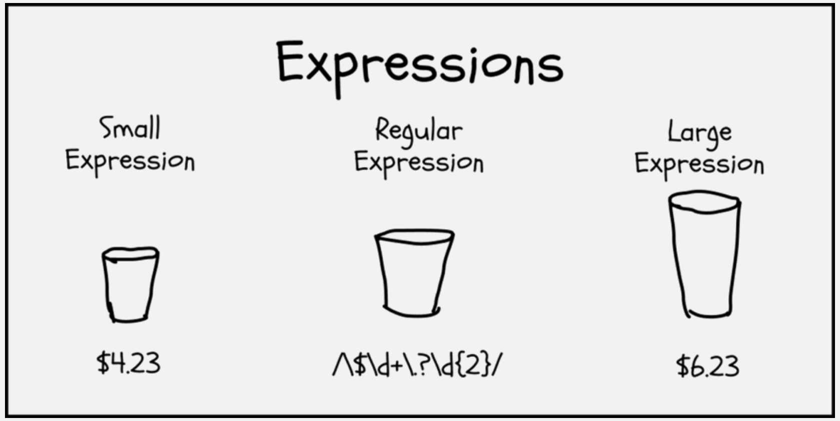
UMB CS 420

Regular Expressions

Thursday, September 29, 2022



Announcements

- HW 2
  - due Sunday 10/2 11:59pm EST

Last Time: A DFA's Language

- Let **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**

Last Time: An NFA's Language

- Let **NFA** $N = (Q, \Sigma, \delta, q_0, F)$

- N **accepts** w if $\hat{\delta}(q_0, w) \cap F \neq \emptyset$
  - i.e., computation ends in at least one accept state

- N **recognizes** language $\left\{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$

An NFA's language is a ___regular?___ language?

Last Time: Concatenation Closed for Reg Langs?

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

  … produces an NFA

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

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

Specifically, we must prove:

NFAs ⇔ regular languages

How to Prove a Statement: $X \Leftrightarrow Y$

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

Proof <u>at minimum</u> has 2 required parts:

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

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

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

(Easier)

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

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

(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**?

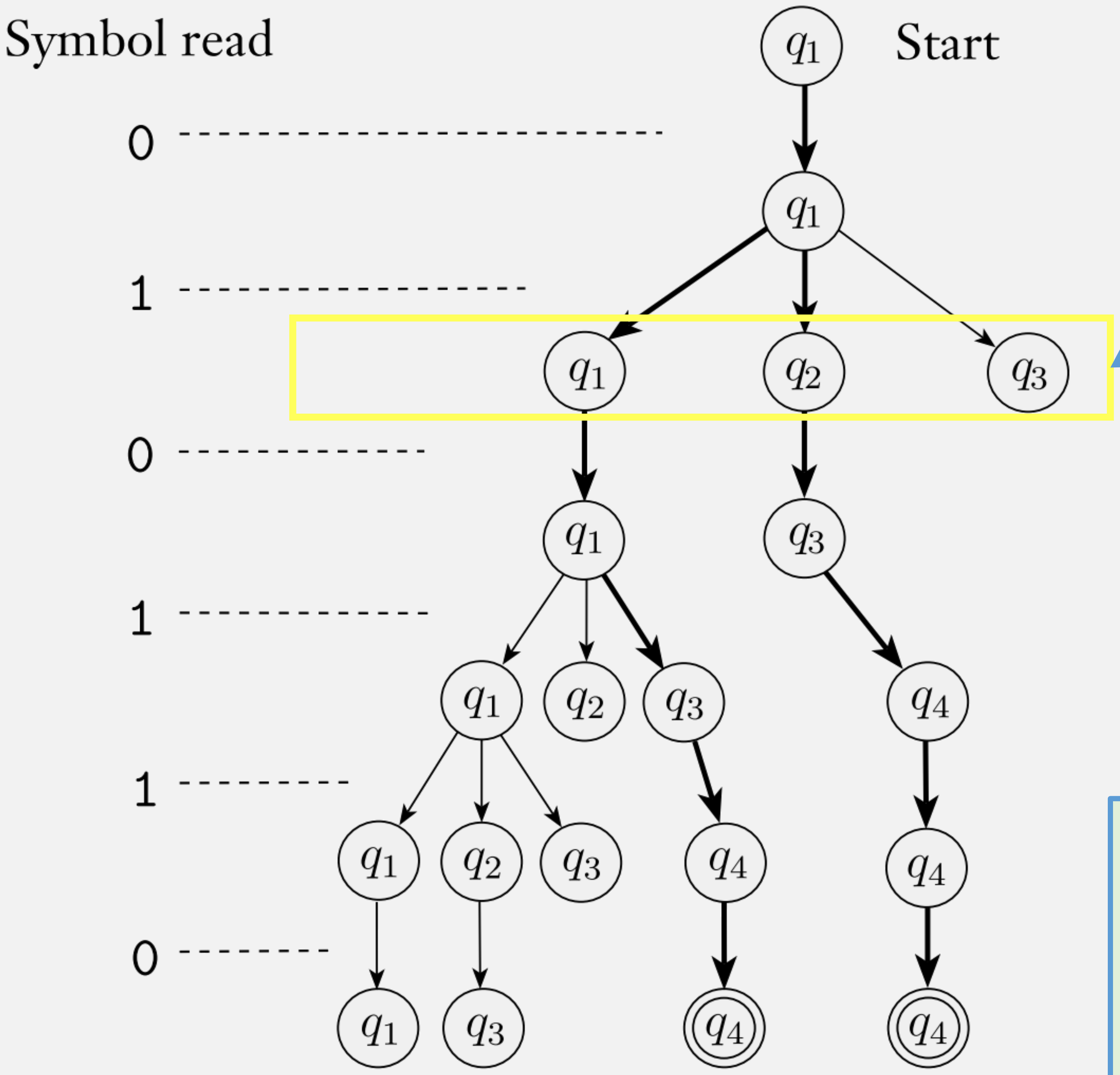
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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 multiple states

DFA computation can only be in one state

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

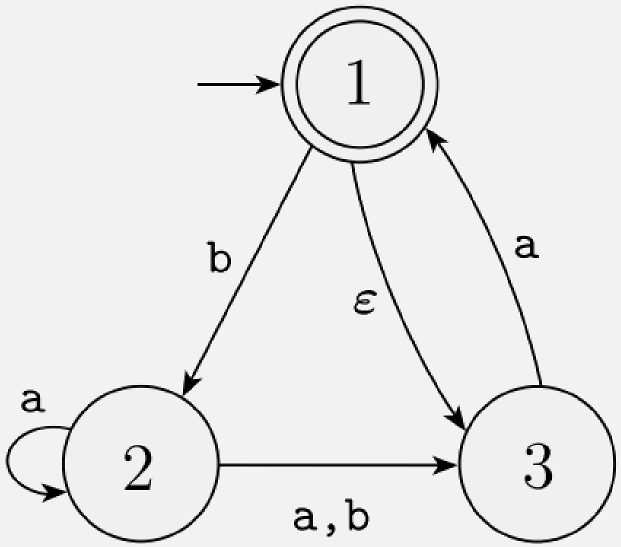
This is similar to the proof strategy from
“Closure of union” where:
a state = a pair of states

Convert **NFA→DFA**, Formally

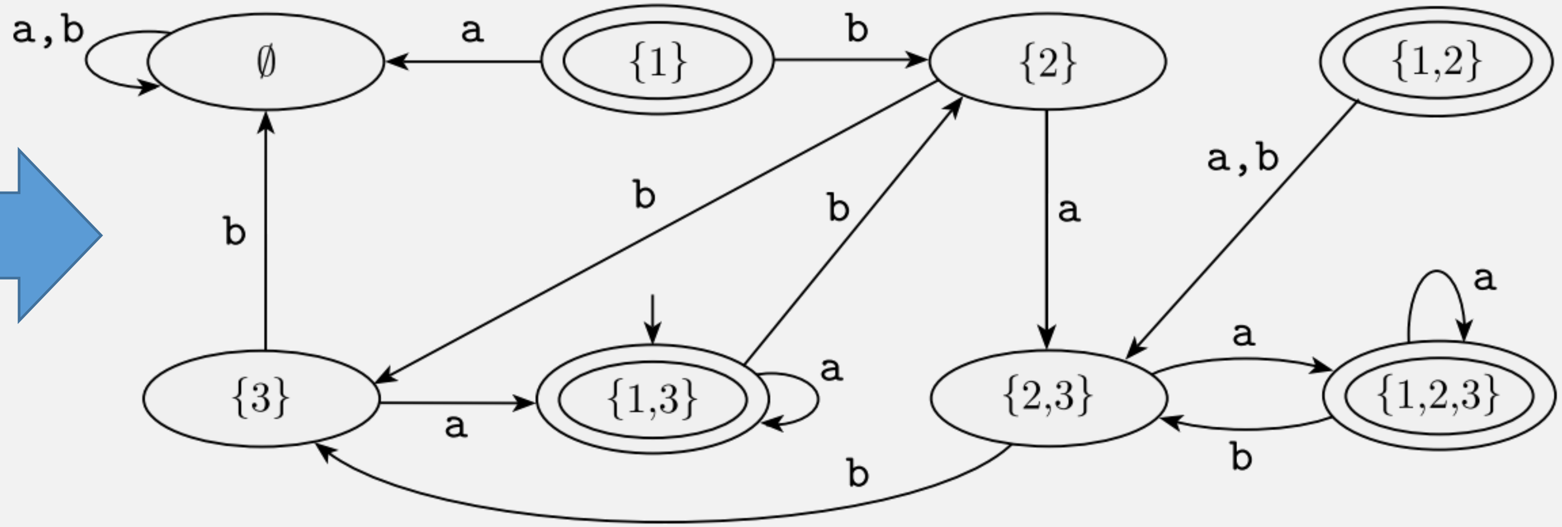
- Let **NFA** N = $(Q, \Sigma, \delta, q_0, F)$

- **An equivalent** DFA M **has states** Q' = $\mathcal{P}(Q)$ (power set of Q)

Example:



The NFA N_4



A DFA D that is equivalent to the NFA N_4

No empty transitions

NFA→DFA

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

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

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

A DFA state = a set of NFA states

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

R = a DFA state = a set of NFA states

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

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

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

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

Flashback: Adding Empty Transitions

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

(Defined recursively)

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

- Inductive case:

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

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

Almost the same, except ...

