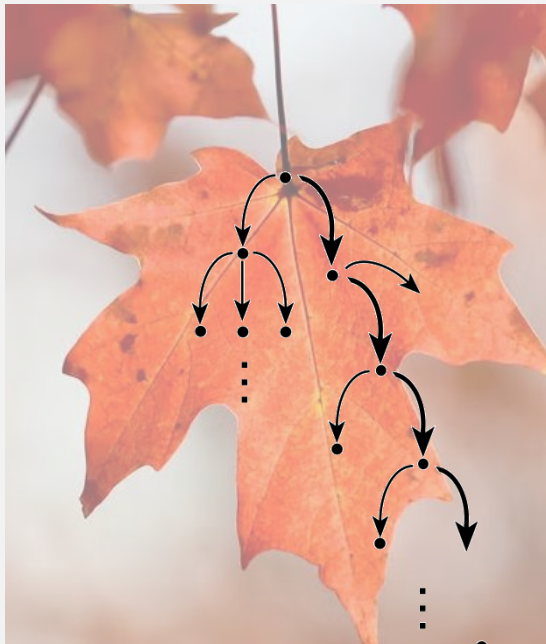
CS 420 / CS 620

Nondeterminism

Wednesday, September 24, 2025

UMass Boston Computer Science



Announcements

- HW 3
  - Out: **Mon 9/22 12pm (noon)**
  - Due: **Mon 9/29 12pm (noon)**

- Questions / Complaints about grading?
  - GradeScope re-grade requests welcome
  - Please be specific
  - Do not ask the instructor
    (we have many graders)

In-class question preview

- What are the different things the epsilon symbol (ε) can represent?

Last Time

Why Care About Closed Ops on Reg Langs?

- **Closed operations** for **Regular langs preserve "regularness"**
  - I.e., it **preserves** the **same computation model!**
- Enables **"combining" smaller "regular" computations** into **bigger ones:**

For Example:

OR: Regular Lang × Regular Lang → Regular Lang

- In general, this semester, **we want operations that are closed!**

Last Time

Is Union Closed For Regular Langs?

In this course, we are interested in closed operations for a set of languages (here the set of regular languages)

(In general, a set is **closed** under an operation if applying the operation to members of the set produces a result in the same set)

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 $A_1 \cup A_2$.

Want to prove this statement

Or this (same) statement

Last Time

Is Union Closed For Regular Langs?

(In general, a **set** is **closed** under an operation if applying the **operation** to **members of the set** produces a **result in the same set**)

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 $A_1 \cup A_2$.

Want to prove this statement

Or this (same) statement

A **member of the set of regular languages** is …

… **a regular language,** which itself is a set (of strings) …

… so the **operations** we're interested in are **set operations**

Last Time

Is Union 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 $A_1 \cup A_2$.

Want to prove this statement

Or this (same) statement

Flashback: Mathematical Statements: IF-THEN

Using:

- If we know: $P \to Q$ is TRUE,
  what do we know about P and Q individually?
  - Either P is FALSE (not too useful, **can't prove anything about Q**), or
  - If P is TRUE, then Q is TRUE (**modus ponens**)

Proving:

- To prove: $P \to Q$ is TRUE:
  - Prove P is FALSE (**usually hard** or **impossible**)
  - Assume P is TRUE, then prove Q is TRUE

p	q	$p \to q$
True	True	True
True	False	False
False	True	True
False	False	True

Is Union Closed For Regular Langs?

Definition of Regular Language

If a **DFA recognizes** a lang, then it's **regular**

Do we know anything about A_1 and A_2?

Sta

1. A_1 and A_2 are regular languages
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ (todo)
5. M recognizes $A_1 \cup A_2$
6. $A_1 \cup A_2$ is a 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 $A_1 \cup A_2$.

How to create this M? Don't know what A_1 and A_2 are!

1. Assumption of If part of If-Then
2. Def of Regular Language
3. Def of Regular Language
4. Def of DFA
7. From stmt #1 and #6

Corollary

Definition of Regular Language (Corollary)

If a lang is **regular**, then it has a **DFA**

To prove $P \rightarrow Q$ is TRUE: Assume P is TRUE, then prove Q is TRUE

Is Union Closed For Regular Langs?

Statements	Justifications
1. A_1 and A_2 are regular languages	1. Assumption of If part of If-Then Corollary
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1	2. Def of Regular Language
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2	3. Def of Regular Language
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ (todo)	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. See examples
6. $A_1 \cup A_2$ is a regular language	6. Def of Regular Language
7.	7. From stmt #1 and #6

???

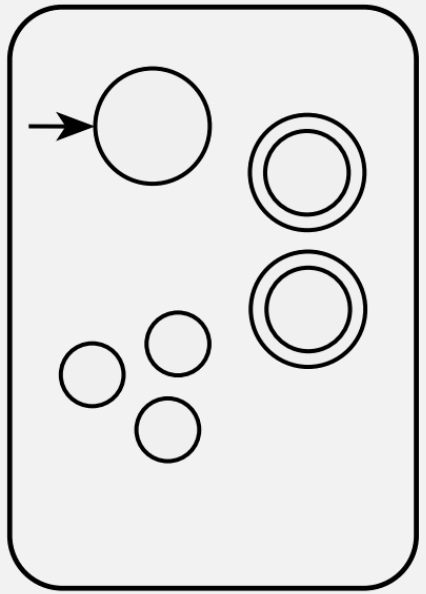
How to create this M? Don't know what A_1 and A_2 are!

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 $A_1 \cup A_2$.

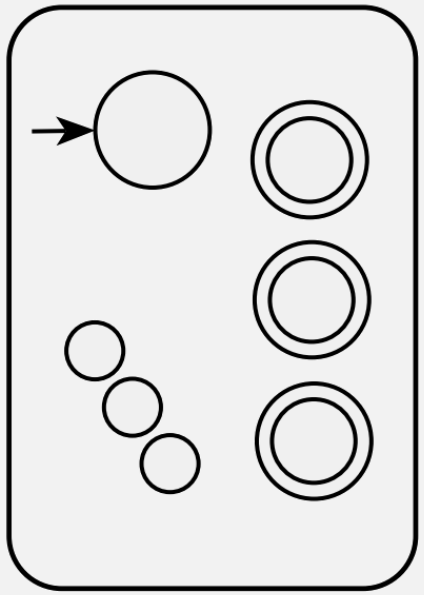
M_1

recognizes A_1



M_2

recognizes A_2



DEFINITION

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***.

Regular language A_1
Regular language A_2

Even if we <u>don't know</u> what these languages are, <u>we still know</u>...

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

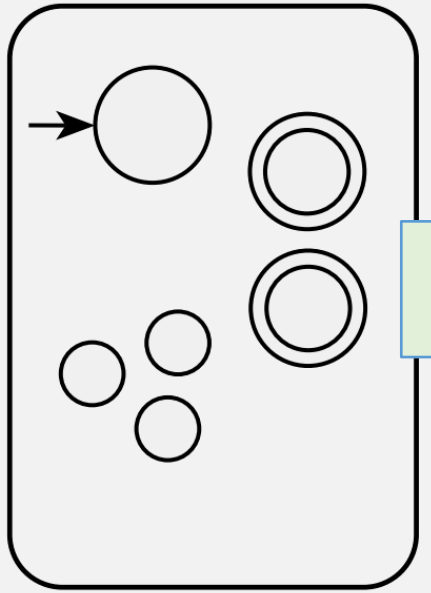
Definition of Regular Language (Corollary)

