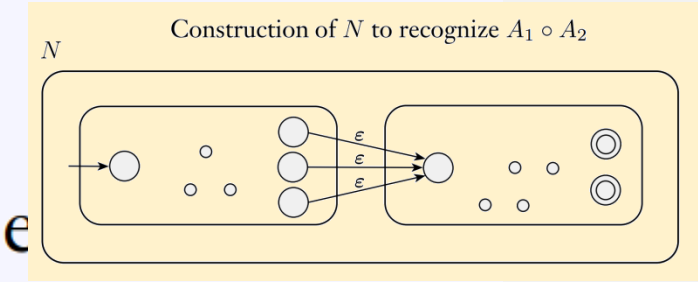
1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 a
5. $(R_1 \circ R_2)$, where R_1 and R_2 a
6. (R_1^*), where R_1 is a regular exp

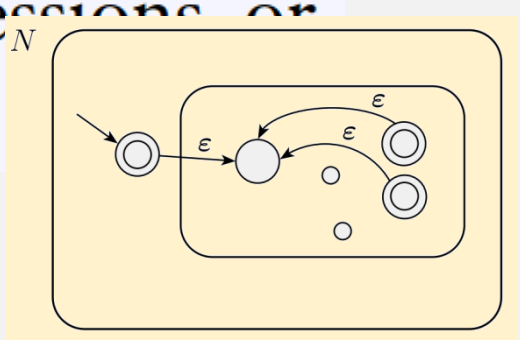












Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If **a language is regular,** then **it's described by a reg expression** (Harder)

- Key step: **Convert an ~~DFA~~ or NFA → equivalent Regular Expression**

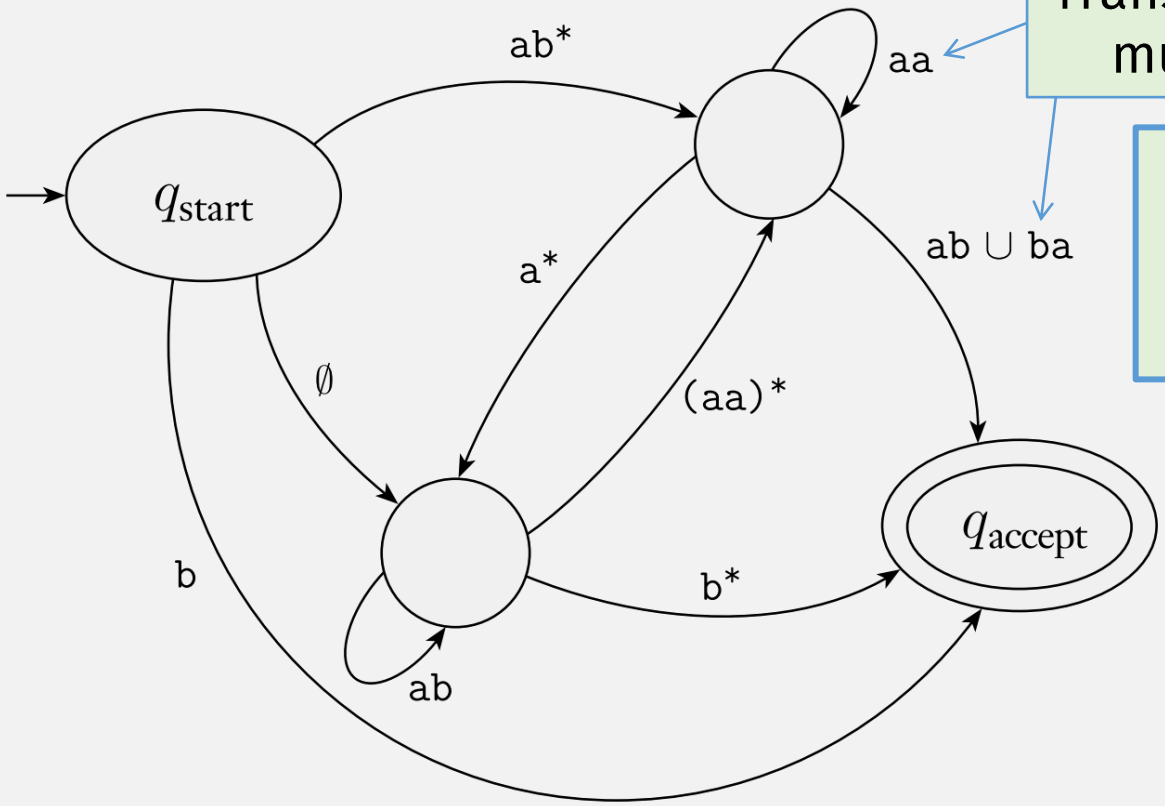
- First, **we need another kind of finite automata: a GNFA**

⇐ If **a language is described by a reg expression,** then **it's regular** (Easier)

☑ • Key step: **Convert the regular expression → an equivalent NFA!**

(full proof requires writing Statements and Justifications, and creating an "Equivalence" Table)

Generalized NFAs (GNFAs)



Transition can read multiple chars

plain NFA
= GNFA with single char regular expr transitions

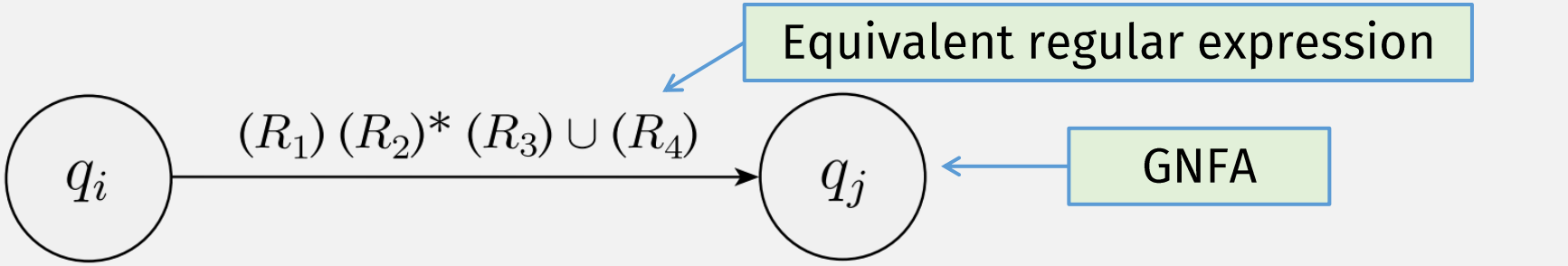
Goal: convert **GNFAs** to equivalent **Regular Exprs**

- GNFA = NFA with regular expression transitions

GNFA→RegExpr function

On **GNFA** input G:

- If G **has 2 states,** **`return`** **the regular expression** (on the transition), e.g.:



Could there be less than 2 states?