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

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

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

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

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

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

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

(Easier)

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

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

(Harder)

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

… using our NFA to DFA algorithm! $\blacksquare$

Concatenation is Closed for Regular Langs

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.

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

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

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

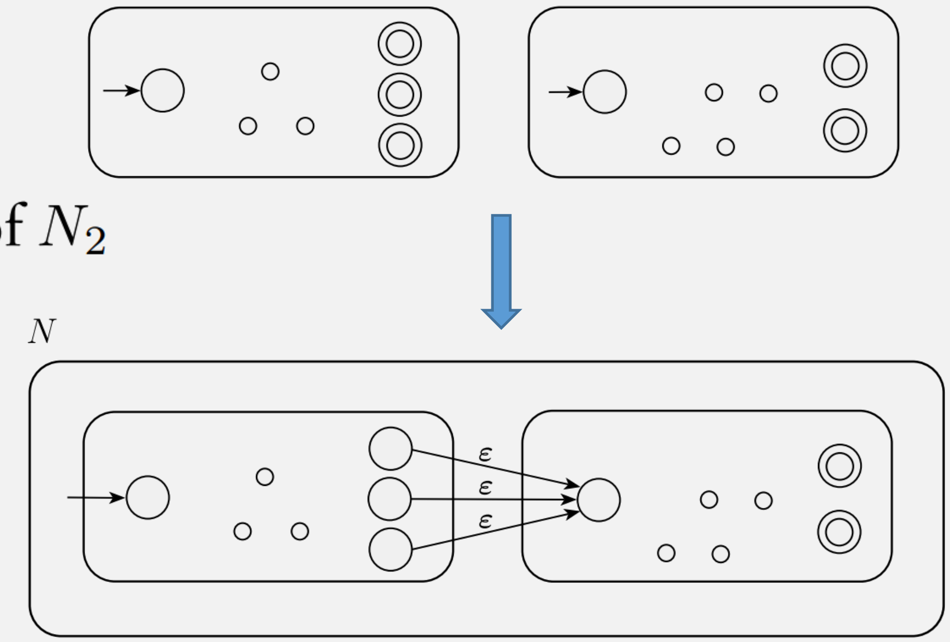
1. $Q = Q_1 \cup Q_2$

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 $a \in \Sigma_\varepsilon$,

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



☑ ??? ■

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

Flashback: Union is Closed For Regular Langs

THEOREM

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

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

Proof:

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

Flashback: Union is Closed For Regular Langs

Proof

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

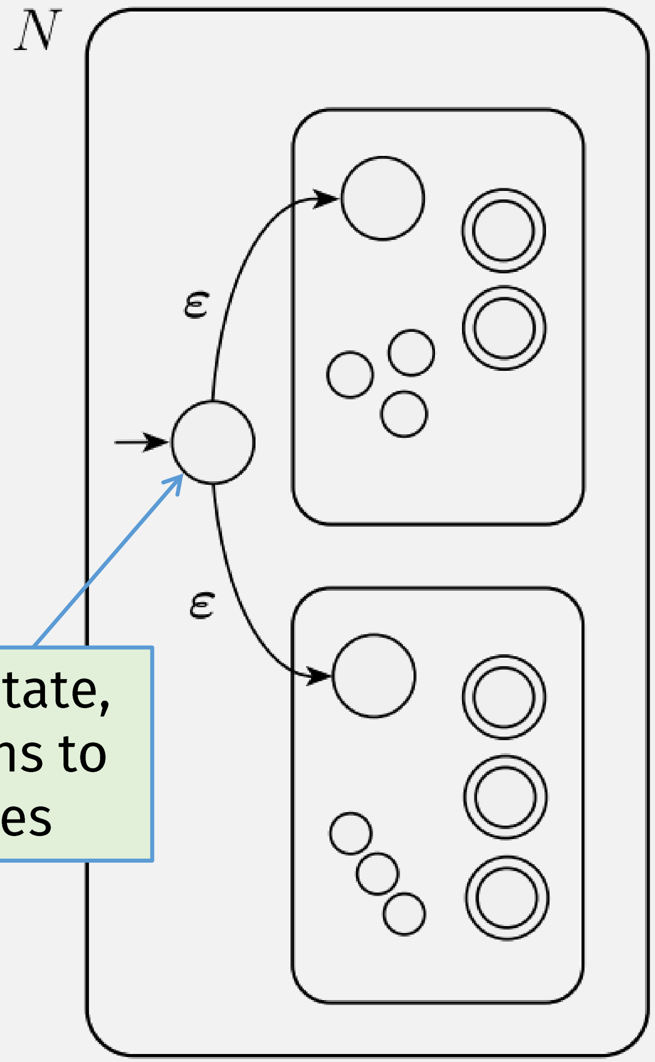
M step = a step in M_1 + a step in M_2

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

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

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

Union is Closed for Regular Languages



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

Union is Closed for Regular Languages

PROOF

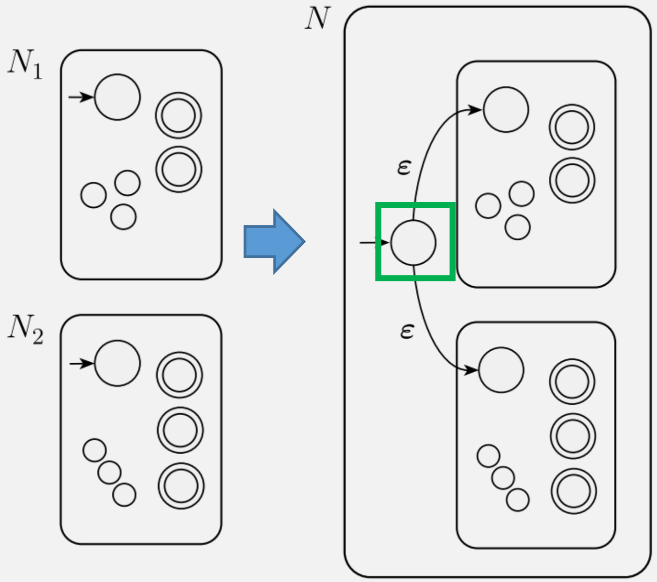
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

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

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

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

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



Union is Closed for Regular Languages

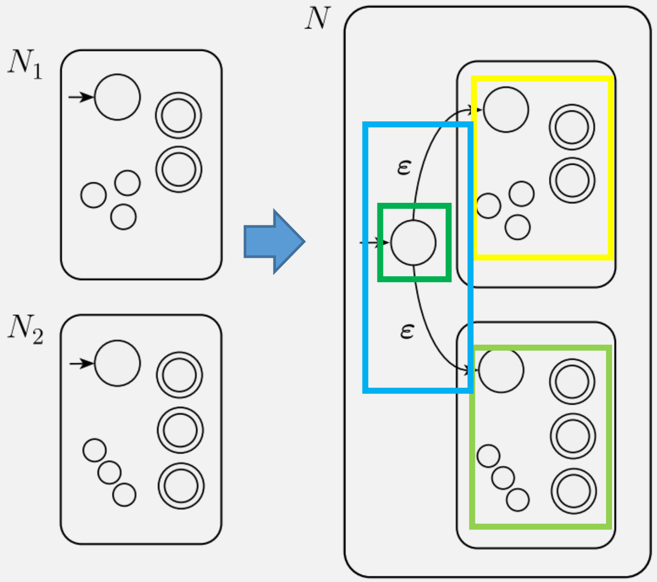
PROOF

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

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

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
2. The state q_0 is the start state of N.
3. The set of accept states $F = F_1 \cup F_2$.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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



List of Closed Ops for Reg Langs (so far)