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

Union

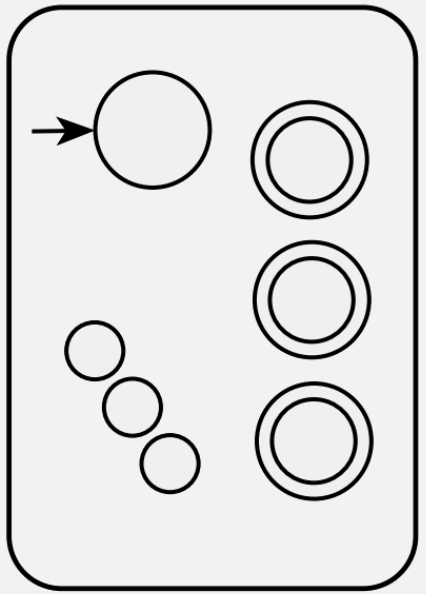
M_1

recognizes A_1



M_2

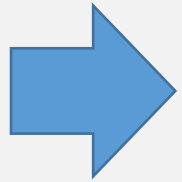
recognizes A_2

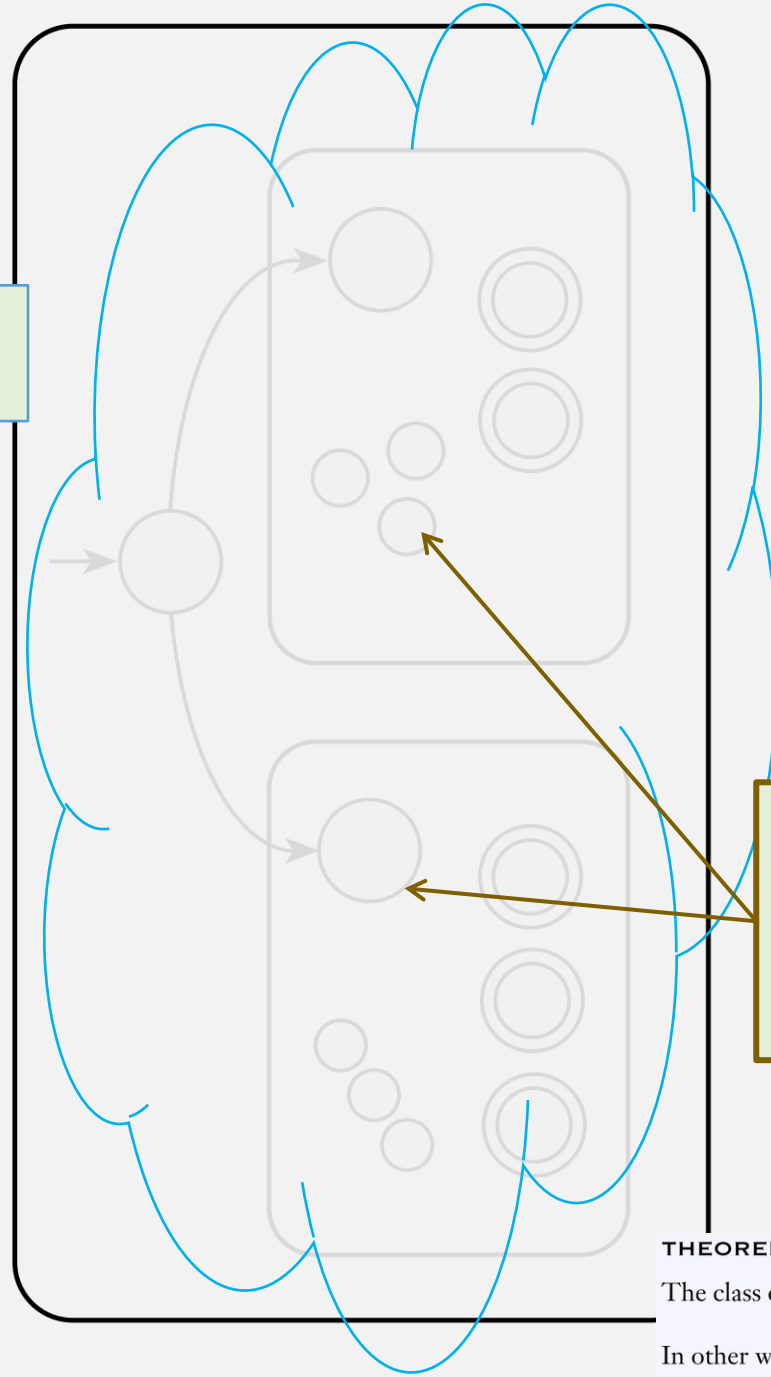


Want: M

Recognizes $A_1 \cup A_2$

(to prove $A_1 \cup A_2$ is regular)





Rough sketch Idea: M is a combination of M_1 and M_2 that: checks whether its input is accepted by either M_1 or M_2

But: a DFA can only read its input once!

Need to: somehow simulate "being in" both an M_1 and M_2 state simultaneously

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 $A_1 \cup A_2$.

Union is Closed For Regular Languages

Proof (continuation)

- 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,

Want: M that can **simultaneously "be in" both** an M_1 and M_2 state

- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M:

$$Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} \quad = Q_1 \times Q_2$$

This set is the ***Cartesian product*** of sets Q_1 and Q_2

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

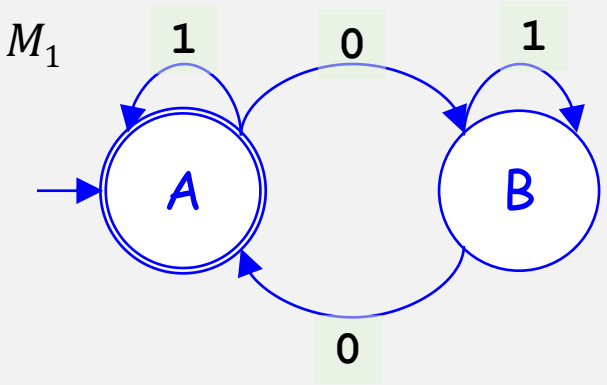
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

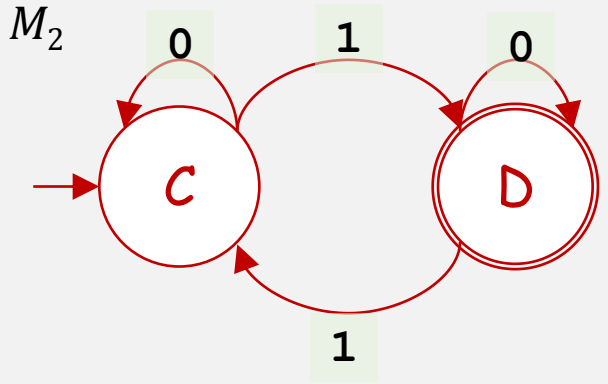
A state of M is a pair:
- first part: state of M_1
- second part: state of M_2

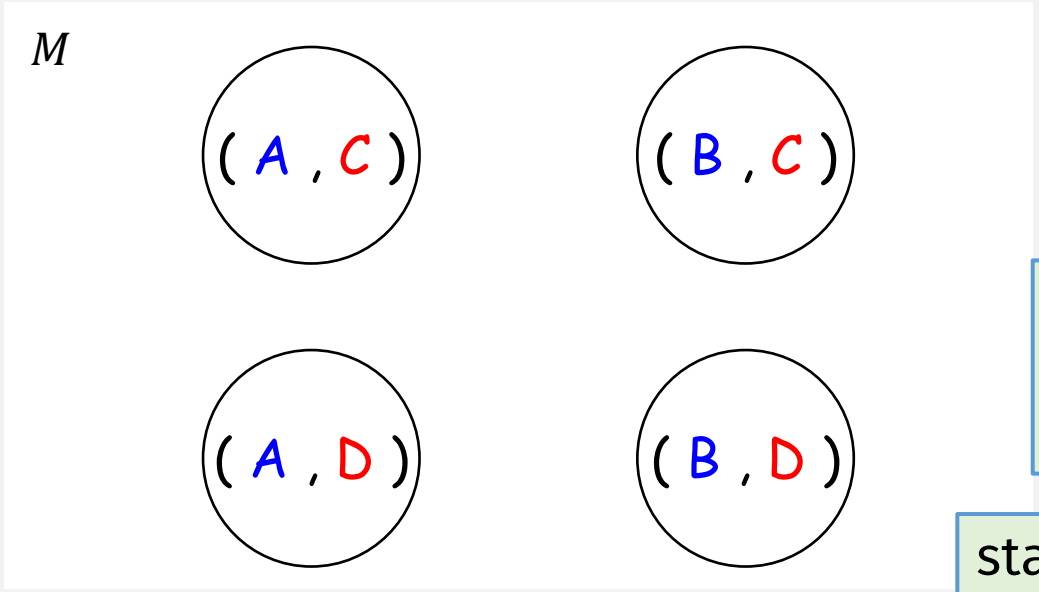
states of M:
all pair combos of M_1 and M_2 states

DFA Union Example

Note:
We do not know M_1 or M_2 exactly!
But: a concrete example helps understanding







A state of M is a pair:
- first part: state of M_1
- second part: state of M_2

states of M:
all pair combos of M_1 and M_2 states

https://cs.wellesley.edu/~cs235/fall10/lectures/14_DFA_operations_revised_2.pdf

Union is Closed For Regular Languages

Proof (continuation)