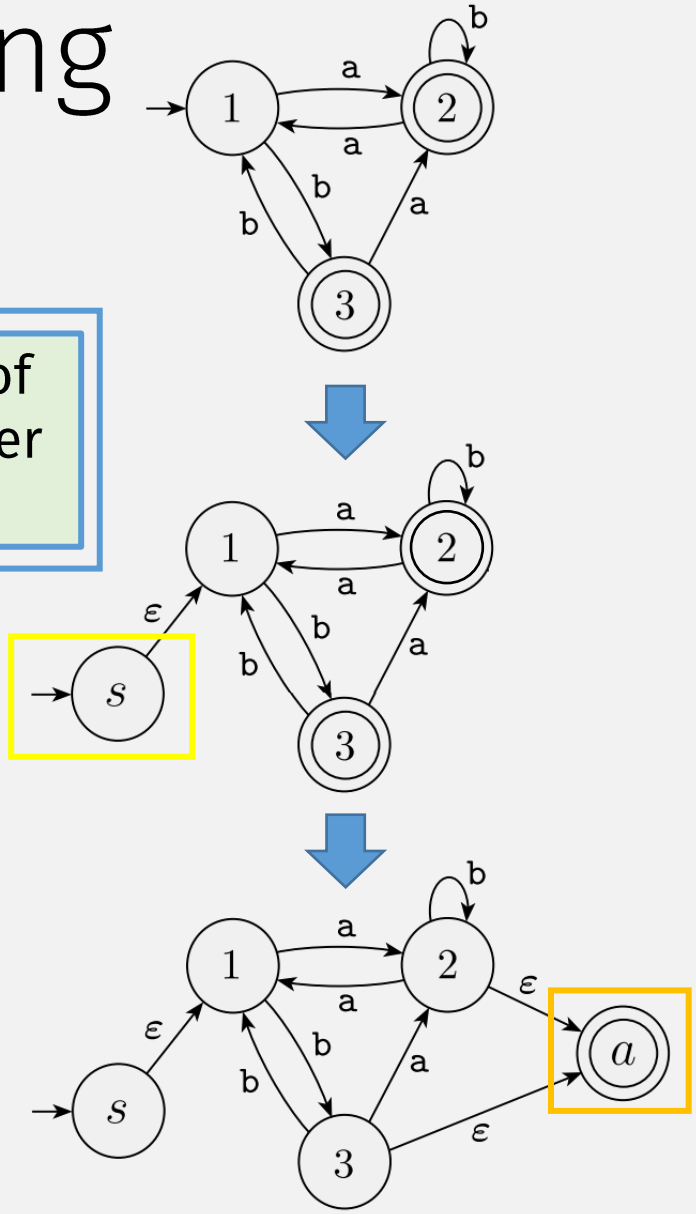
GNFA→RegExpr Preprocessing

- Modify input machine to have:

Does this change the language of the machine? i.e., are before/after machines equivalent?

- New start state:
  - No incoming transitions
  - ε transition to old start state
- New, single accept state:
  - With ε transitions from old accept states

Modified machine always has 2+ states:
- New start state
- New accept state

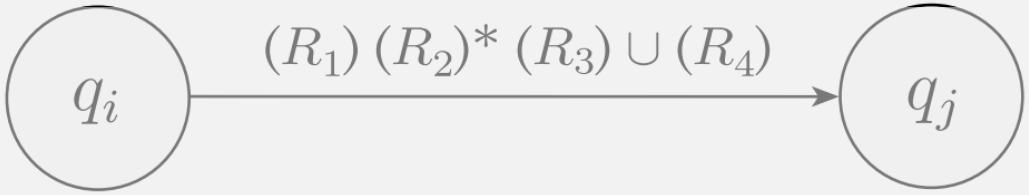


GNFA→RegExpr function (recursive)

On **GNFA** <u>input</u> G:

Base Case

- If G has 2 states, **`return`** the regular expression (from transition), e.g.:

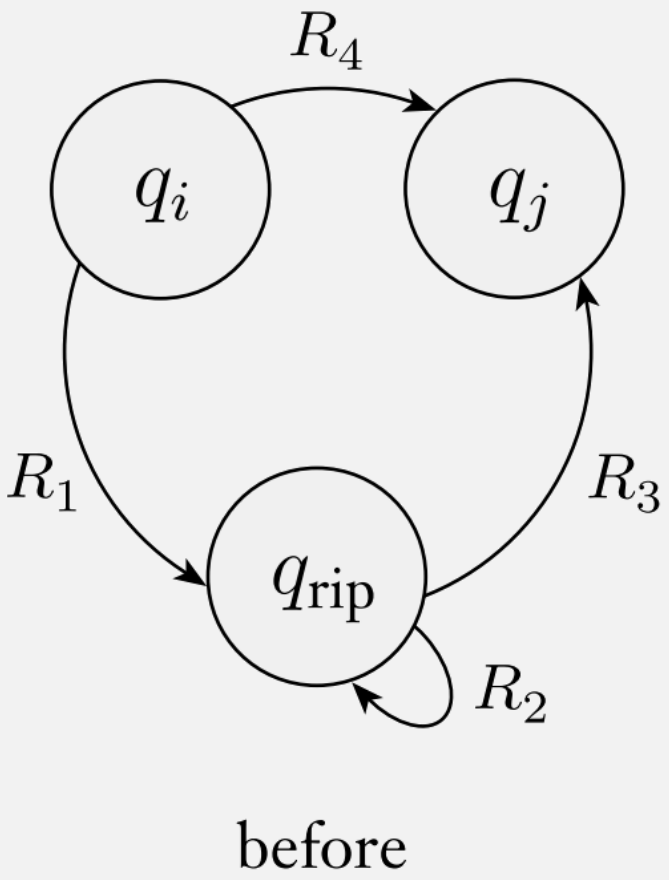


Recursive Case

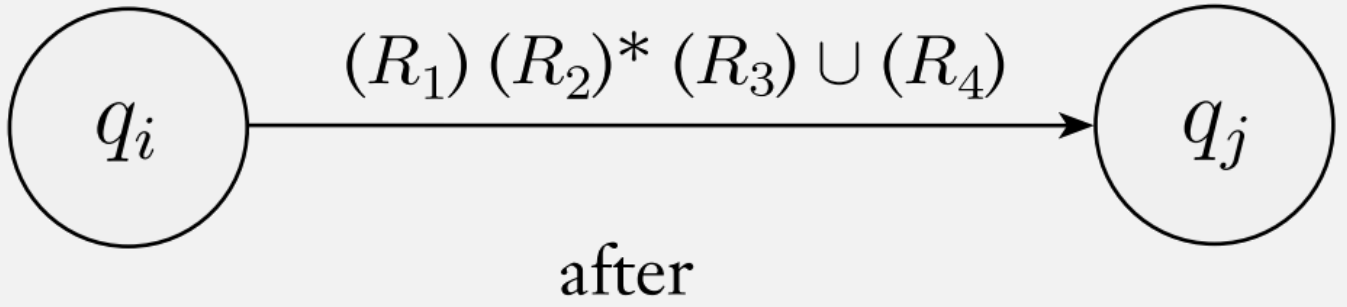
- Else:
  - "Rip out" one state
  - "Repair" the machine to get an <u>equivalent</u> GNFA G'
  - <u>Recursively</u> call **GNFA→RegExpr**(G')