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

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

Kleene Star Example

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

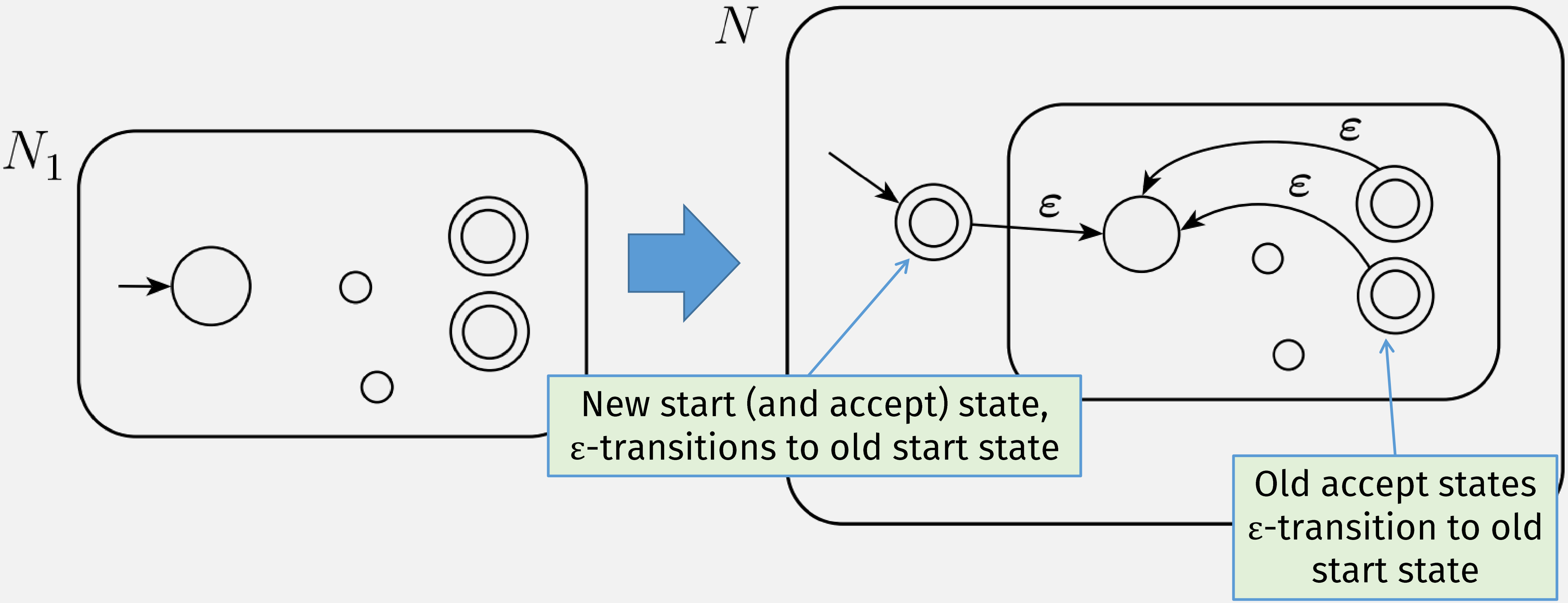
If $A = \{\mathtt{good}, \mathtt{bad}\}$ and $B = \{\mathtt{boy}, \mathtt{girl}\}$, then

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

Note: **repeat zero or more times**

(this is an infinite language!)

Kleene Star


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

In-class exercise:

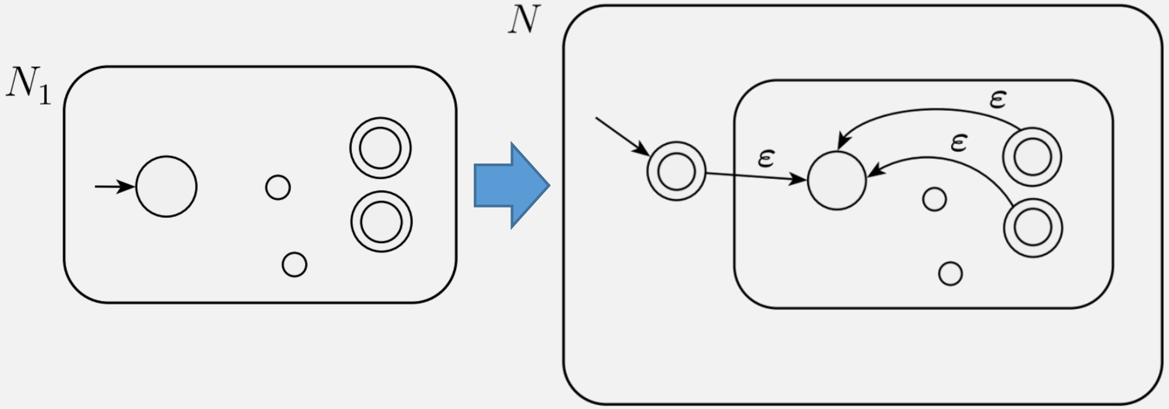
Kleene Star is Closed for Regular Langs

THEOREM

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

Kleene Star is Closed for Regular Langs

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

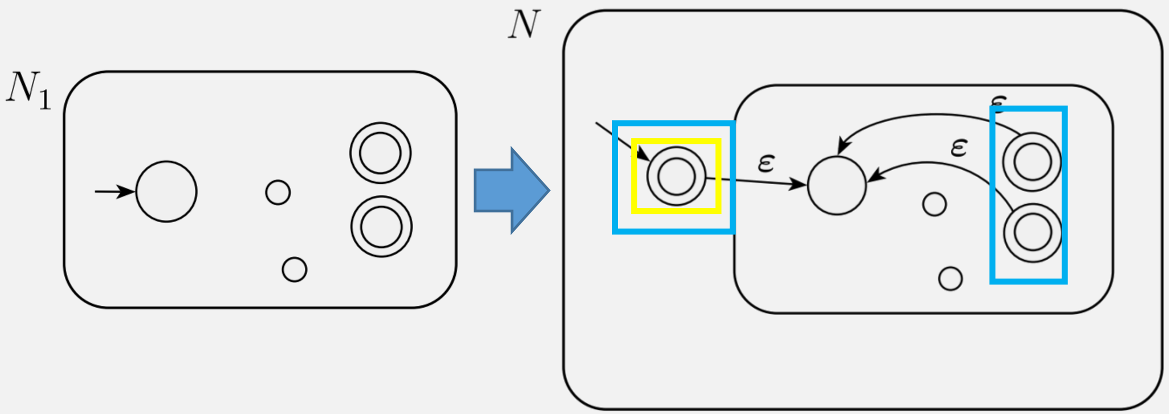


Kleene Star is Closed for Regular Langs

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

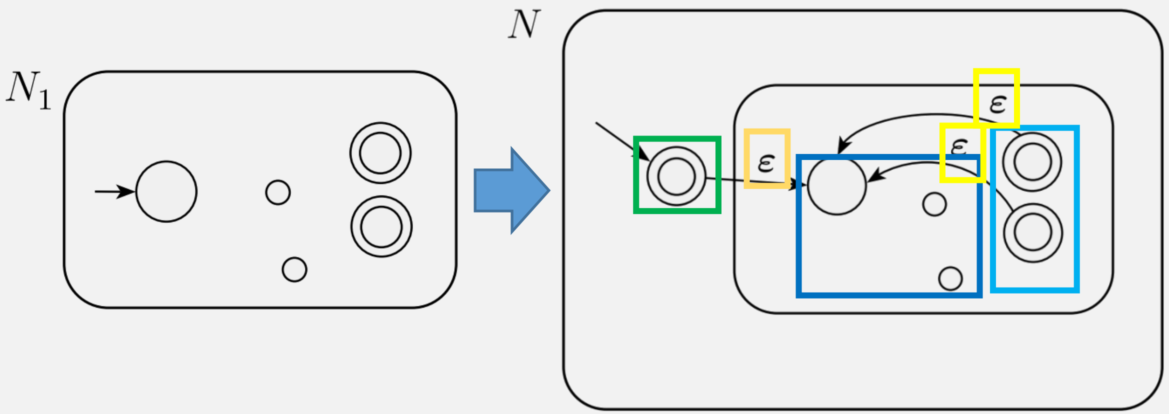
1. $Q = \{q_0\} \cup Q_1$
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$

Kleene star of a language must accept the empty string!



Kleene Star is Closed for Regular Langs

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



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

2. The state q_0 is the new start state.

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

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

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

Many More Closed Operations on Regular Languages!

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

Why do we care about these ops?

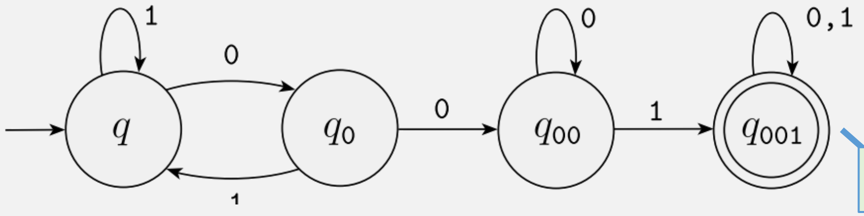
- Union
- Concat
- Kleene star

- The are sufficient to represent all regular languages!

- I.e., they define **regular expressions**

So Far: Regular Language Representations

1. State diagram (NFA/DFA)



(it doesn't fit!)