- 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: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M:

$$Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$$

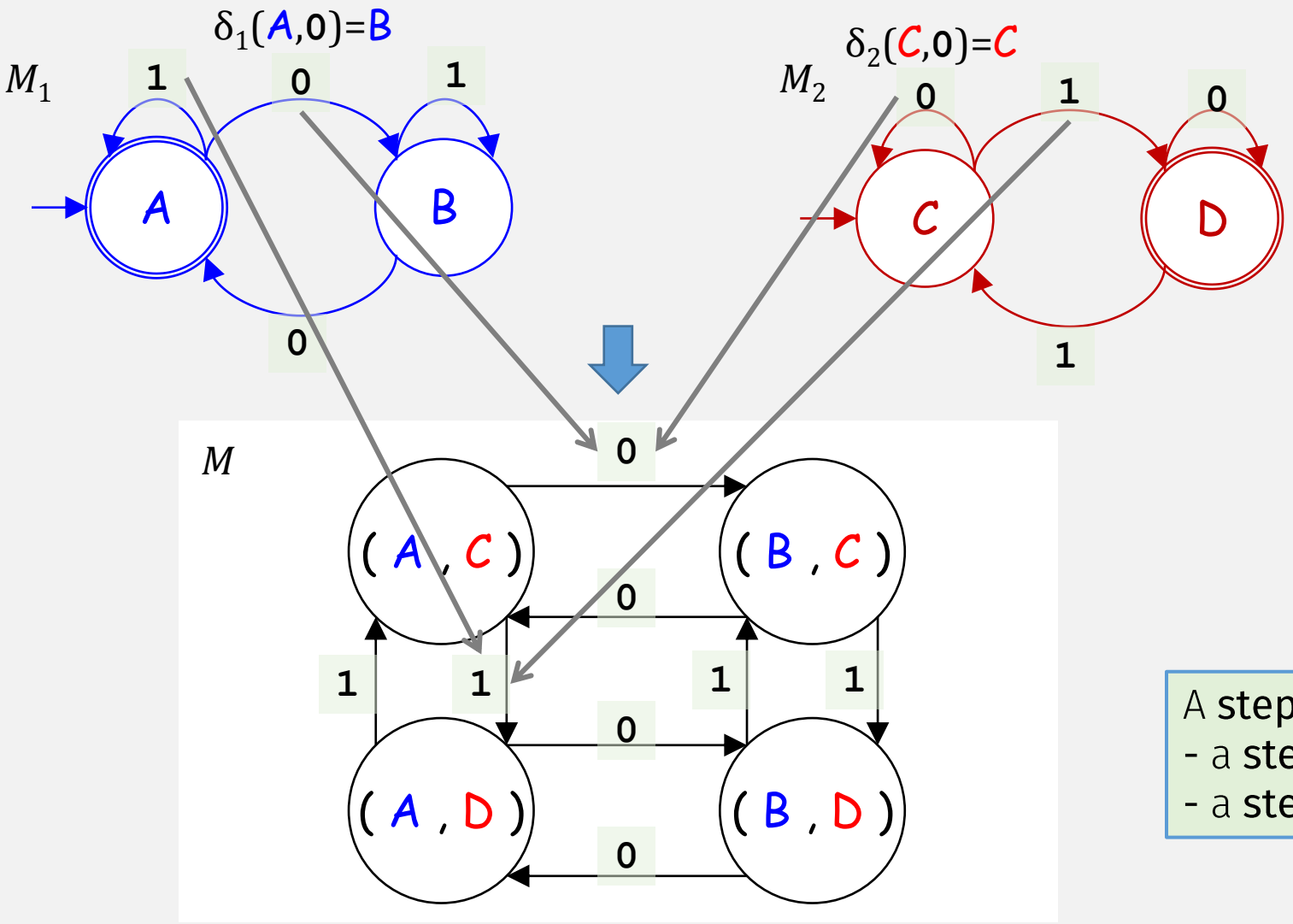
This set is the ***Cartesian product*** of sets Q_1 and Q_2

$$a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

A step in M is both:
- a step in M_1, and
- a step in M_2

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 **step** in M is <u>both</u>:
- a **step** in M_1, and
- a **step** in M_2

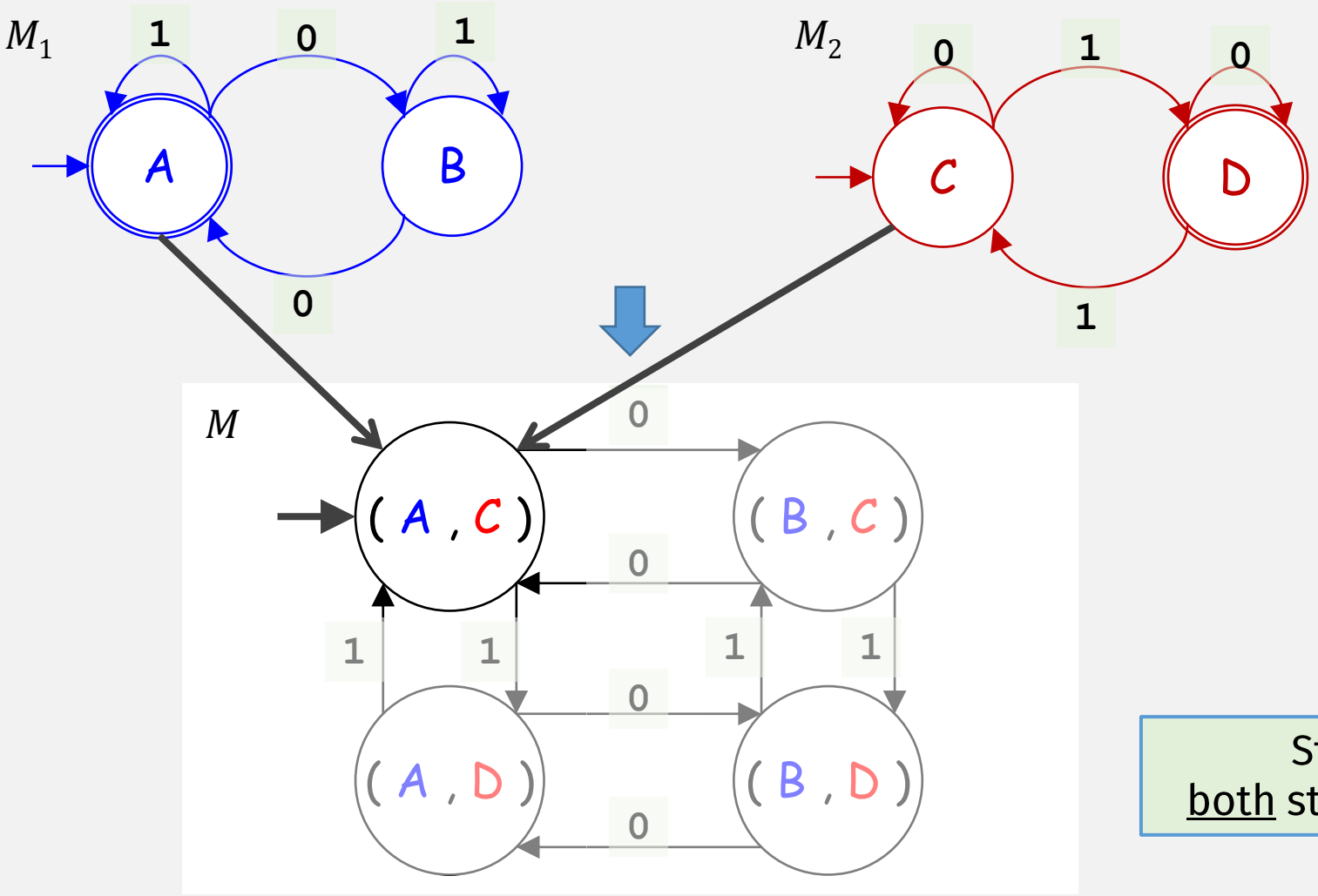
Union is Closed For Regular Languages

Proof (continuation)

- 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: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- 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
- M transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- M start state: (q_1, q_2)