Recursive definitions have:
- <u>base case</u> and
- <u>recursive case</u>
  (with "smaller" self-reference)

GNFA→RegExpr: “Rip/Repair” step



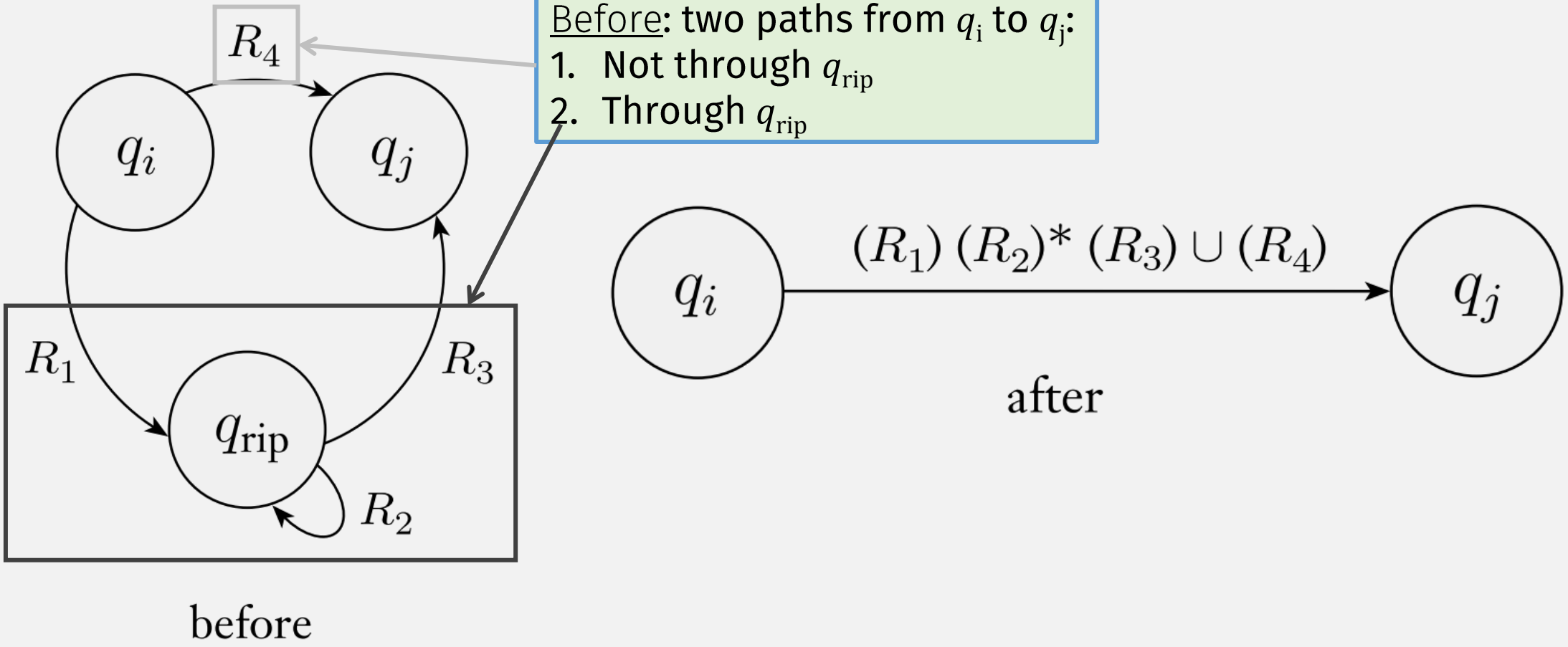
before



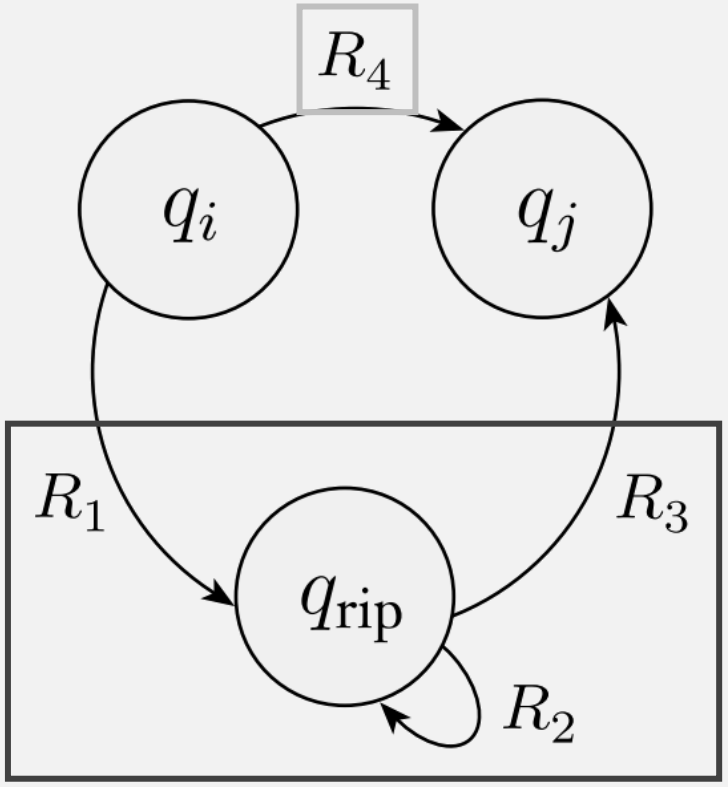
after

To convert a GNFA to a regular expression:
“rip out” state, then “repair”,
and repeat until only 2 states remain