A practical application: **text search**

2. Formal description

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

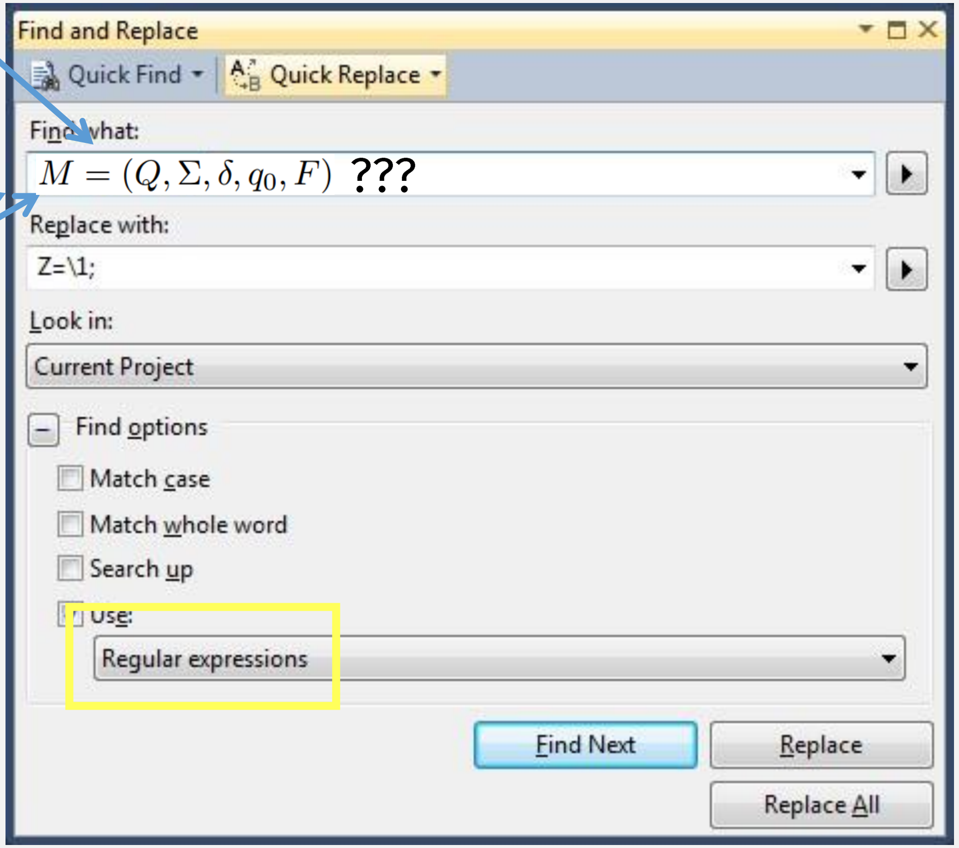
Analogy:
- All **regular languages** ~ a "programming language"
- One **regular language** ~ a **"program"** (e.g., find strings containing **001**)

q_2	q_3	q_2
q_3	q_2	q_2,

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

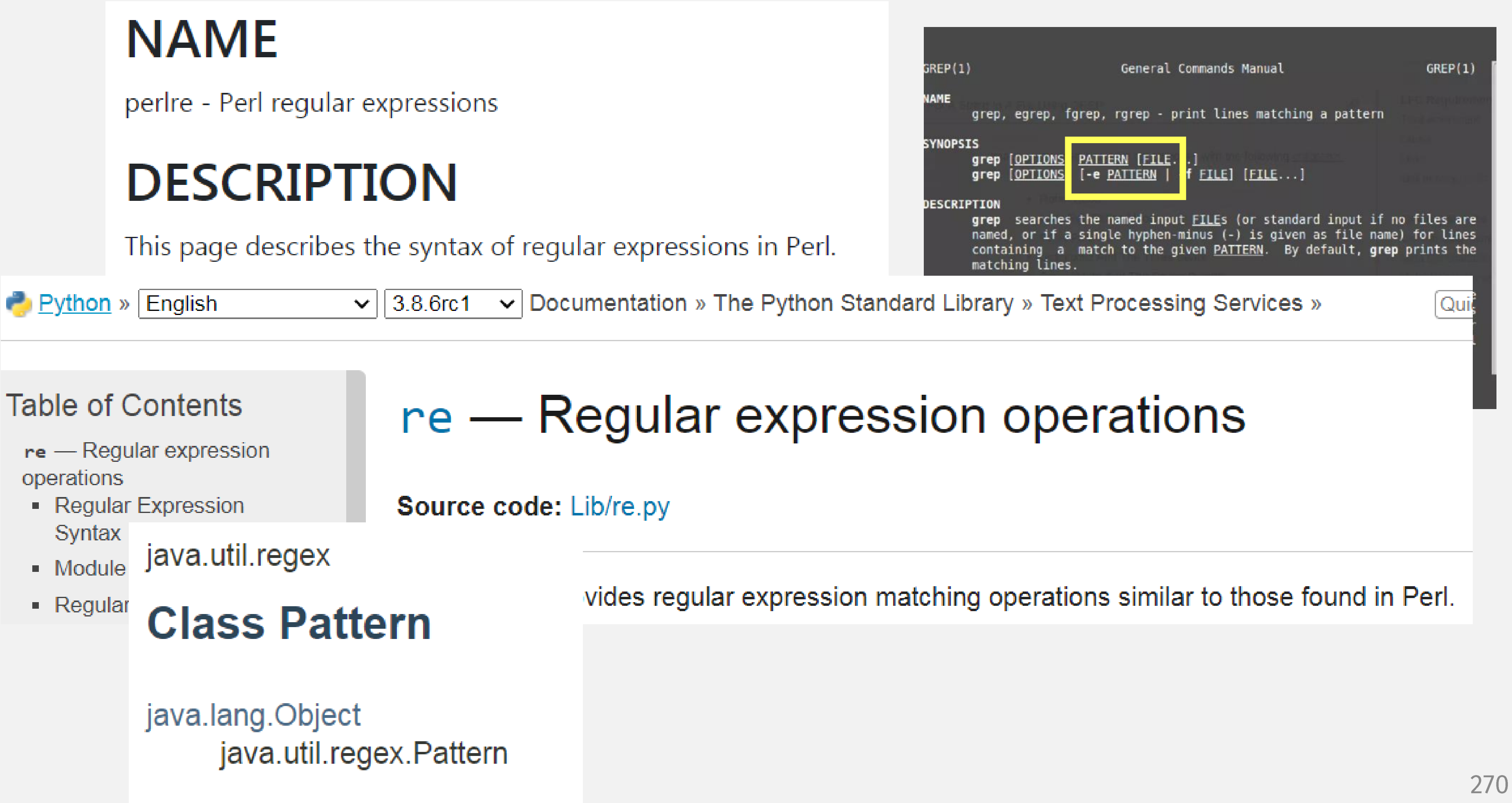
3. $\Sigma^*001\Sigma^*$

Need a more concise (textual) notation



Regular Expressions:
A Widely Used Programming Language
(inside other programming languages)

- Unix
- Perl
- Python
- Java



Why do we care about these ops?

- Union
- Concat
- Kleene star

- The are sufficient to represent all regular languages!

- I.e., they define **regular expressions**

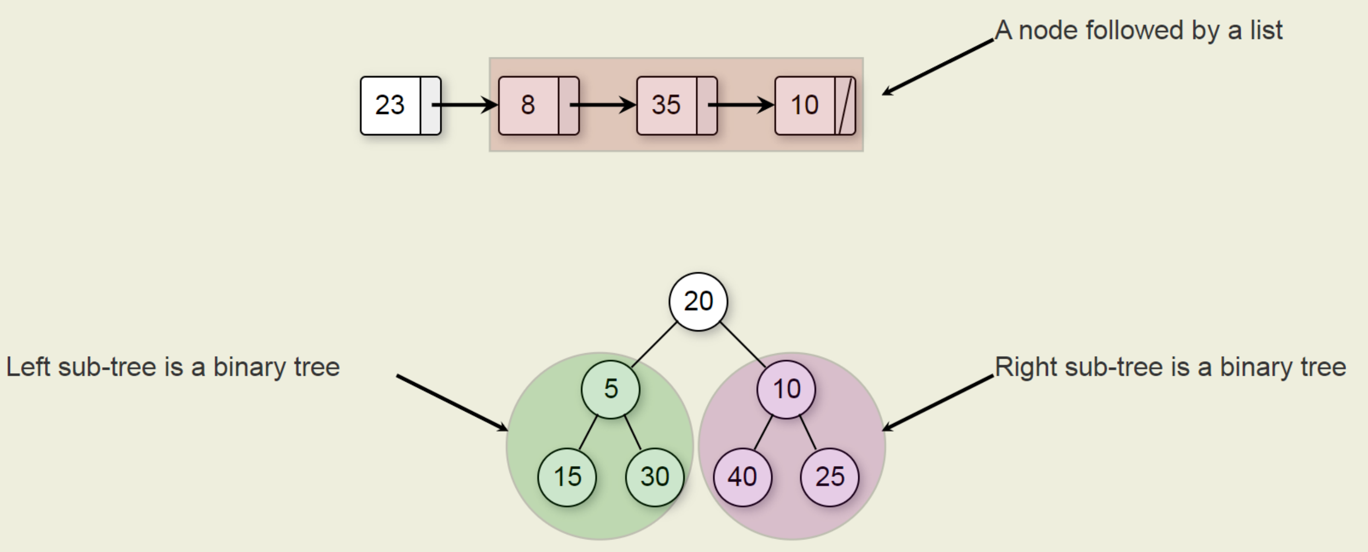
Regular Expressions: Formal Definition

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>

Recursive Definitions



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

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

This is a <u>recursive definition:</u>
Node used before it's defined
(but must be "smaller")