Start state of M is:
both start states of M_1 and M_2

DFA Union Example



Start state of M is:
<u>both</u> start states of M_1 and M_2

https://cs.wellesley.edu/~cs235/fall10/lectures/14_DFA_operations_revised_2.pdf

Union is Closed For Regular Languages

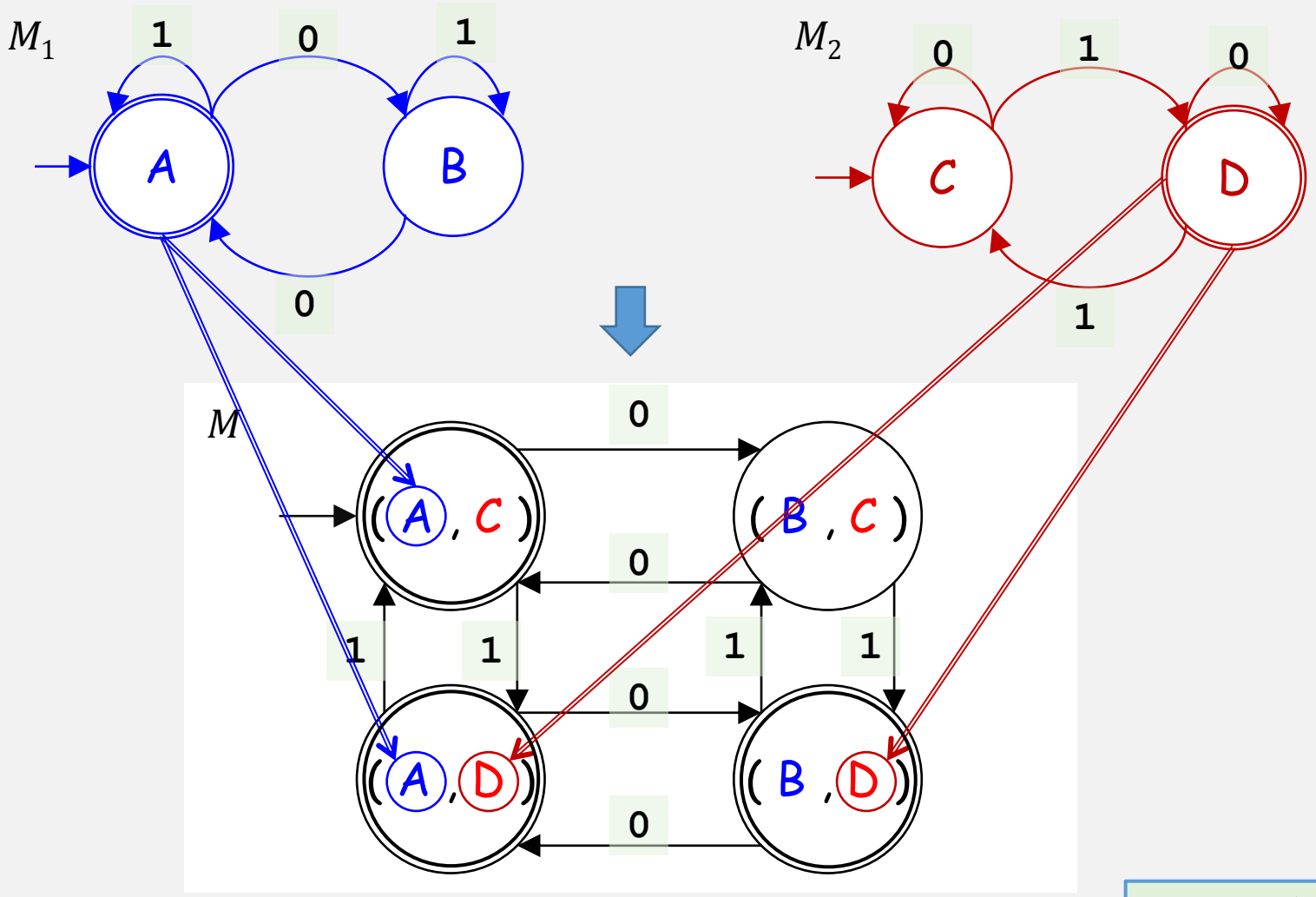
Proof (continuation)

- 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: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\} \quad = Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- M transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- M start state: (q_1, q_2)
- M accept states: $F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Remember:
Accept states must be subset of Q

Accept if either M_1 or M_2 accept

DFA Union Example



Accept if either M_1 or M_2 accept

https://cs.wellesley.edu/~cs235/fall10/lectures/14_DFA_operations_revised_2.pdf

Union is Closed For Regular Languages

Proof (continuation)

- 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,

Define the function:

$\text{UNION}_{\text{DFA}}(M_1, M_2) = M =$ $(Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$

- 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
- M transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- M start state: (q_1, q_2)
- M accept states: $F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Q.E.D.? ■

Previously

Is Union Closed For Regular Langs?

Statements	Justifications
1. A_1 and A_2 are regular languages	1. Assumption
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1	2. Def of Regular Language
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2	3. Def of Regular Language
4. Construct DFA M = $\text{UNION}_{\text{DFA}}(M_1, M_2)$ ☑	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. See examples (TODO!)
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

???

How to create this? Don't know what A_1 and A_2 are!

“Prove” that DFA recognizes a language

Let $s_1 \in A_1$ and $s_2 \in A_2$
Let $s_3 \notin A_1$ and $s_4 \notin A_2$

Be careful when choosing examples!

In this class, **a table like this is sufficient to “prove” that a DFA recognizes a language**

String	In lang $A_1 \cup A_2$?	Accepted by M ?
s_1	Yes	
s_2	Yes	
s_3	???	
s_4	???	

Don’t know A_1 and A_2 exactly …

… but we know …

… they are **sets of strings**!

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

constructed $M = (Q, \Sigma, \delta, q_0, F)$ recognizes $A_1 \cup A_2$?

“Prove” that DFA recognizes a language

Let $s_1 \in A_1$ and $s_2 \in A_2$

~~Let $s_3 \notin A_1$ and $s_4 \notin A_2$~~

Let $s_5 \notin A_1$ and $\notin A_2$

String	In lang A_1 U A_2 ?	Accepted by M ?
s_1	Yes	
s_2	Yes	
~~s_3~~	~~???~~	
~~s_4~~	~~???~~	
s_5	No	

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

constructed $M = (Q, \Sigma, \delta, q_0, F)$ recognizes A_1 U A_2?

Union is Closed For Regular Languages

Proof (continuation)

- 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: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- 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
- M transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- M start state: (q_1, q_2)

Accept if either M_1 or M_2 accept

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

"Prove" that DFA recognizes a language

Let $s_1 \in A_1$ and $s_2 \in A_2$

Let $s_5 \notin A_1$ and $\notin A_2$

(this column needed when **machine is not concrete**, i.e., **can't directly run machine** to check if string is accepted)

String	In lang $A_1 \cup A_2$?	Accepted by M ?	Justification
s_1	Yes	Accept ??	(J1)
s_2	Yes	Accept	(J1)
~~s_3~~	~~???~~	???	
~~s_4~~	~~???~~	???	
s_5	No	Reject ??	(J2)

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

a string $\in A_2 \rightarrow$ accepted by $M_2 \rightarrow$ accepted by M	(J1)
string $\notin A_1$ and $\notin A_2 \rightarrow M_1$ and M_2 rejects $\rightarrow M$ rejects	(J2)

constructed $M = (Q, \Sigma, \delta, q_0, F)$ to Accept if either M_1 or M_2 accept Else reject

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

Alternate Examples Table

"Prove" that DFA recognizes a language

Let $s_1 \in A_1$ and $s_2 \in A_2$

(required when **machine is not concrete**, i.e., **can't directly run machine** to check if string is accepted)

Let $s_5 \notin A_1$ and $\notin A_2$

String	In lang $A_1 \cup A_2$?	Accepted by $M = \text{UNION}_{\text{DFA}}(M_1, M_2)$	n
s_1	Yes	Accept	(J1)
s_2	Yes	Accept	(J1)
~~s_3~~	~~???~~	???	
~~s_4~~	~~???~~	???	
s_5	No	Reject	(J2)

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

$M = \text{UNION}_{\text{DFA}}(M_1, M_2)$

$s_1 \in A_1 \rightarrow$ accepted by $M_1 \rightarrow$ accepted by M	(J1)
$s_5 \notin A_1$ and $\notin A_2 \rightarrow M_1$ and M_2 rejects $\rightarrow M$ rejects	(J2)

Alternate Examples Table

"Prove" that DFA recognizes a language

Let $s_1 \in A_1$ and $s_2 \in A_2$

Let $s_5 \notin A_1$ and $\notin A_2$

(required when **machine is not concrete**, i.e., **can't directly run machine** to check if string is accepted)

String	$\in A_1$?	$\in A_2$?	M_1 result?	M_2 result?	$\in A_1 \cup A_2$?	M = UNION$_{DFA}$ (M_1,M_2) result?
s_1	Yes		Accept		Yes	Accept
s_2		Yes		Accept	Yes	Accept
~~s_3~~						
~~s_4~~						
s_5	No	No	Reject	Reject	No	Reject

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,

$s_1 \in A_1 \rightarrow$ accepted by $M_1 \rightarrow$ accepted by M

$s_5 \notin A_1$ and $\notin A_2 \rightarrow M_1$ and M_2 rejects $\rightarrow M$ rejects