GNFA→RegExpr: "Rip/Repair" step

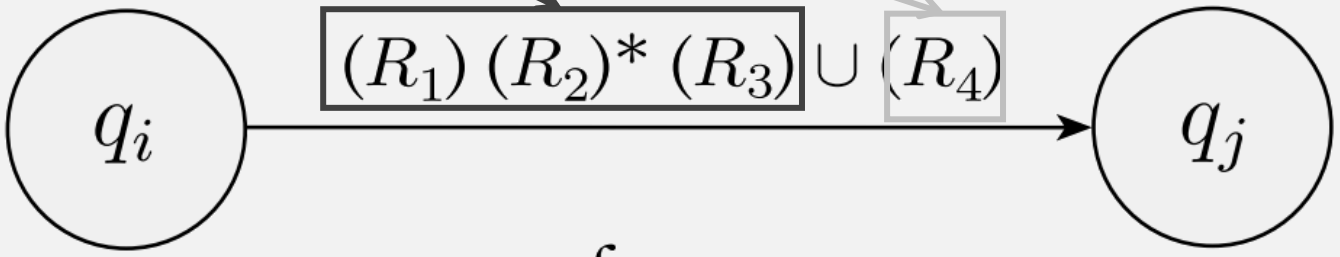


GNFA→RegExpr: "Rip/Repair" step



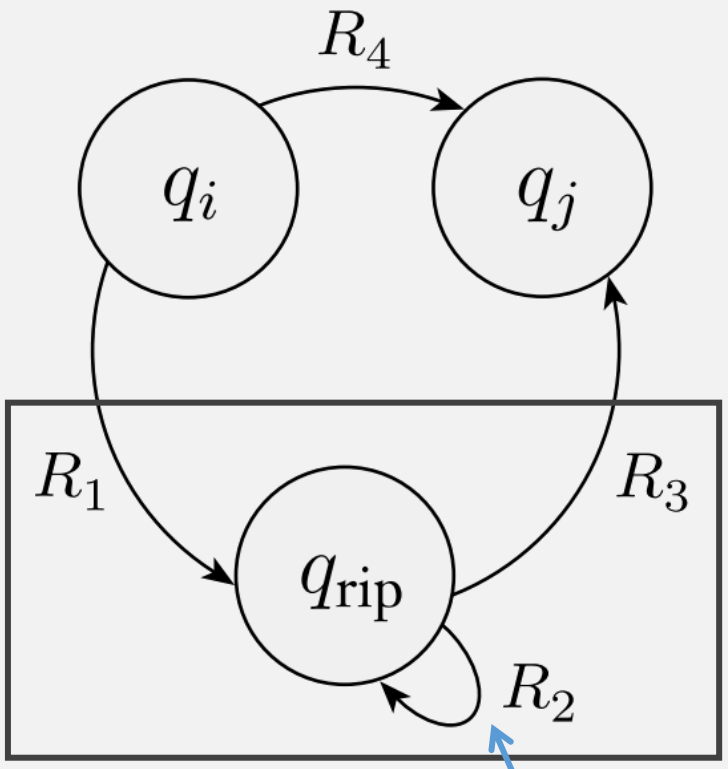
before

After: union of two "paths" from q_i to q_j
1. Not through q_{rip}
2. Through q_{rip}

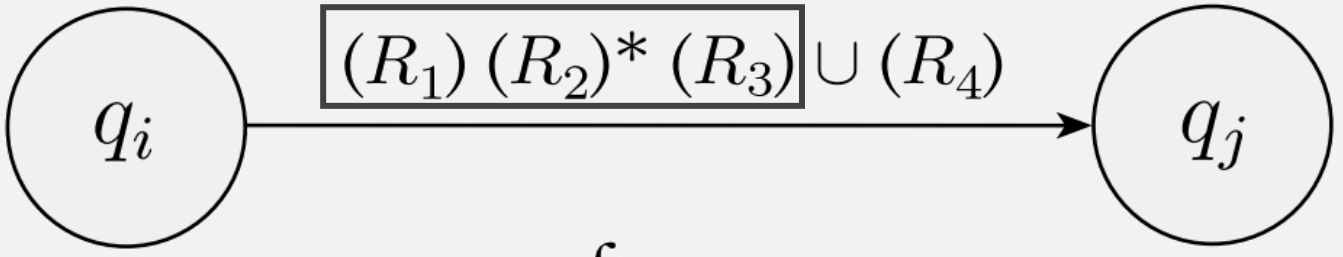


after

GNFA→RegExpr: “Rip/Repair” step



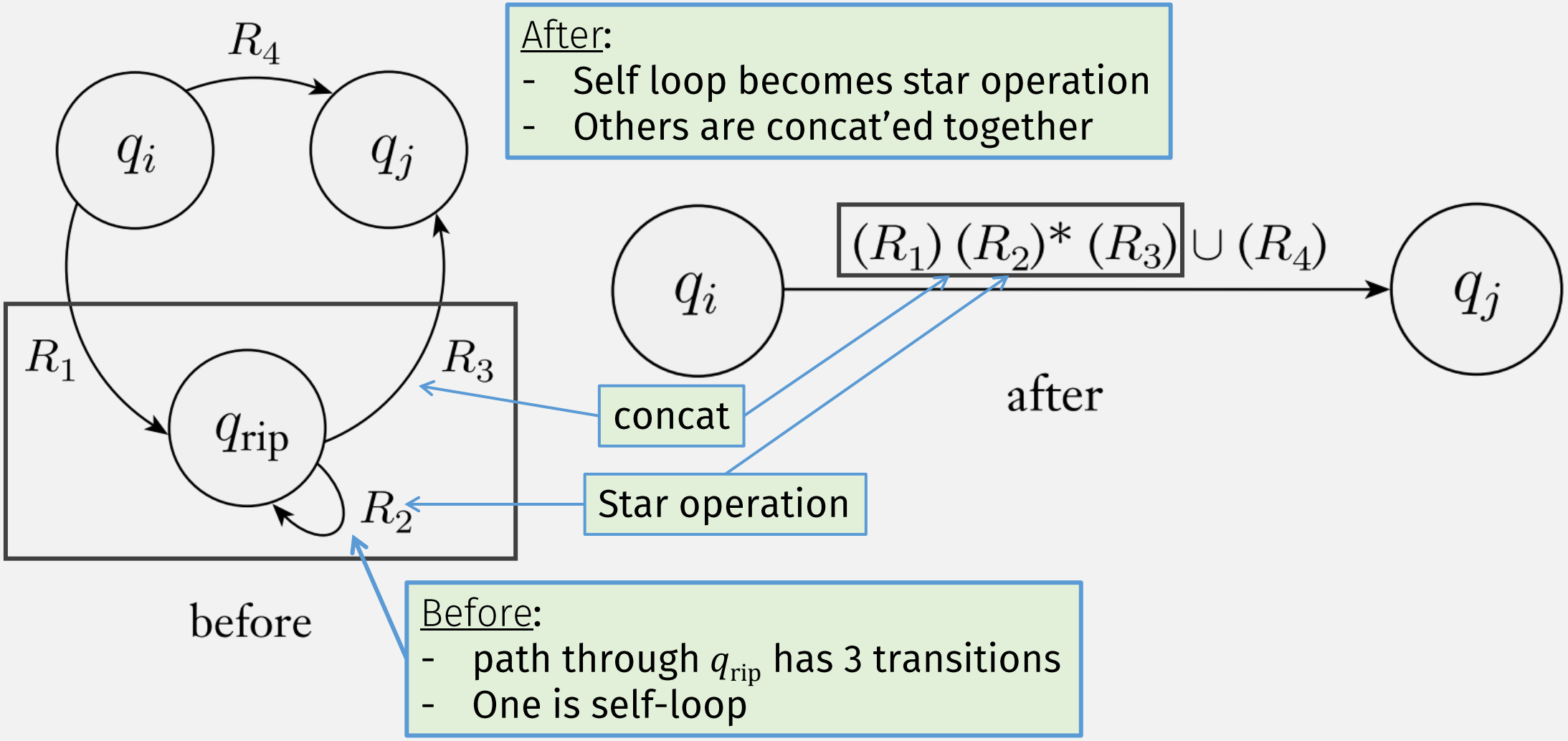
before



after

Before:
- path through q_{rip} has 3 transitions
- One is self-loop

GNFA→RegExpr: "Rip/Repair" step



Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If **a language is regular,** then **it's described by a regular expr**

Need to convert DFA or NFA to Regular Expression …

- Use **GNFA→RegExpr** to convert GNFA → equiv regular expression!