Regular Expressions: Formal Definition

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

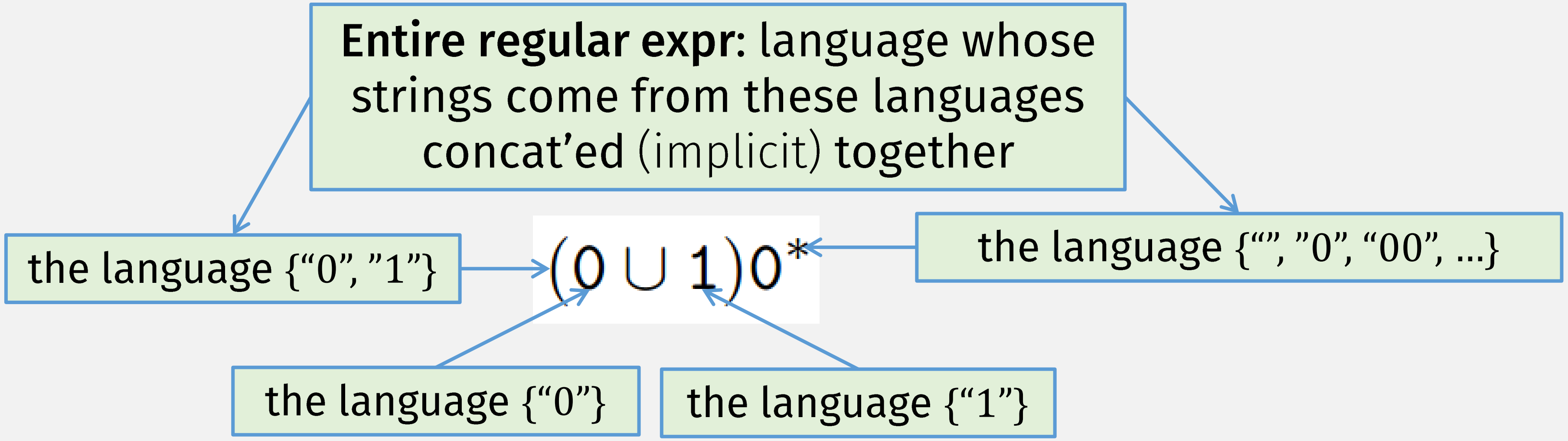
3 Base Cases

1. a for some a in the alphabet Σ, (A lang containing a) **length-1 string**
2. ε, (A lang containing) **the empty string**
3. $\emptyset$, **The empty set** (i.e., a lang containing no strings)

3 Recursive Cases

4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions, ← union
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or ← concat
6. (R_1^*), where R_1 is a regular expression. ← star

Regular Expression: Concrete Example



- Operator Precedence:
  - Parentheses
  - Kleene Star
  - Concat (sometimes ∘, 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.

Regular Expressions = Regular Langs?

3 Base Cases

3 Recursive Cases

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.

Base cases + union, concat, and Kleene star can express any regular language!

(But we have to prove it)

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

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

⇐ If **a language is described by a reg expression,** it **is regular**

(Easier)

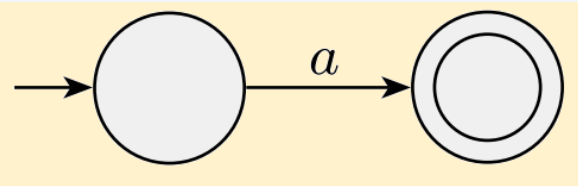
- To prove this part: convert reg expr → equivalent NFA!
- (Hint: we mostly did this already when discussing closed ops)

How to show that a language is regular?

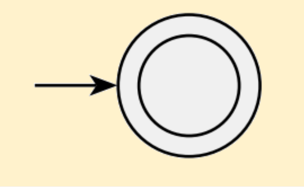
Construct a DFA *or* NFA!

RegExpr→NFA

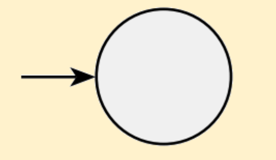
R is a ***regular expression*** if R is



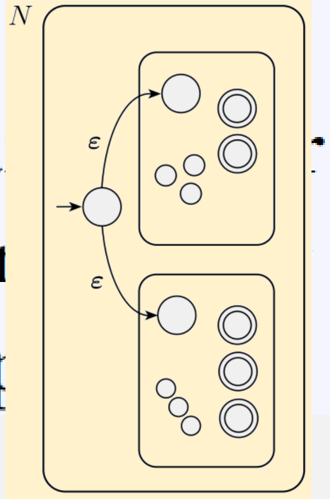
1. a for some a in the alphabet Σ,
2. ε,

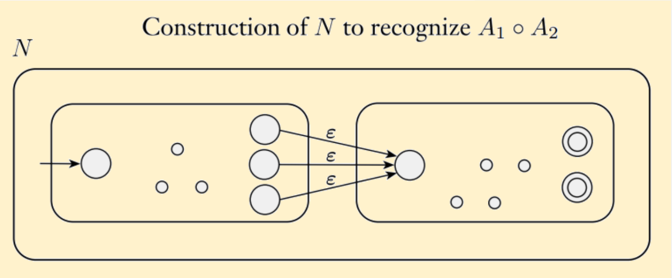


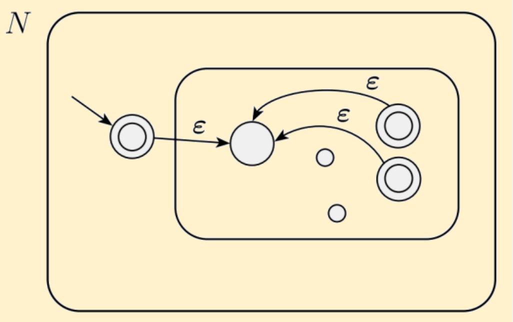
3. $\emptyset$,



4. $(R_1 \cup R_2)$, where R_1 and R_2 a... e
5. $(R_1 \circ R_2)$, where R_1 and R_2 ar... expressions, or
6. (R_1^*), where R_1 is a regular exp







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

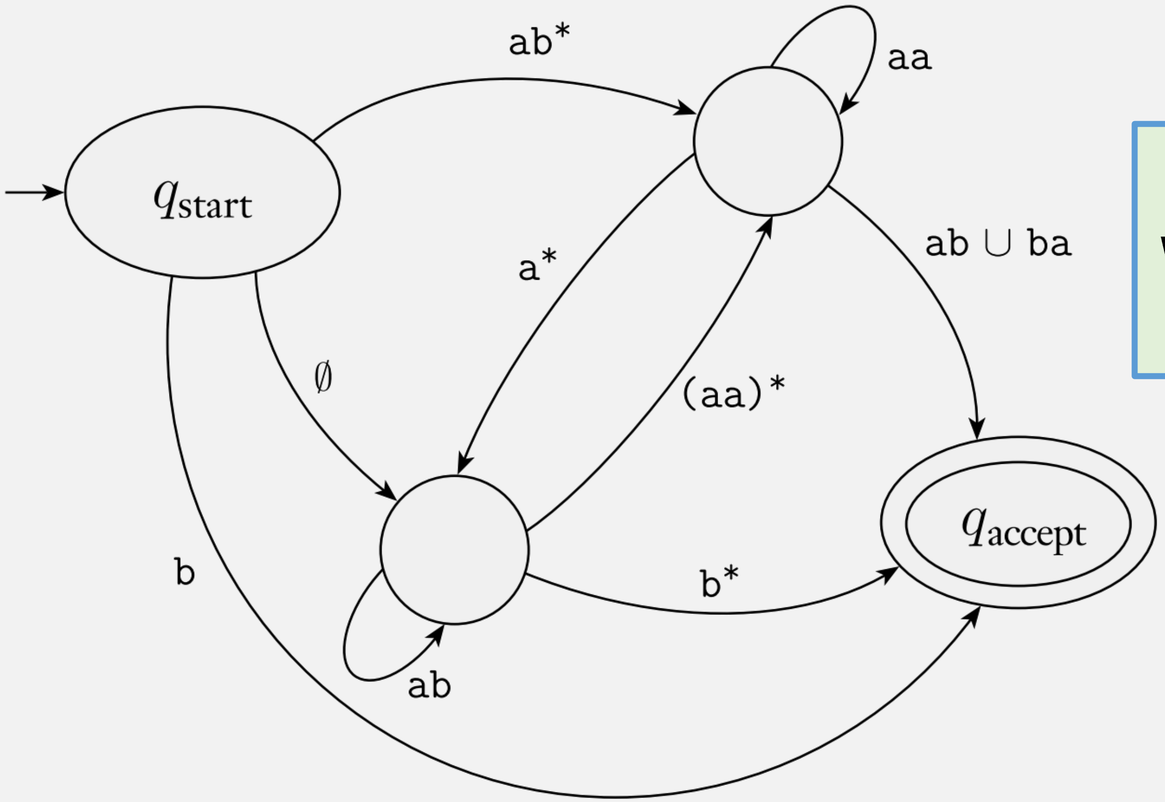
⇒ If **a language is regular,** it **is described by a reg expression**
(Harder)

- To prove this part: Convert an DFA or NFA → equivalent Regular Expression
- To do so, we first need another kind of finite automata: a **GNFA**

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

☑ • Convert the regular expression → an equivalent NFA!