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

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

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

Previously

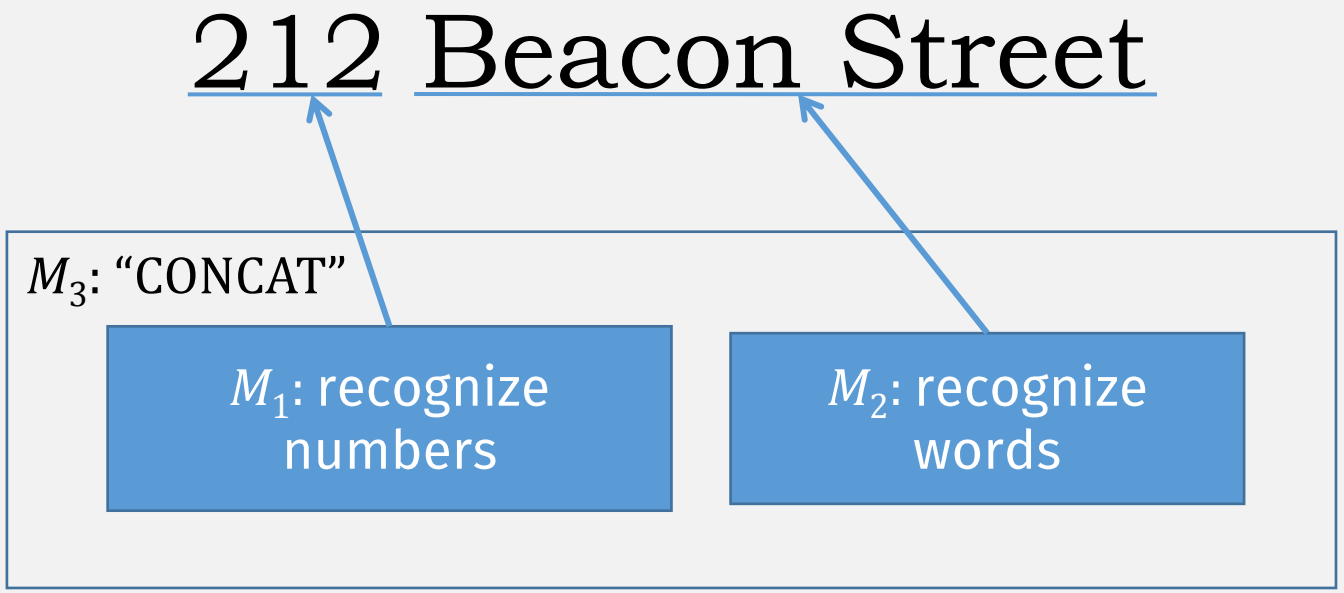
Is Union Closed For Regular Langs?

Statements	**Justifications**
1. A_1 and A_2 are regular languages	1. Assumption
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1	2. Def of Regular Language
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2	3. Def of Regular Language
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ ☑	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. See Examples 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. ■

Another (common string) operation: Concatenation

Example: **Recognizing street addresses**



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

Concatenation of Languages

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

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

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

Is Concatenation Closed?

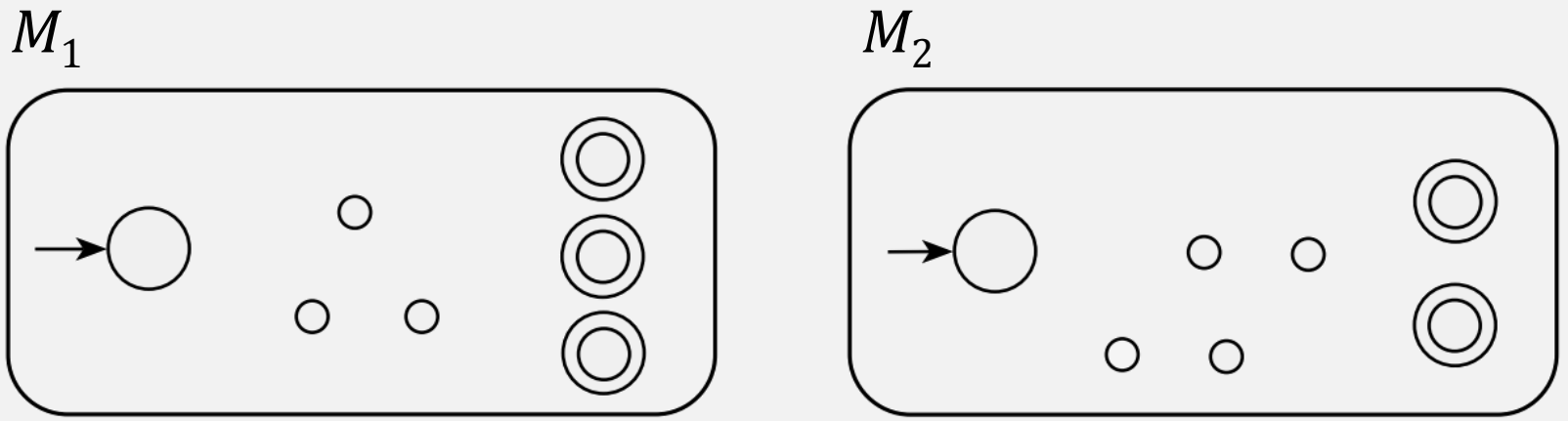
THEOREM

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 $A_1 \circ A_2$.

- **Construct a <u>new</u> machine M recognizing $A_1 \circ A_2$?** (like union)
  - Using **DFA M_1** (which recognizes A_1),
  - and **DFA M_2** (which recognizes A_2)

Concatentation

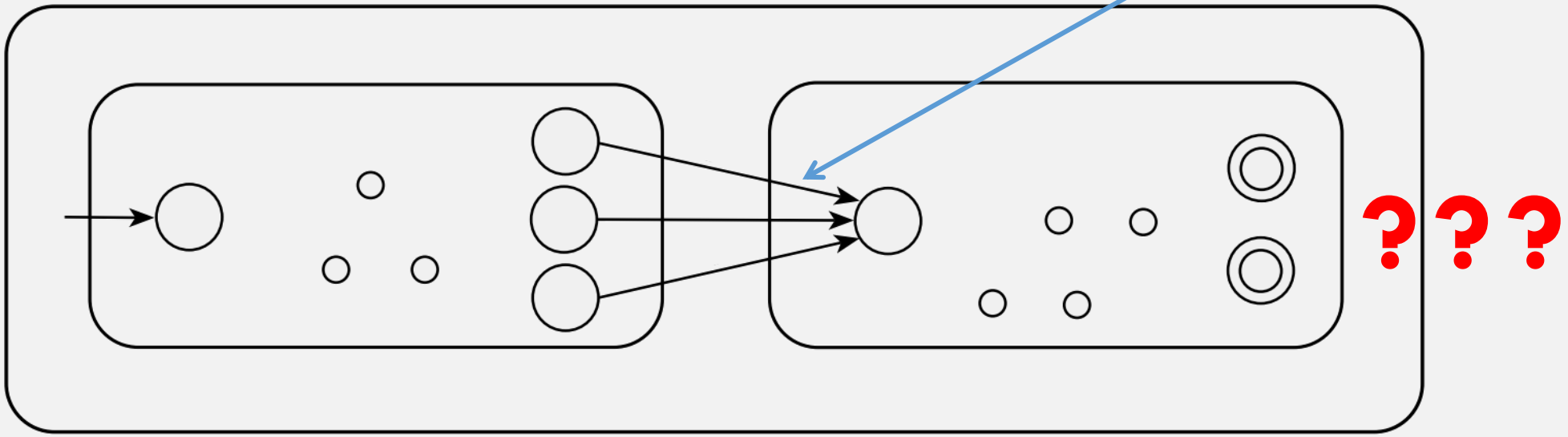


PROBLEM:
Can only read input once, can't backtrack

Let M_1 recognize A_1, and M_2 recognize A_2.

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

Need to switch machines at some point, but when?



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

Overlapping Concatenation Example

- Let M_1 recognize language A = { `jen`, `jens` }
- and M_2 recognize language B = { `smith` }
- Want: Construct M to recognize $A \circ B$ = { `jen``smith`, `jen``s`**`smith`** }

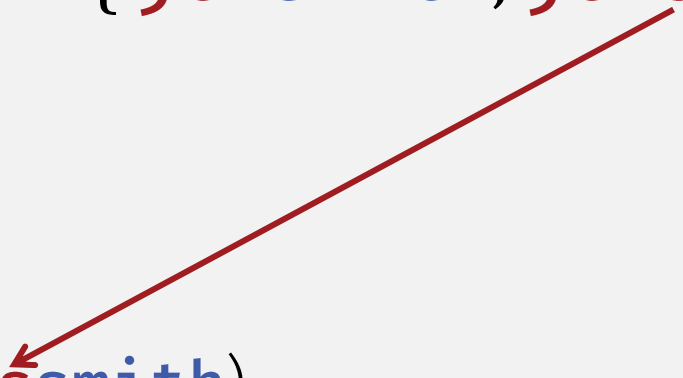
- If M sees `jen` …
- M must decide to either:

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

Overlapping Concatenation Example

- Let M_1 recognize language A = { `jen`, `jens` }
- and M_2 recognize language B = { `smith` }
- Want: Construct M to recognize $A \circ B$ = { `jensmith`, `jenssmith` }

- If M sees `jen` …
- M must decide to either:
  - stay in M_1 (correct, if full input is `jenssmith`)



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

Overlapping Concatenation Example

- Let M_1 recognize language A = { `jen`, `jens` }
- and M_2 recognize language B = { `smith` }
- Want: Construct M to recognize $A \circ B$ = { `jensmith`, `jenssmith` }

- If M sees `jen` …
- M must decide to either:
  - stay in M_1 (correct, if full input is `jenssmith`)
  - or switch to M_2 (correct, if full input is `jensmith`)
- But to recognize $A \circ B$, it needs to handle both cases!!
  - Without backtracking

A **DFA** can't do this!

Is Concatenation Closed?

FALSE?