☑ ???

This time, let's really prove equivalence!
(we previously "proved" it with some examples)

⇐ If a language is described by a regular expr, then it's regular

☑ • Convert regular expression → equiv NFA!

GNFA→RegExpr Correctness

- **Correct** = **input** and **output** are **equivalent**
- **Equivalent** = the **language does not change** (same strings)**!**

Statement to Prove:

???

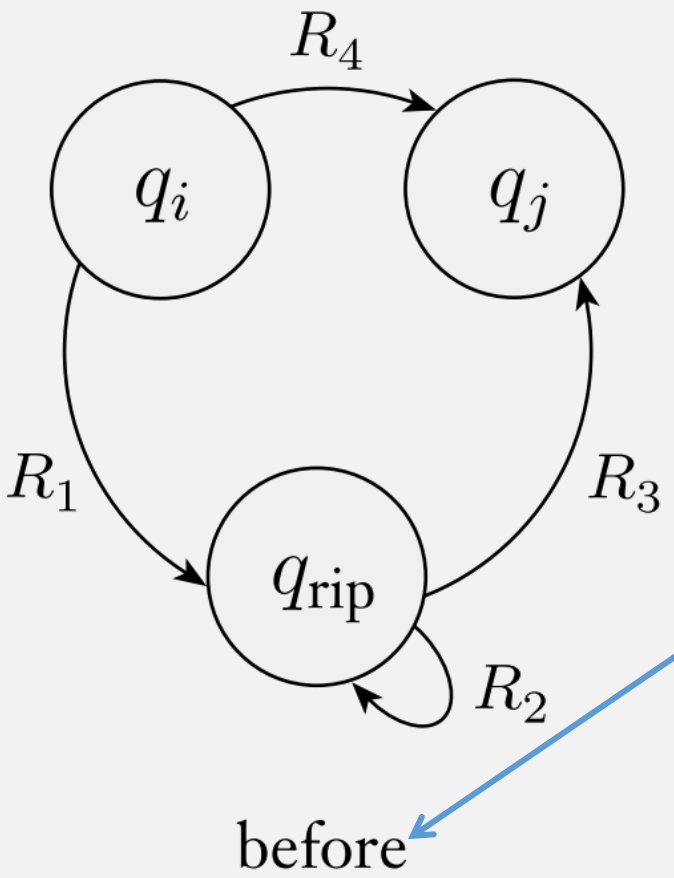
$$\text{LANGOF}(G) = \text{LANGOF}(R)$$

We are ready to <u>really prove</u> equivalence!
(we previously "proved" it with some examples)

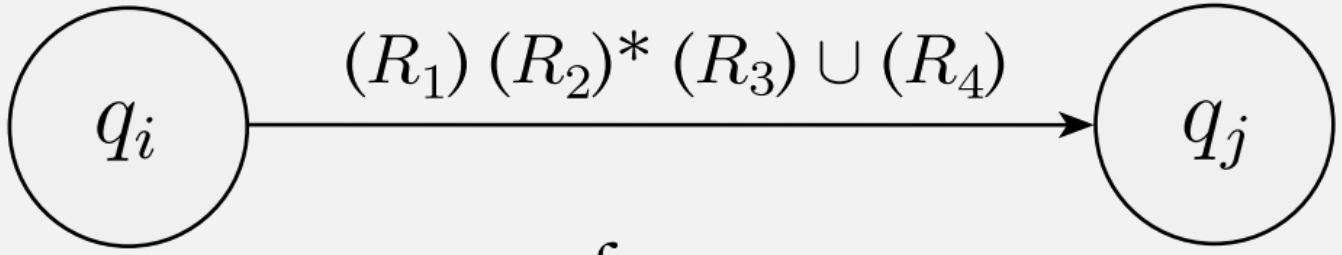
- <u>where</u>:
  - G = a GNFA
  - R = a Regular Expression = **GNFA→RegExpr**(G)

- Key step: **the rip/repair step**

GNFA→RegExpr: Rip/Repair Correctness



before

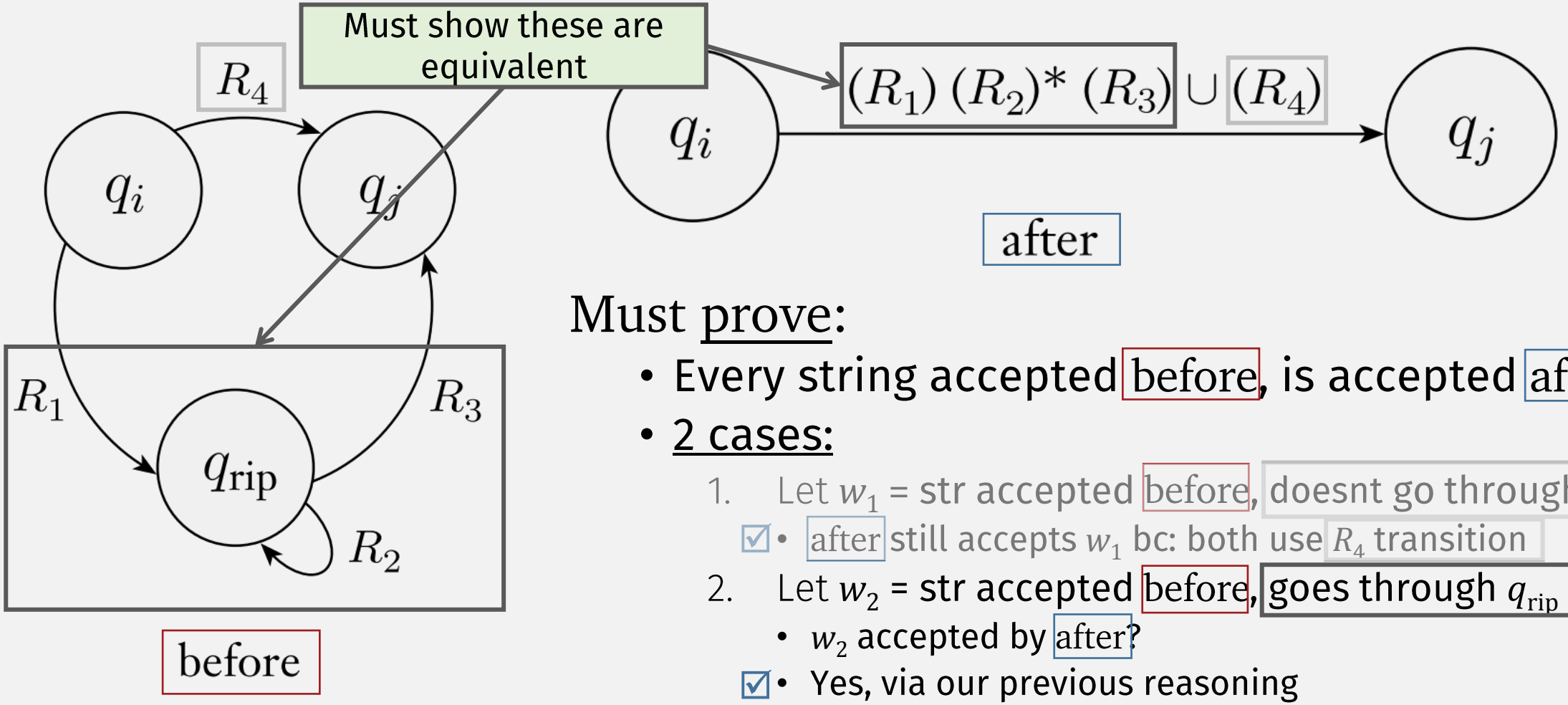


after

Must show these are equivalent

Equivalent = same language = accepts the same strings

GNFA→RegExpr: Rip/Repair Correctness



Must prove:

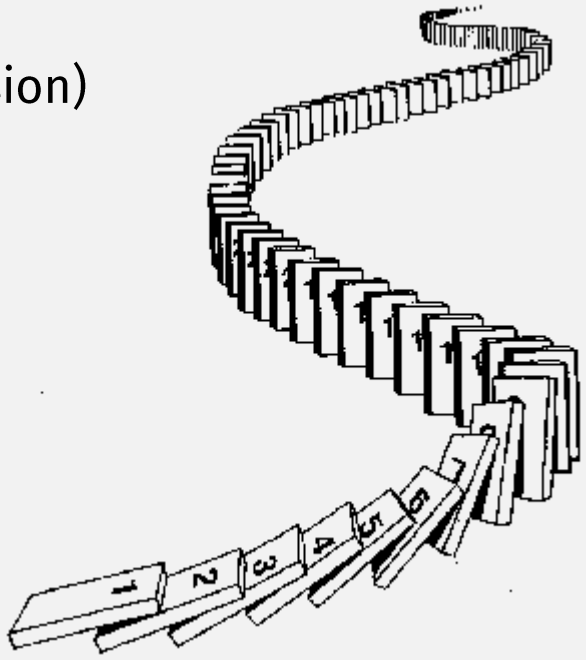
- Every string accepted before, is accepted after
- 2 cases:
  1. Let w_1 = str accepted before, doesnt go through q_{rip}
     - ☑ after still accepts w_1 bc: both use R_4 transition
  2. Let w_2 = str accepted before, goes through q_{rip}
     - w_2 accepted by after?
     - ☑ Yes, via our previous reasoning

UMB CS 420

Inductive Proofs

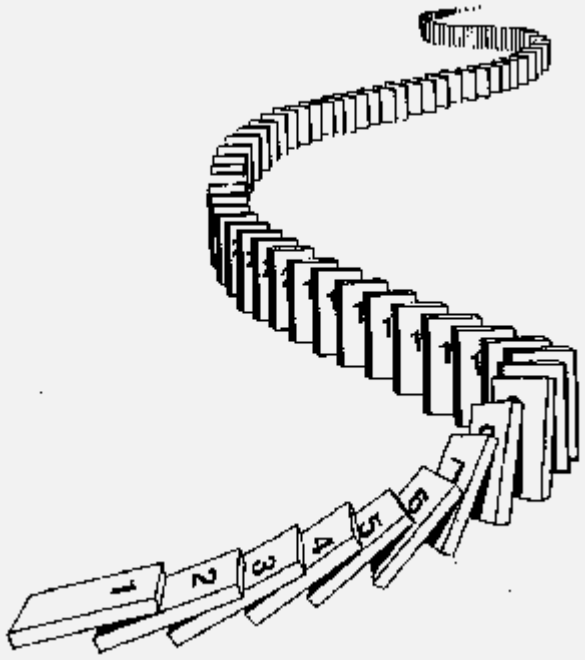
(Proofs involving recursion)

Monday March 4, 2024



Announcements

- HW 3 in
  - ~~Due Mon 3/4 12pm EST (noon)~~
- HW 4 out
  - Due Mon 3/18 12pm EST (noon)
  - (After spring break)



Last Time

Thm: A Lang is Regular **iff** Some Reg Expr Describes It

⇒ If **a language is regular,** then **it's described by a regular expr**

- Use **GNFA→RegExpr** to convert GNFA → equiv regular expression!

☑ ???

This time, let's **really prove equivalence!** (we previously "proved" it with some examples)

⇐ If **a language is described by a regular expr,** then **it's regular**

☑ • Convert regular expression → equivalent NFA!

GNFA→RegExpr Equivalence

- **Equivalent** = the **language does not change** (i.e., same set of strings)**!**

Statement to Prove:

input

output ???

$$\text{LangOf}(G) = \text{LangOf}(R)$$

This time, let's **really prove equivalence!** (we previously "proved" it with some examples)

- where:
  - G = a GNFA
  - R = a Regular Expression = **GNFA→RegExpr**(G)

Language could be infinite set of strings!

(how can we show **equivalence** for a possibly **infinite set of strings**?)

Recursion!

Kinds of Mathematical Proof

- **Deductive proof** (from before)
  - Start with: **assumptions, axioms,** and **definitions**
  - Prove: **news conclusions by making logical inferences** (e.g., **modus ponens**)

- **Proof by induction** (i.e., **"a proof involving recursion"**) (now)
  - Same as above …
  - But: **use this when proving something that is recursively defined**

A valid recursive definition has:
- **base case**(s) and
- **recursive case**(s) (with **"smaller" self-reference**)

Proof by Induction

(A proof for **each case** of some recursive definition)

To Prove: ***Statement*** for recursively defined "thing" x:

1. Prove: *Statement* for base case of x
2. Prove: *Statement* for recursive case of x:
   - Assume: **induction hypothesis** (IH)

     I.e., *Statement* is true for some x_{smaller}
     - E.g., if x is number, then "smaller" = lesser number

   - Prove: *Statement* for x, using IH (and known definitions, theorems …)
     - Typically: show that going from x_{smaller} to larger x is true!

A valid recursive definition has:
- **base case**(s) and
- **recursive case**(s) (with "smaller" self-reference)

Natural Numbers Are Recursively Defined

A **Natural Number** is:

Base Case
- **0**

Recursive Case
- Or $k + 1$, where k is a **Natural Number**

Self-reference

Recursive definition is valid because self-reference is "smaller"

So, **proving** things about:
recursive Natural Numbers requires
recursive proof,
i.e., **proof by induction!**

A valid recursive definition has:
- **base case** and
- **recursive case** (with "smaller" self-reference)

Proof By Induction Example (Sipser Ch 0)

Prove true: $P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right)$

- P_t = loan balance after t months
- t = # months
- P = principal = original amount of loan
- M = interest (multiplier)
- Y = monthly payment

(Details of these variables not too important here)

Proof By Induction Example (Sipser Ch 0)

Prove true: $P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right)$

Proof: by **induction** on natural number t

A proof by induction follows the cases of the recursive definition (here, natural numbers) that the induction is "on"

Base Case, $t = 0$:

A Natural Number is:

- 0 ☑
- Or $k + 1$, where k is a natural number

$$P_0 = PM^0 - Y\left(\frac{M^0 - 1}{M - 1}\right) = P$$

Plug in $t = 0$

Simplify

$P_0 = P$ is a true statement ! (amount owed at start = loan amount)

Proof By Induction Example (Sipser Ch 0)

Prove true: $P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right)$

A **proof by induction** follows cases of recursive definition (here, **natural numbers**) that the **induction is "on"**

A Natural Number is:
- 0 ☑
- $k + 1$, for some **nat num** k

Inductive Case: $t = k + 1$, for some **natural num** k

- **Inductive Hypothesis** (IH), **assume statement is true** for some $t =$ (smaller) k

IH plugs in "smaller" k

$$P_k = PM^k - Y\left(\frac{M^k - 1}{M - 1}\right)$$

- Goal **statement to prove,** for $t = k+1$:

$$P_{k+1} = PM^{k+1} - Y\left(\frac{M^{k+1} - 1}{M - 1}\right)$$

Write $t = k+1$ case in terms of "smaller" k

Plug in IH for P_k

Simplify, to get to goal statement

- Proof of Goal:

$$P_{k+1} = P_k M - Y$$

Definition of Loan:
amt owed in month $k+1$ =
amt owed in month k * interest M – amt paid Y

In-class Exercise: Proof By Induction

A **proof by induction** follows cases of recursive definition (here, **natural numbers**) that the **induction is "on"**

A Natural Number is:

- 0
- $k + 1$, for some **nat num** k

Prove: $(z \neq 1)$

$$\sum_{i=0}^{m} z^i = \frac{1 - z^{m+1}}{1 - z}$$

Use **Proof by Induction**.

Make sure to clearly state what (number) the induction is "on"

Proof by Induction: CS 420 Example

Statement to prove: $\text{LANGOF}(G) = \text{LANGOF}(R = \textbf{GNFA→RegExpr}(G))$

- Where:
  - G = a GNFA
  - R = a Regular Expression **GNFA→RegExpr**(G)

- i.e., **GNFA→RegExpr** must not change the language!

This time, let's **really prove equivalence!** (we previously "proved" it with some examples)

Proof by Induction: CS 420 Example

Statement to prove: $\text{LANGOF}(G) = \text{LANGOF}(\textbf{GNFA→RegExpr}(G))$

Recursively defined "thing"

Proof: by Induction on # of states in G

1. Prove *Statement* is true for base case

G has 2 states

Why is this an ok base case (instead of zero)?

(Modified) Recursive definition:

A "NatNumber > 1" is:

- 2
- Or $k + 1$, where k is a "NatNumber > 1"

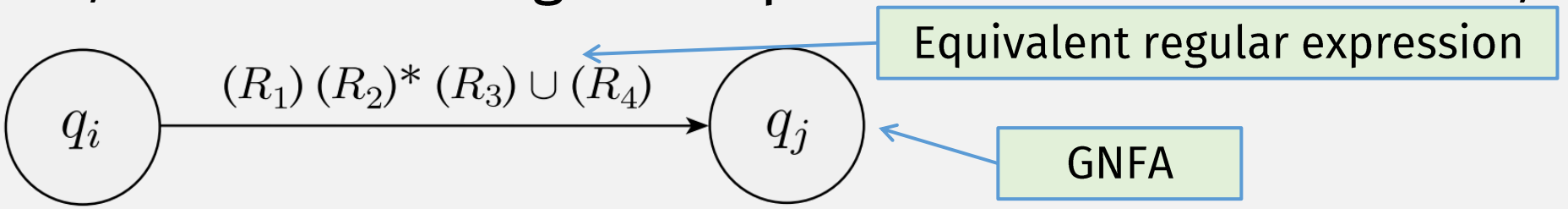
Last Time

GNFA→RegExpr (recursive) function

On GNFA input G:

Base Case

- If G has 2 states, **`return`** the regular expression (from the transition), e.g.:



Proof by Induction: CS 420 Example

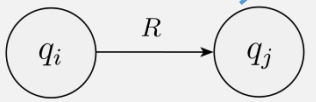
Statement to prove: LANGOF (G) = LANGOF (**GNFA→RegExpr**(G))

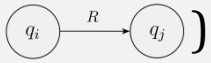
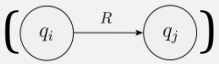
Plug in

Proof: by Induction on # of states in G

☑ 1. Prove *Statement* is true for base case

G has 2 states



Statements	**Justifications**
1. LANGOF () = LANGOF (R)	1. Definition of GNFA
2. **GNFA→RegExpr**() = R	2. Definition of **GNFA→RegExpr** (base case)
Goal: LANGOF () = LANGOF (**GNFA→RegExpr**())	3. From (1) and (2)

Plug in R

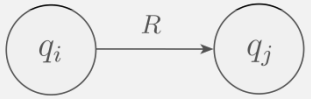
Don't forget the Statements / Justifications !

Proof by Induction: CS 420 Example

Statement to prove: LANGOF (G) = LANGOF (**GNFA→RegExpr**(G))

Proof: by Induction on # of states in G

☑ 1. Prove *Statement* is true for base case — G has 2 states



2. Prove *Statement* is true for recursive case: G has > 2 states

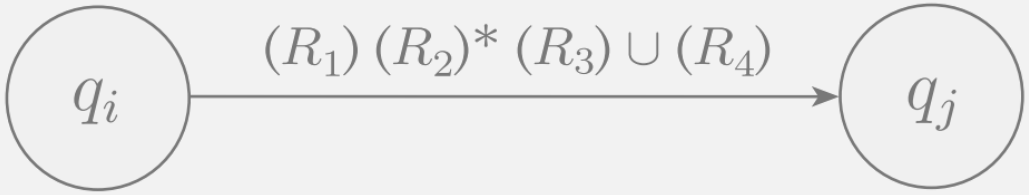
Last Time

GNFA→RegExpr (recursive) function

On GNFA input G:

Base Case

- If G has 2 states, **return** the regular expression (from the transition), e.g.:



- Else:

Recursive Case

- "Rip out" one state
- "Repair" the machine to get an equivalent GNFA G'
- Recursively call **GNFA→RegExpr**(G')

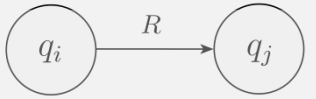
Recursive call (with a "smaller" G')

Proof by Induction: CS 420 Example

Statement to prove: LANGOF (G) = LANGOF (**GNFA→RegExpr**(G))

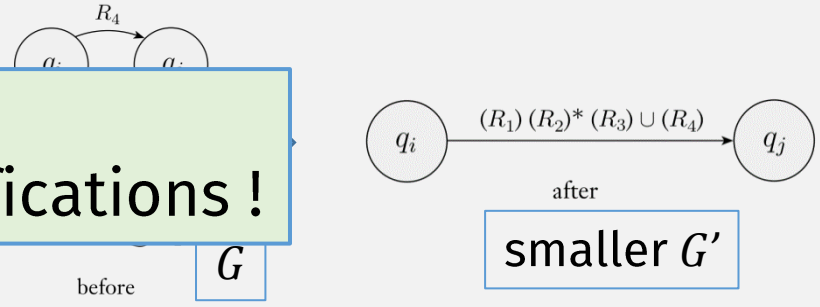
Proof: by Induction on # of states in G

☑ 1. Prove *Statement* is true for base case — G has 2 states



2. Prove *Statement* is true for recursive case: G has > 2 states
   - Assume the **induction hypothesis** (IH):
     - *Statement* is true for smaller G'
   - Use it to prove *Statement* is true for G > 2 states
     - Show that going from G to smaller G' is true!