Generalized NFAs (GNFAs)



A regular NFA is a GNFA with only single character regular expr transitions

Goal: **convert GNFAs to Regular Exprs**

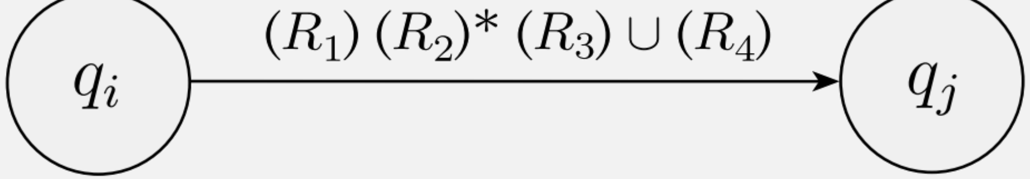
- GNFA = NFA with regular expression transitions

`GNFA→RegExpr` function

On GNFA input G:

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

Equivalent regular expression



GNFA

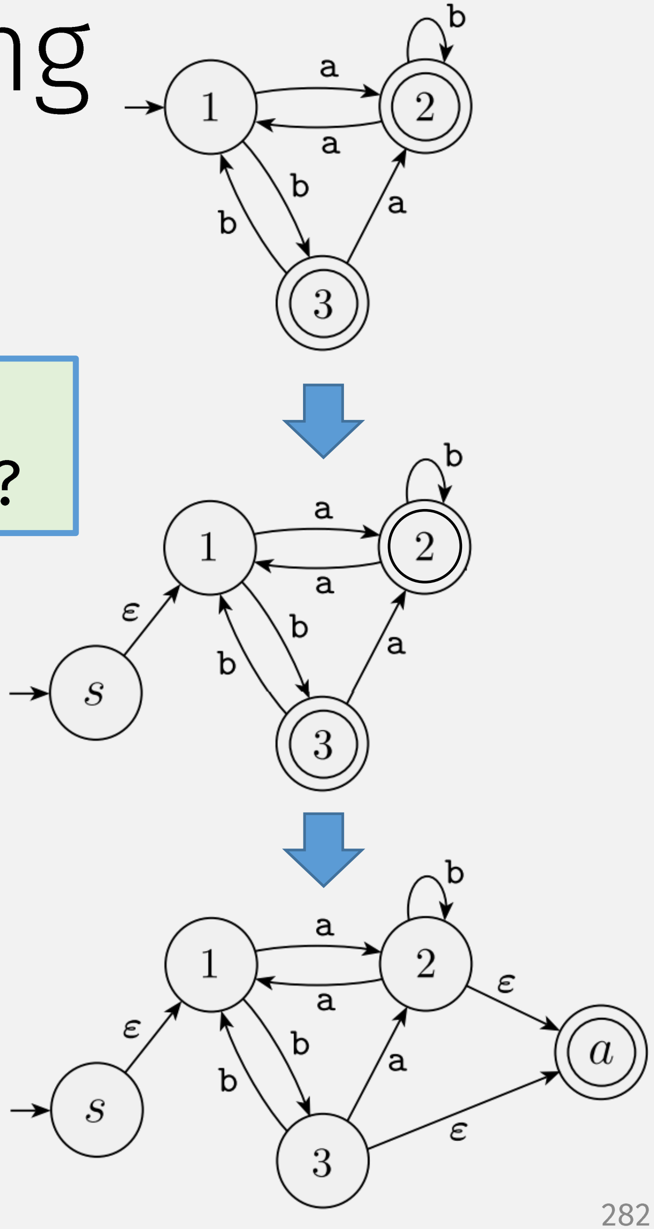
Could there be less than 2 states?

GNFA→RegExpr Preprocessing

- First, modify input machine to have:

> Does this change the language of the machine?

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

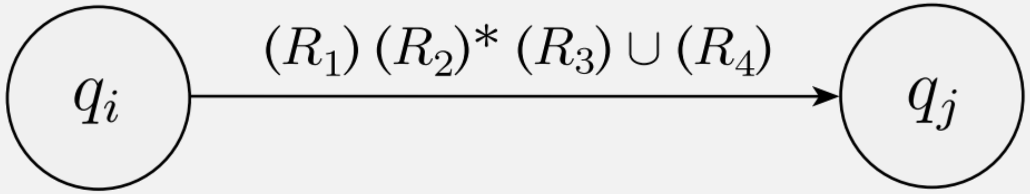


GNFA→RegExpr function (recursive)

On GNFA input G:

Base Case

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

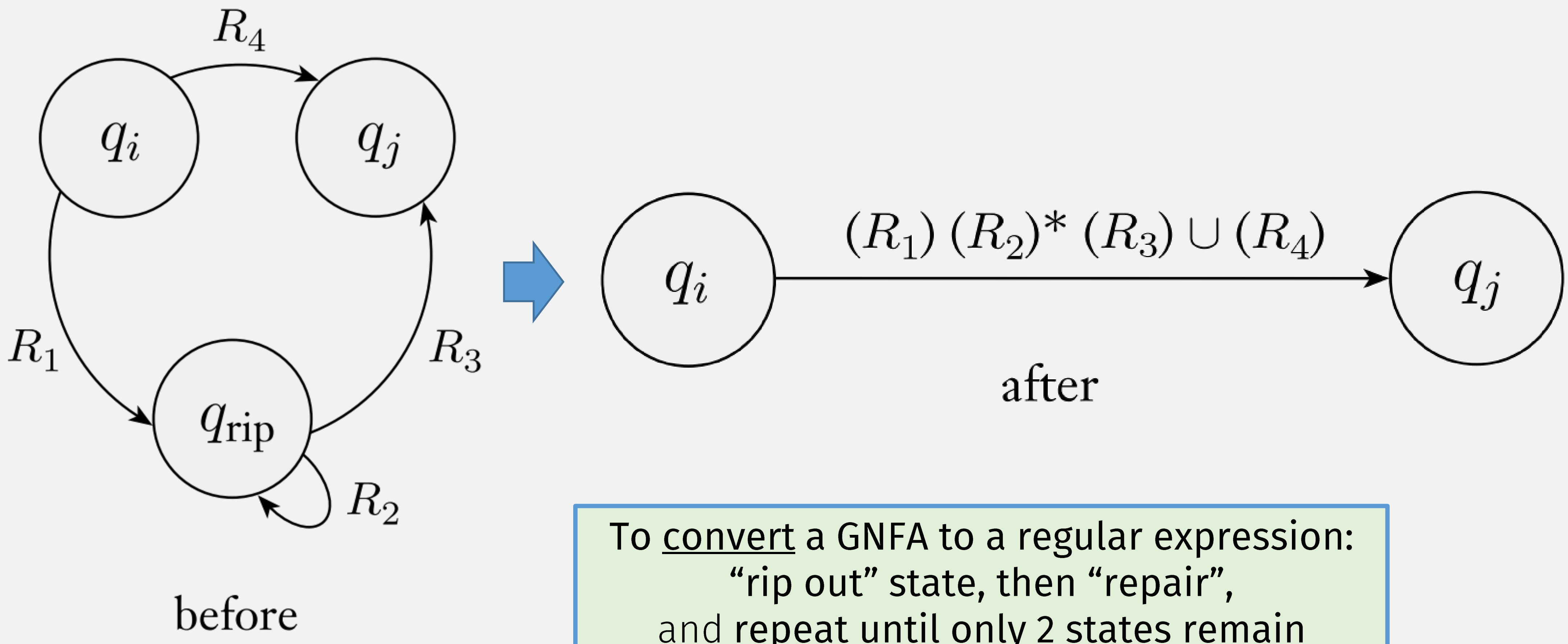


Recursive Case

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

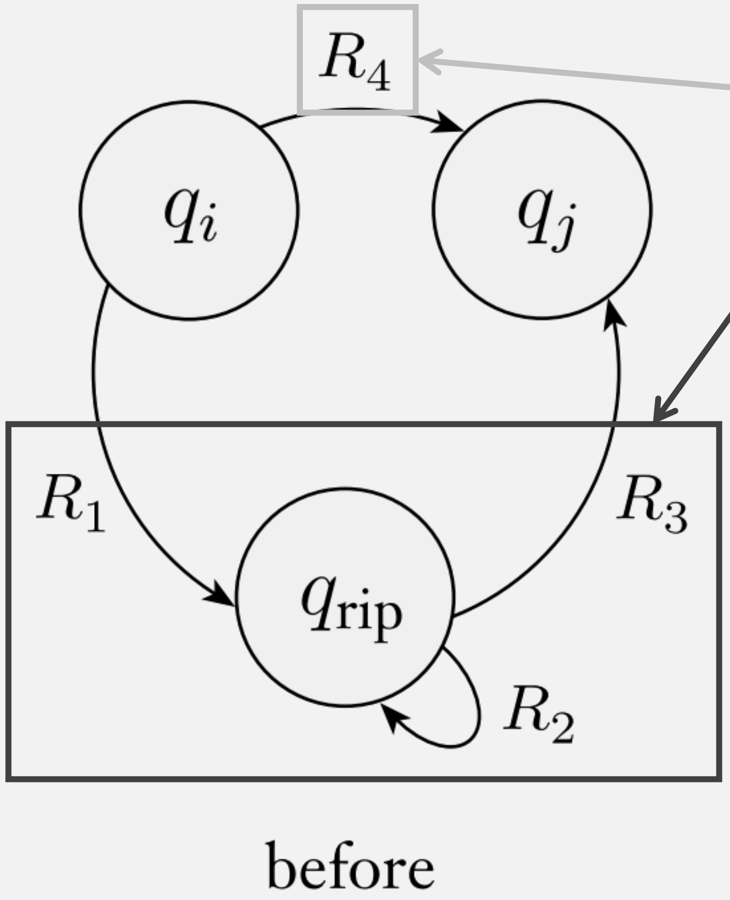
Recursive definitions have:
- base case and
- recursive case
  (with a "smaller" object)