THEOREM

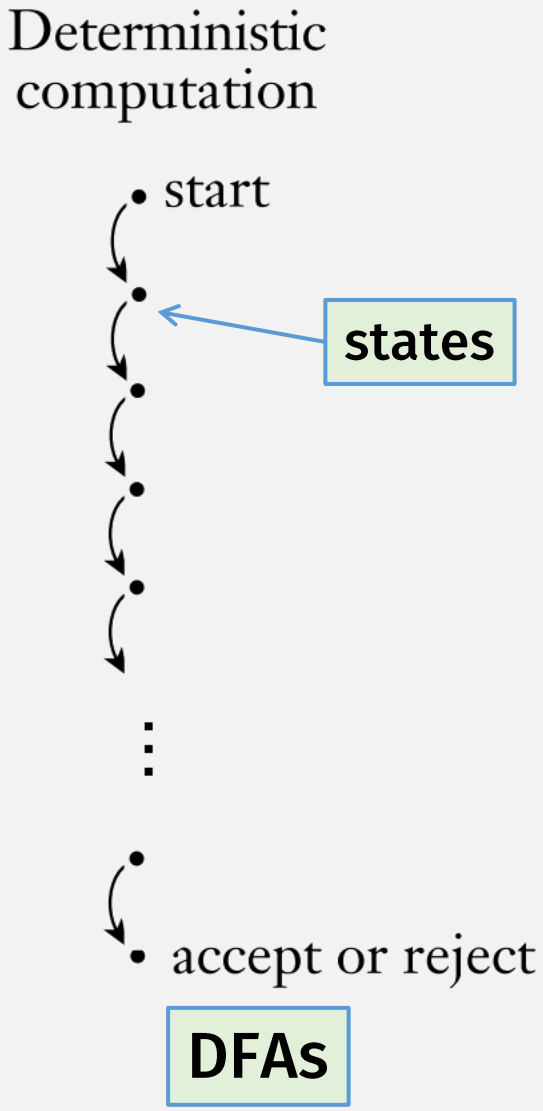
The class of regular languages is closed under the concatenation operation.

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

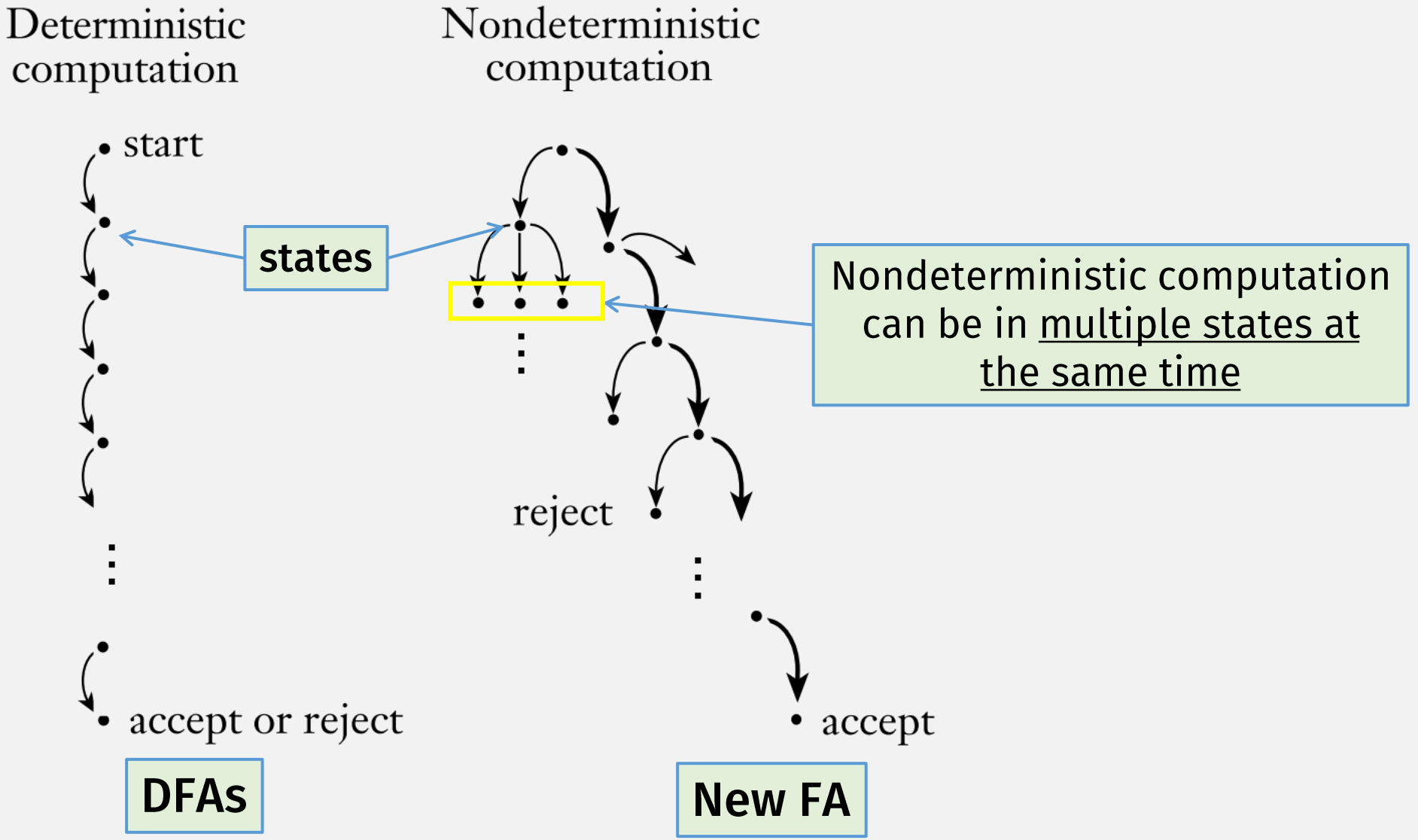
- Cannot **combine A_1 and A_2's machine because:**
  - **Need to switch from A_1 to A_2 at some point …**
  - … but **we don't know when!** (we can only read input once)
- **This requires a new kind of machine!**
- But **does this mean concatenation is not closed for regular langs?**

Nondeterminism

Deterministic vs Nondeterministic



Deterministic vs Nondeterministic



Previously

DFAs: The Formal Definition

DEFINITION

deterministic

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***.

Deterministic Finite Automata (DFA)

Nondeterministic Finite Automata (NFA)

DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Compare with 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***.

Difference

Power set, i.e. a transition results in set of states

Power Sets

- A **power set** is the set of all subsets of a set
- Example: S = {a, b, c}
- Power set of S =
  - { { }, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c} }
  - Note: includes the empty set!

Nondeterministic Finite Automata (NFA)

DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

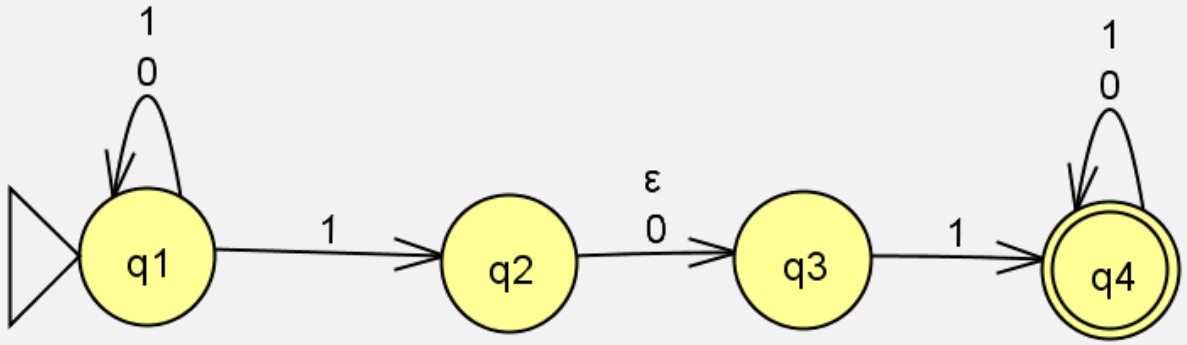
Transition label can be "empty", i.e., machine can transition without reading input

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$$

CAREFUL:
ε symbol is reused here, as a transition label (ie, an argument to δ)
- **It's not the empty string!**
- And it's (still) not a character in the alphabet Σ!