LANGOF (G') = LANGOF (**GNFA→RegExpr**(G'))
(Where G' has less states than G)



Don't forget the Statements / Justifications !

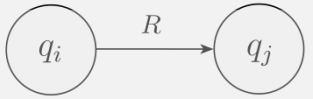
Show that "rip/repair" step ☑ converts G to smaller, equivalent G'

Proof by Induction: CS 420 Example

Statement to prove: LANGOF (G) = LANGOF (**GNFA→RegExpr**(G))

Proof: by Induction on # of states in G

☑ 1. Prove *Statement* is true for base case — G has 2 states



☑ 2. Prove *Statement* is true for recursive case: G has > 2 states

- Assume the i
  - *Statement*
- Use it to prov tates
  - Show that e!

Known "facts" available to use:
- ☑IH
- ☑Equiv of Rip/Repair step
- ☑Def of GNFA->RegExpr

LANGOF (G')
=
LANGOF (**GNFA→RegExpr**(G'))
(Where G' has less states than G)

Statements	**Justifications**
1. LANGOF (G') = LANGOF (**GNFA→RegExpr**(G'))	1. IH
2. LANGOF (G) = LANGOF (G')	2. Equivalence of Rip/Repair step (prev)
3. **GNFA→RegExpr**(G) = **GNFA→RegExpr**(G') — Plug in	3. Def of **GNFA→RegExpr** (recursive call)
Goal 4. LANGOF (G) = LANGOF (**GNFA→RegExpr**(G))	4. From (1), (2), and (3)

Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a regular expr

☑ • Use **GNFA→RegExpr** to convert GNFA → equiv regular expression!

⇐ If a language is described by a regular expr, then it's regular

☑ • Convert regular expression → equiv NFA! ■

Now: we can use regular expressions to represent regular langs!

So we also have another way to prove things about regular languages!

So a regular language has these equivalent representations:
- DFA
- NFA
- Regular Expression

So Far: How to Prove A Language Is Regular?

Key step, either:

- Construct DFA
- Construct NFA
- Create Regular Expression

Slightly different because of recursive 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.

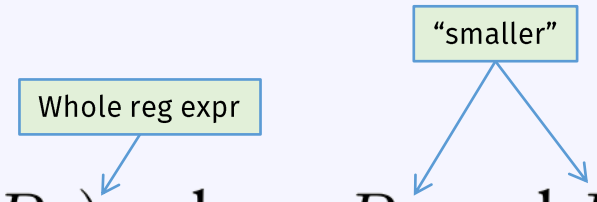
Proof by Induction

To Prove: a ***Statement*** about a recursively defined "thing" x:

1. Prove: *Statement* for base case of x
2. Prove: *Statement* for recursive case of x:
   - Assume: **induction hypothesis** (IH)

     I.e., *Statement* is true for some x_{smaller}
     - E.g., if x is number, then "smaller" = lesser number
     - E.g., if x is regular expression, then "smaller" = …
   - Prove: *Statement* for x, using IH (and known definitions, theorems …)
     - Usually, must show that going from x_{smaller} to larger x is true!

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.



Thm: Reverse is Closed for Regular Langs

Example string: $\mathbf{abc}^{\mathcal{R}} = \mathbf{cba}$

For any string $w = w_1 w_2 \cdots w_n$, the ***reverse*** of w, written $w^{\mathcal{R}}$, is the string w in reverse order, $w_n \cdots w_2 w_1$.

For any language A, let $A^{\mathcal{R}} = \{w^{\mathcal{R}} \mid w \in A\}$

Example language:

$\{\mathbf{a}, \mathbf{ab}, \mathbf{abc}\}^{\mathcal{R}} = \{\mathbf{a}, \mathbf{ba}, \mathbf{cba}\}$

Theorem: if A is regular, so is $A^{\mathcal{R}}$

Proof: by induction on the regular expression of A

Thm: Reverse is Closed for Regular Langs

if A is regular, so is $A^{\mathcal{R}}$

Proof: by Induction on regular expression of A: (6 cases)

Base cases

1. a for some a in the alphabet Σ, — same reg. expr. represents $A^{\mathcal{R}}$ so it is regular
2. ε, — same reg. expr. represents $A^{\mathcal{R}}$ so it is regular
3. $\emptyset$, — same reg. expr. represents $A^{\mathcal{R}}$ so it is regular

Inductive cases

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.

Need to Prove: if A is a regular language, described by reg expr $R_1 \cup R_2$, then $A^{\mathcal{R}}$ is regular

IH1: if A_1 is a regular language, described by reg expr R_1, then $A_1{}^{\mathcal{R}}$ is regular

IH1: if A_2 is a regular language, described by reg expr R_2, then $A_2{}^{\mathcal{R}}$ is regular

"smaller"

Thm: Reverse is Closed for Regular Langs

if A is regular, so is $A^{\mathcal{R}}$

Proof: by Induction on regular expression of A: (Case # 4)

	Statements	**Justifications**
	1. Language A is regular, with reg expr $R_1 \cup R_2$	1. Assumption of IF in IF-THEN
	2. R_1 and R_2 are regular expressions	2. Def of Regular Expression
	3. R_1 and R_2 describe regular langs A_1 and A_2	3. Reg Expr ⇔ Reg Lang (Prev Thm)
	4. If A_1 is a regular language, then $A_1^{\mathcal{R}}$ is regular	4. IH
	5. If A_2 is a regular language, then $A_2^{\mathcal{R}}$ is regular	5. IH
	6. $A_1^{\mathcal{R}}$ and $A_2^{\mathcal{R}}$ are regular	6. By (3), (4), and (5)
	7. $A_1^{\mathcal{R}} \cup A_2^{\mathcal{R}}$ is regular	7. Union Closed for Reg Langs
	8. $A_1^{\mathcal{R}} \cup A_2^{\mathcal{R}} = (A_1 \cup A_2)^{\mathcal{R}}$	8. Reverse and Union Ops Commute
	9. $A = A_1 \cup A_2$	9. By (1), (2), and (3)
Goal	10. $A^{\mathcal{R}}$ is regular	10. By (7), (8), (9)

Thm: Reverse is Closed for Regular Langs

if A is regular, so is $A^{\mathcal{R}}$

Proof: by Induction on regular expression of A: (6 cases)

Base cases

Inductive cases

☑ **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.

Remaining cases will use similar reasoning

Next Time

Non-Regular Languages?

- Are there **languages** that are **not regular languages?**

- How can we **prove** that a **language** is **not a regular language?**