GNFA→RegExpr: “Rip/Repair” step



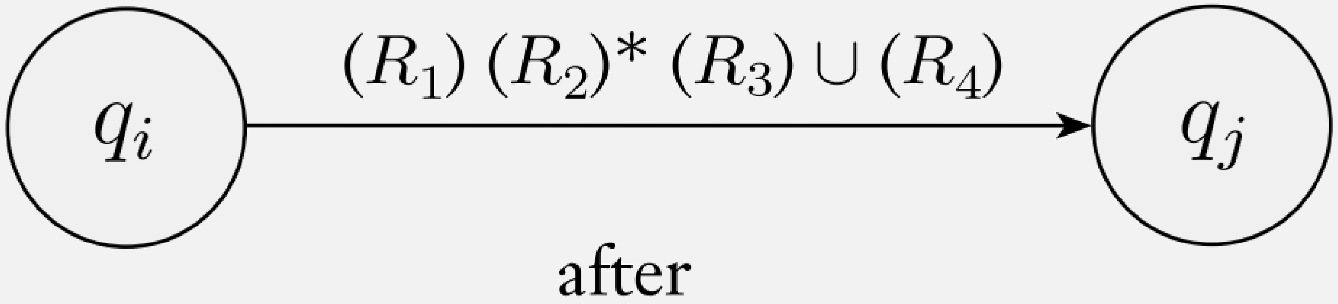
To convert a GNFA to a regular expression:
“rip out” state, then “repair”,
and repeat until only 2 states remain

GNFA→RegExpr: "Rip/Repair" step

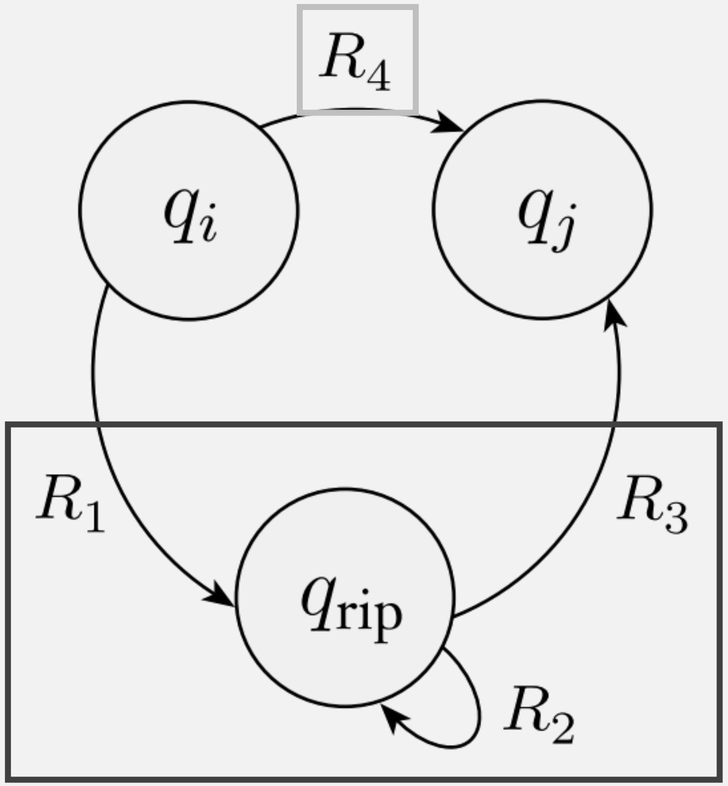


Before: two paths from q_i to q_j:

1. Not through q_{rip}
2. Through q_{rip}



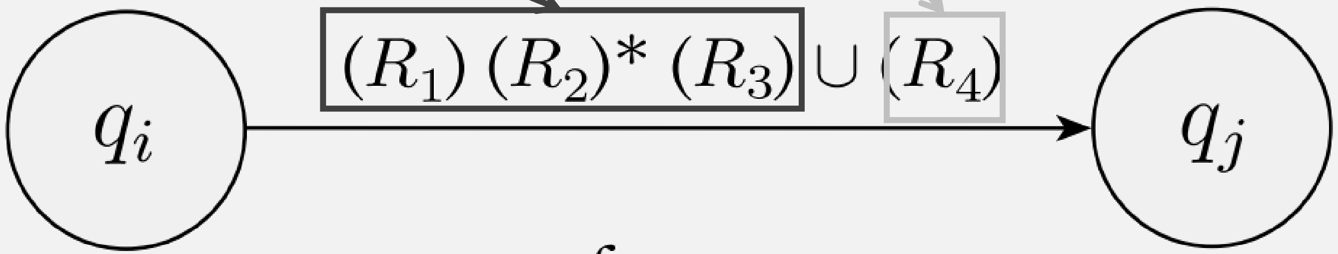
`GNFA→RegExpr`: "Rip/Repair" step



before

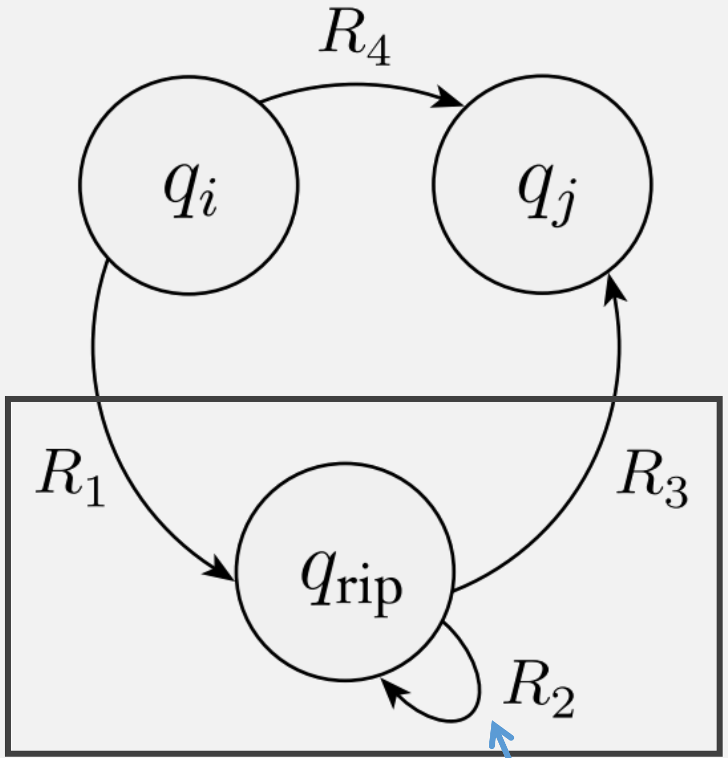
After: still two "paths" from q_i to q_j