NFA Example

- Come up with a formal description of the following NFA:



DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

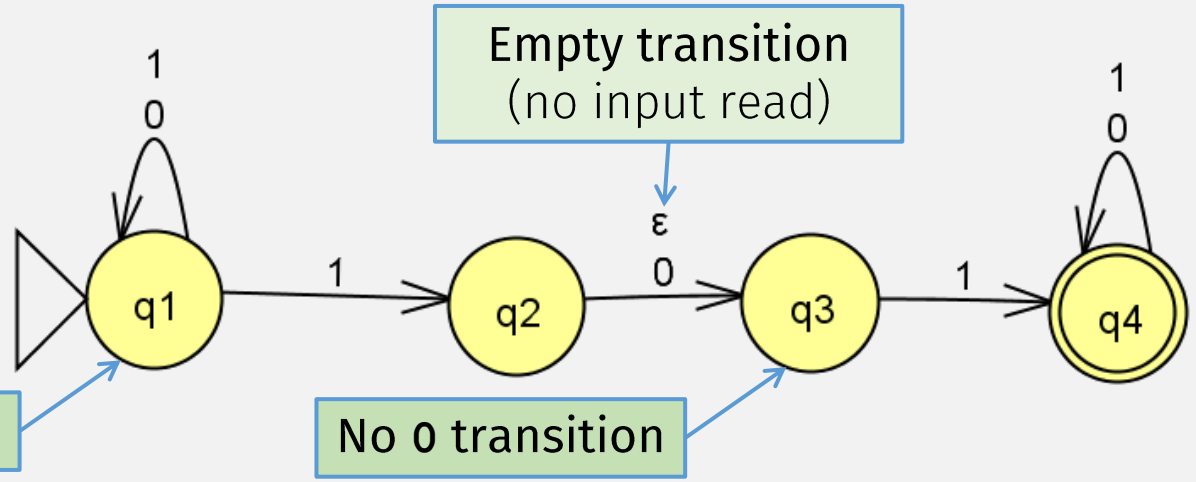
$$\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$$

Empty transition (no input read)

Result of transition is a set

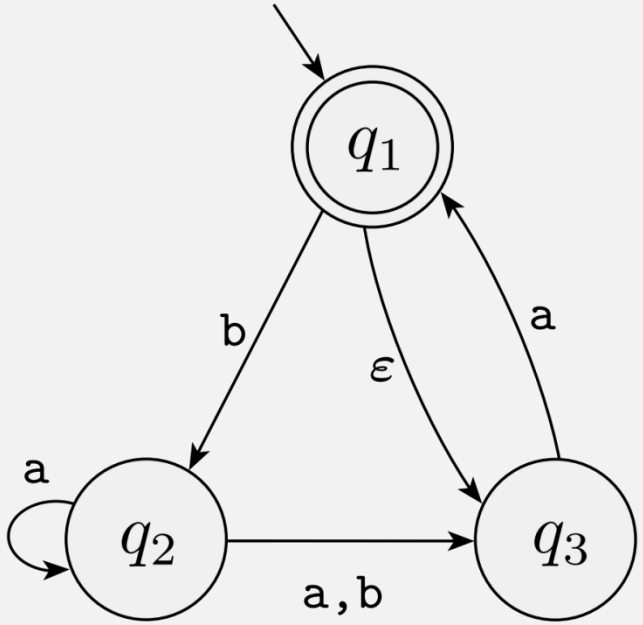
	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.



In-class Exercise

- Come up with a formal description for the following NFA
  - $\Sigma = \{ \texttt{a}, \texttt{b} \}$



DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

In-class Exercise Solution

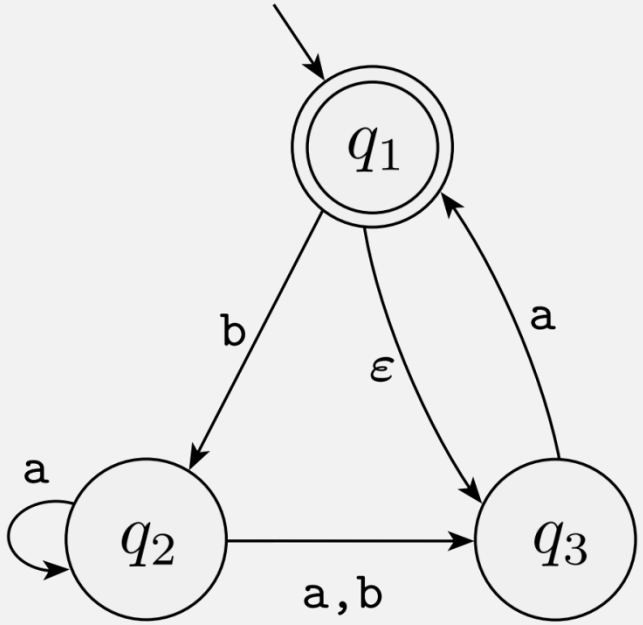
Let $N = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{ q_1, q_2, q_3 \}$
- $\Sigma = \{ \mathtt{a}, \mathtt{b} \}$
- δ ...
- $q_0 = q_1$
- $F = \{ q_1 \}$

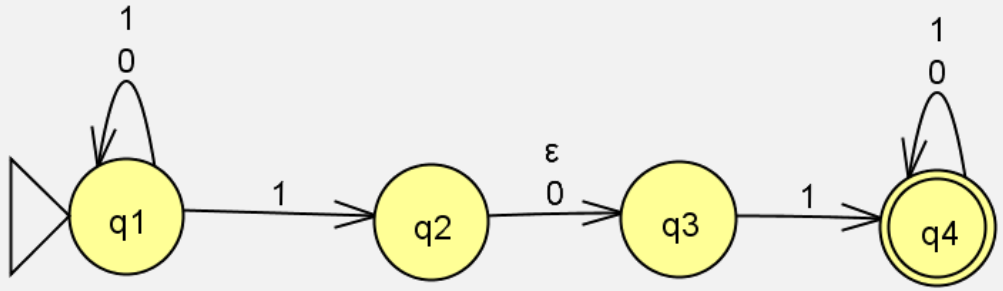
$$\delta(q_1, \mathtt{a}) = \{\}$$
$$\delta(q_1, \mathtt{b}) = \{q_2\}$$
$$\delta(q_1, \varepsilon) = \{q_3\}$$
$$\delta(q_2, \mathtt{a}) = \{q_2, q_3\}$$
$$\delta(q_2, \mathtt{b}) = \{q_3\}$$
$$\delta(q_2, \varepsilon) = \{\}$$
$$\delta(q_3, \mathtt{a}) = \{q_1\}$$
$$\delta(q_3, \mathtt{b}) = \{\}$$
$$\delta(q_3, \varepsilon) = \{\}$$

Differences with DFA?

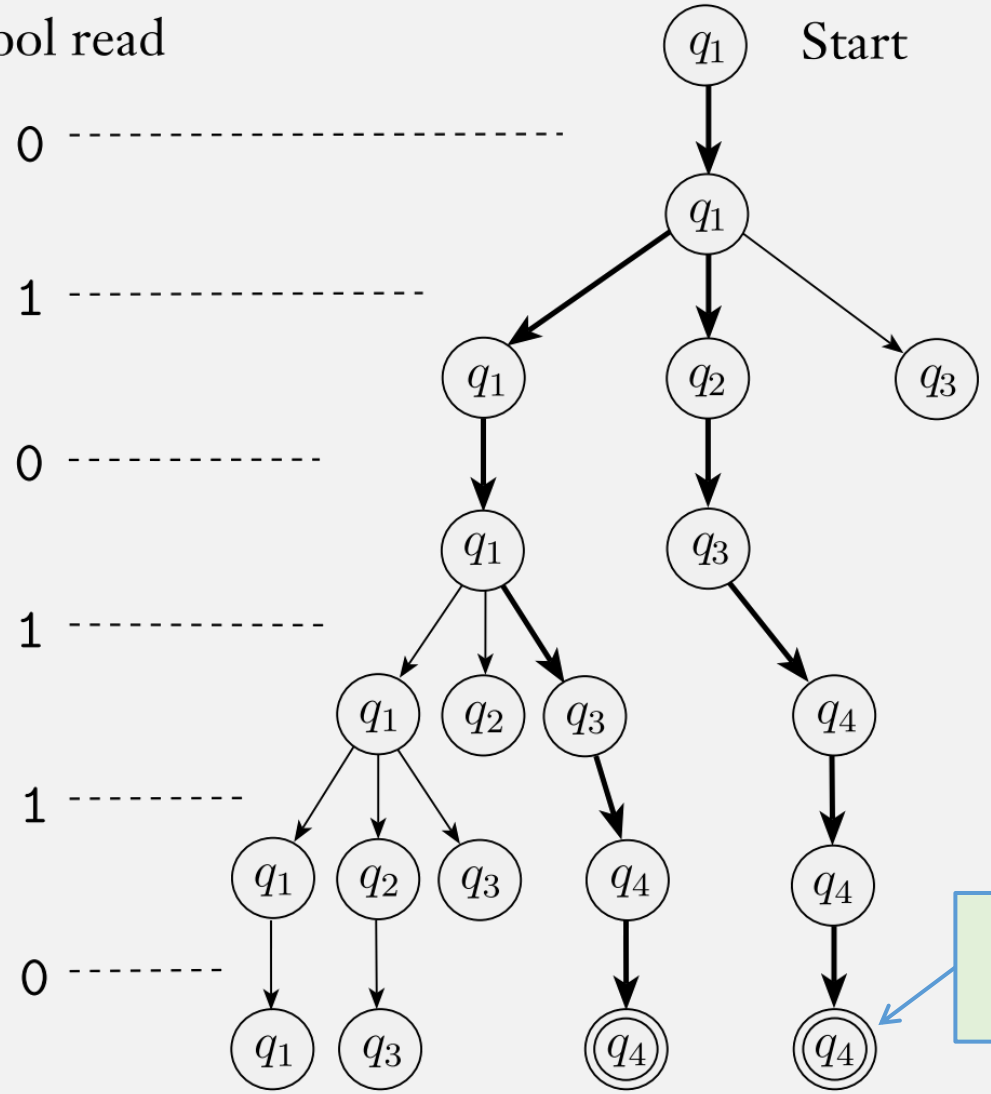
- δ output is a set
- state doesn't need transition for every alphabet symbol
- state can have multiple transitions for one symbol
- can have "empty" transitions (δ output is empty set)



NFA Computation (JFLAP demo): **010110**



NFA Computation Sequence



NFA **accepts** input if at least one path ends in accept state

Each step can branch into multiple states at the same time!