1. Not through q_{rip}
2. Through q_{rip}

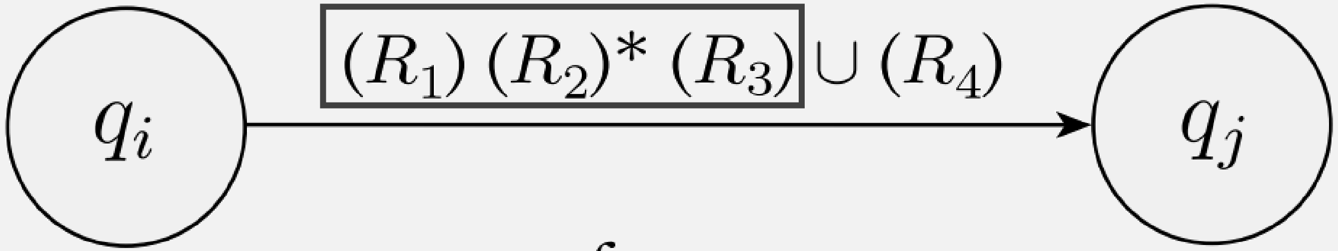


after

GNFA→RegExpr: "Rip/Repair" step



before

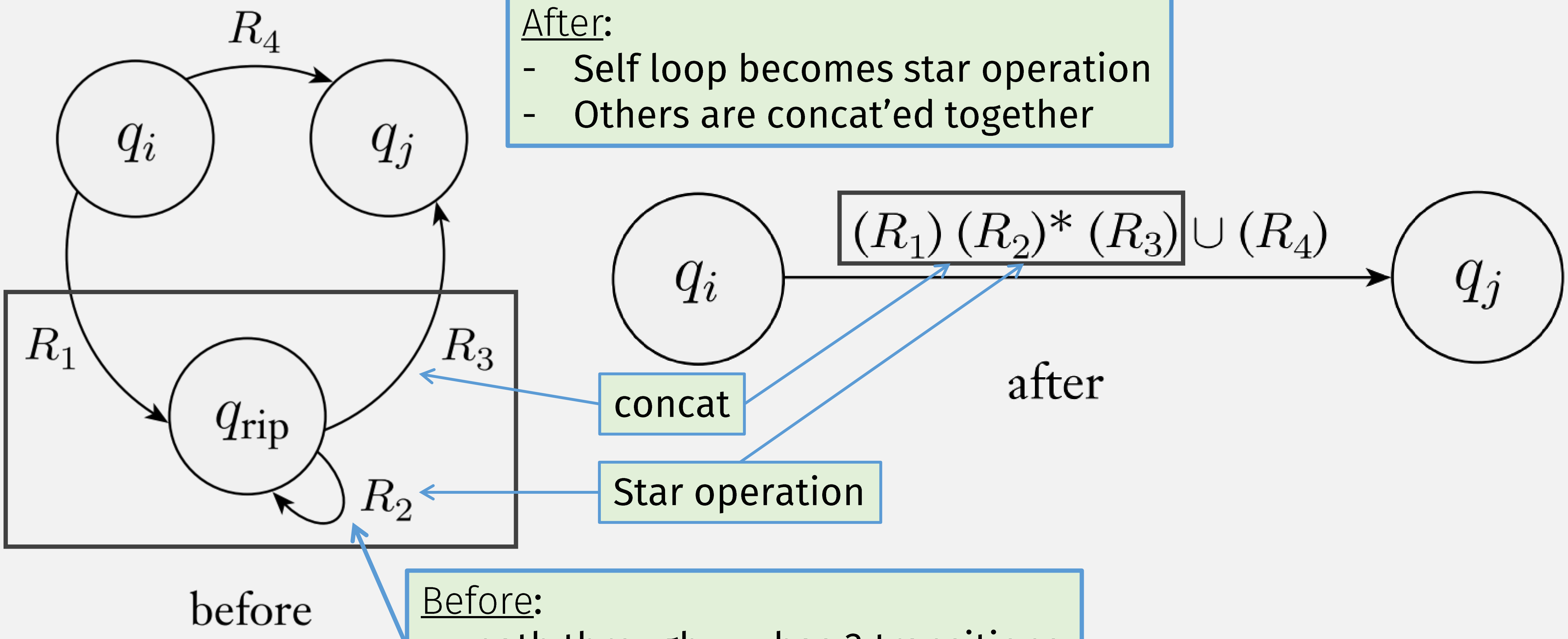


after

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

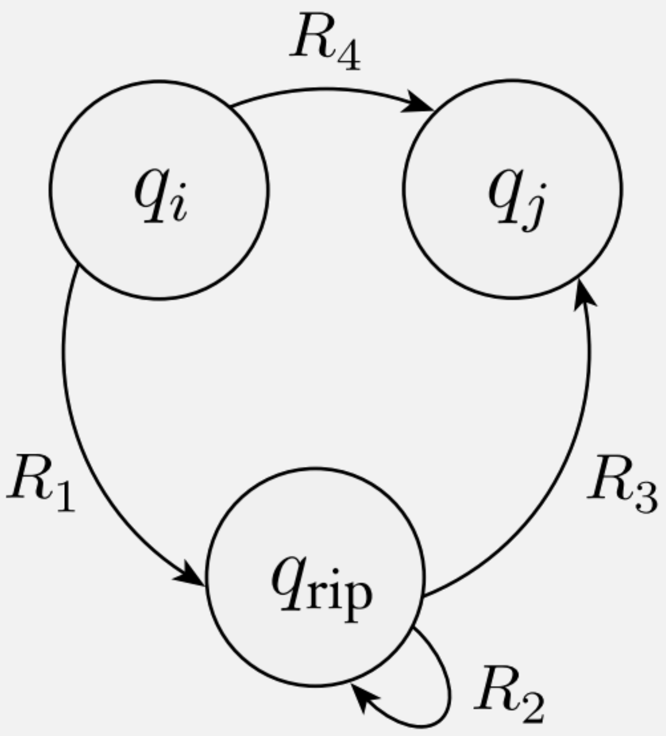
GNFA→RegExpr: "Rip/Repair" step

<u>After</u>:
- Self loop becomes star operation
- Others are concat'ed together

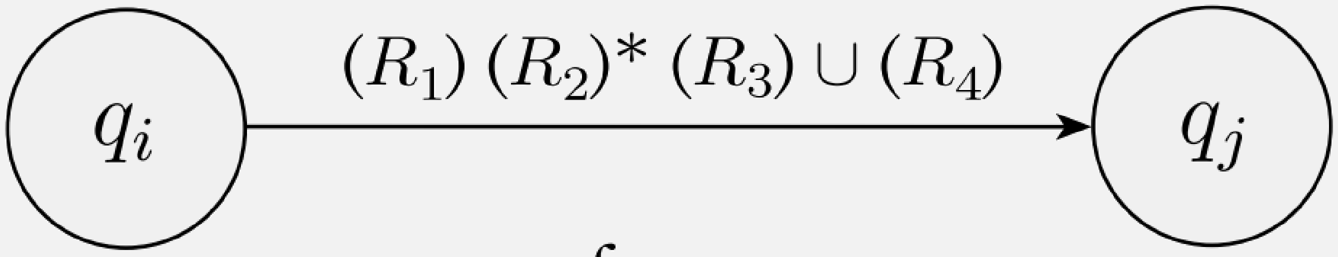


<u>Before</u>:
- path through q_{rip} has 3 transitions
- One is self loop

GNFA→RegExpr: Rip/Repair “Correctness”



before



after

Must show these are <u>equivalent</u>

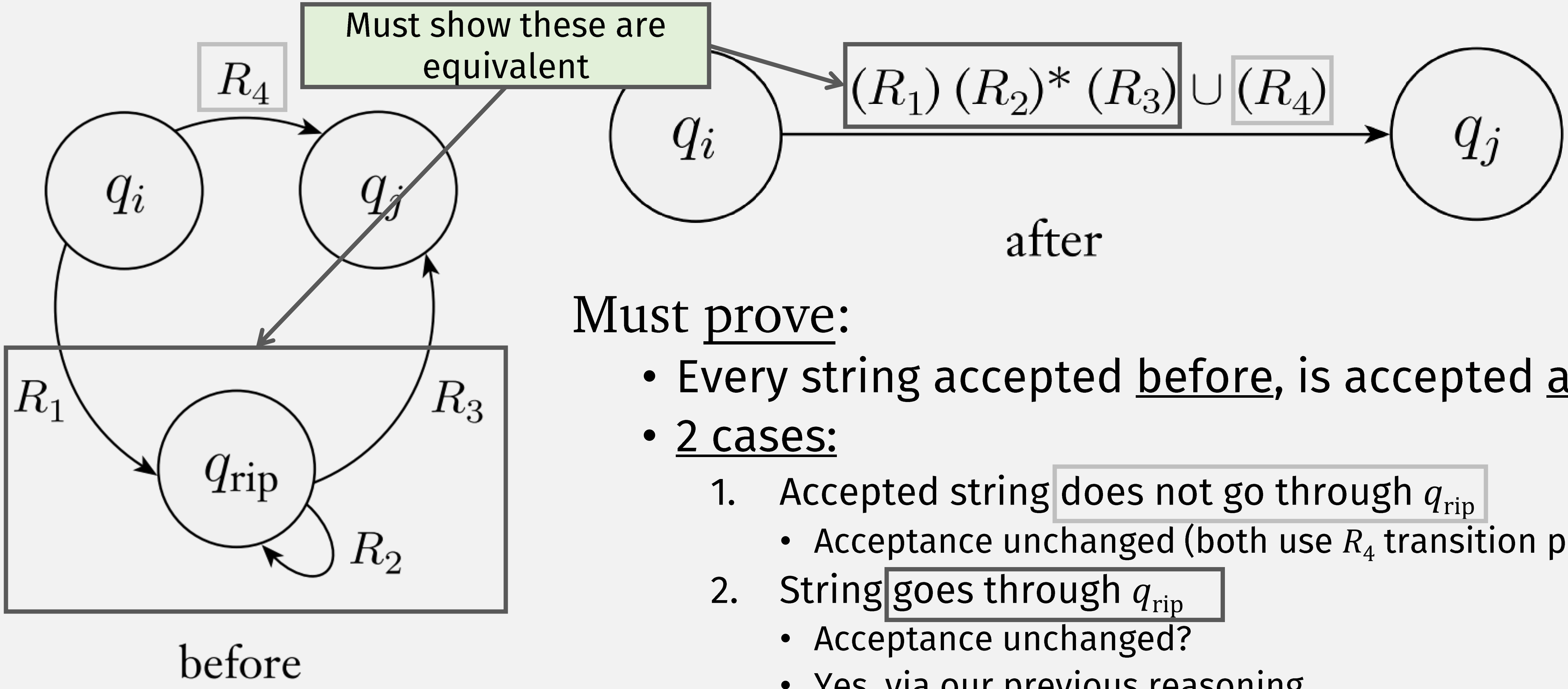
GNFA→RegExpr "Correctness"

- "Correct" / "Equivalent" means:

$$\text{LANGOF}\,(\,G\,) = \text{LANGOF}\,(\,\mathbf{GNFA{\rightarrow}RegExpr}(\,G\,)\,)$$

- i.e., **GNFA→RegExpr** must not change the language!
  - Key step: the rip/repair step

GNFA→RegExpr: Rip/Repair "Correctness"



Must prove:

- Every string accepted before, is accepted after
- 2 cases:
  1. Accepted string does not go through q_{rip}
     - Acceptance unchanged (both use R_4 transition part)
  2. String goes through q_{rip}
     - Acceptance unchanged?
     - Yes, via our previous reasoning

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

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

Need to convert DFA or NFA to Regular Expression …

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

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

☑ • Convert regular expression → equiv NFA!

■

Now we may use regular expressions to represent regular langs.

So a regular language has these equivalent representations:

- DFA
- NFA
- Regular Expression

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

How to Prove A Language Is Regular?

- 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.

Kinds of Mathematical Proof

- Proof by construction
- Proof by induction
  - Use this when working with <u>recursive</u> definitions

In-Class quiz 9/29

See gradescope