So this is an **accepting computation**

Flashback

DFA Computation Rules

Informally

Given
- A **DFA** (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A **DFA computation** (~ "Program run"):
- **Start** in ***start state***

- **Repeat**:
  - **Read 1 char** from **Input,** and
  - **Change state** according to ***transition rules***

Result of **computation:**
- **Accept** if **last state** is ***Accept state***
- **Reject** otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

A DFA **computation** is a sequence of states:

- specified by $\hat{\delta}(q_0, w)$ where:

- M **accepts** w if $\hat{\delta}(q_0, w) \in F$
- M **rejects** otherwise

Flashback

DFA Computation Rules

Informally

Given

- A **DFA** (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A **DFA** computation (~ "Program run"):

- Start in *start state*

- Repeat:
  - Read **1 char** from **Input,** and
  - Change state according to *transition rules*

Result of **computation:**

- Accept if **last state** is *Accept state*
- Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

A **DFA computation** is a sequence of states:

- specified by $\hat{\delta}(q_0, w)$ where:

- M **accepts** w if $\hat{\delta}(q_0, w) \in F$
- M **rejects** otherwise

Ignoring ε transitions, for now!

NFA Computation Rules

Informally

Given

- An NFA (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

An NFA **computation** (~ "Program run"):

- Start in *start state*
- Repeat:
  - Read 1 char from **Input**, and according to *transition rules*

For each "current" state, go to next states

... then combine all "next states"

Result of **computation**:

- Accept if last set of states has accept state
- Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

An **NFA computation** is a ...

- specified by $\hat{\delta}(q_0, w)$ where:

- M **accepts** w if ...
- M **rejects** ...

Ignoring ε transitions, for now!

NFA Computation Rules

Informally

Given
- An NFA (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A DFA computation (~ "Program run"):
- Start in *start state*
- Repeat:
  - Read 1 char from **Input**, and according to *transition rules*

For each "current" state, go to next states

... then combine all "next states"

Result of **computation**:
- Accept if last set of states has accept state
- Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

An **NFA computation** is a sequence of: sets of states

- specified by $\hat{\delta}(q_0, w)$ where:

???

- M **accepts** w if ...
- M **rejects** ...

Flashback

DFA Multi-Step Transition Function

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

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

Recursive Input Data needs Recursive Function

Base case

A **String** is either:

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

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

Flashback

DFA Multi-Step Transition Function

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

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

(Defined recursively)

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

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

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

Recursive Input Data needs Recursive Function

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

Recursive case

Recursion on string

"smaller" argument

string

char

Recursion on string

string

char

"second to last" state

Flashback

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

DFA Multi-Step Transition Function

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

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

(Defined recursively)

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

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

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

Single step from "second to last" state and last char gets to last state

Recursive Input Data needs Recursive Function

A **String** is either:

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

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

NFA Multi-Step Transition Function

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

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

Result is set of states

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

NFA Multi-Step Transition Function

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

Result is set of states

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

  states $qs \subseteq Q$

(Defined recursively)

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

Recursively Defined Input needs Recursive Function

Base case

A **String** is either:

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

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

NFA Multi-Step Transition Function

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

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

  states $qs \subseteq Q$

(Defined recursively)

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

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

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

Recursion on recursive part

"second to last" set of states

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

Recursively Defined Input needs Recursive Function

A **String** is either:

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

Recursive case

Recursive part

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

NFA Multi-Step Transition Function

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

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

(Defined recursively)

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

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

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

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

Recursively Defined Input needs Recursive Function

A **String** is either:

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

Last char

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

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

NFA Multi-Step Transition Function

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

- Domain (inpu
  - state $q \in$
  - string $w =$
- Range (outpu

  states $qs \subseteq$

(Defined recurs

Given
- An **NFA** (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A **DFA computation** (~ "Program run"):
- Start in *start state*
- Repeat:
  - Read 1 char from **Input,** and
  - according to *transition rules*

Recursively Defined Input needs

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

Still ignoring ε transitions!

For each "current" state, go to next states

... then combine all sets of "next states"

Recursive Case

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

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

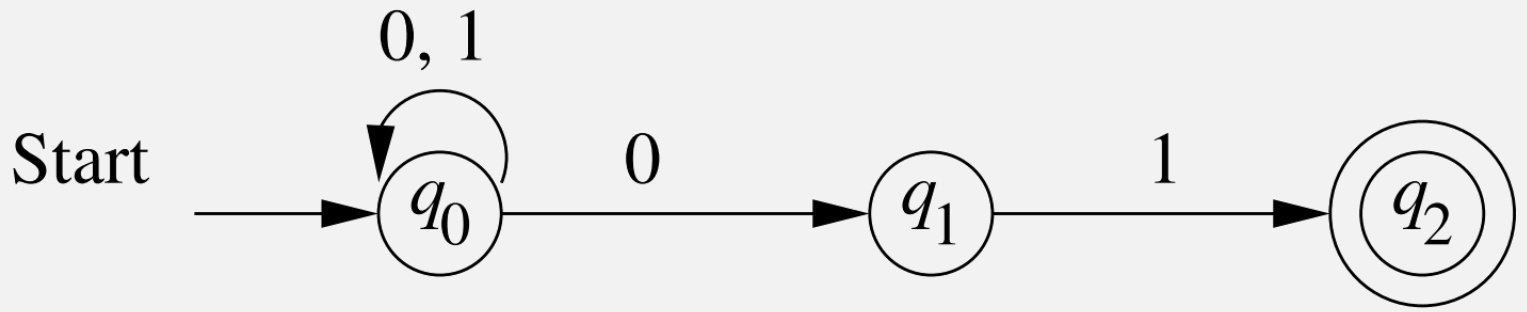
$$\hat{\delta}(q, w') = \{q_1, \ldots, q_k\}$$

NFA Multi-Step δ Example

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

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

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



- $\hat{\delta}(q_0, \epsilon) = \{q_0\}$
- $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

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

Combine result of recursive call with "last step"

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

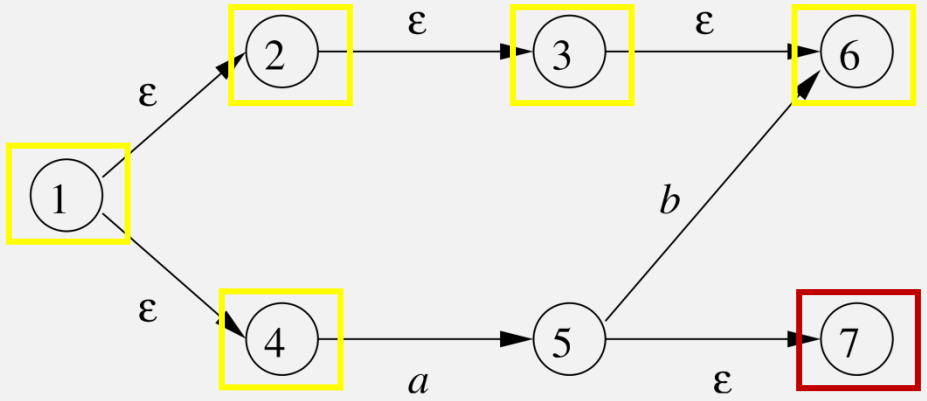
- Recursive case:

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

A state is in the reachable set if …

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

ε-REACHABLE Example



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

Handling ε transitions now!

NFA Multi-Step Transition Function

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

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

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

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

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

(Defined recursively)

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

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

Handling ε transitions now!

NFA Multi-Step Transition Function

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

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

All chars except last

"second to last" set of states

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

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

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

"last" set of states (no ε)

(Defined recursively)

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

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

Summary: NFA vs DFA Computation

DFAs

- Can only be in one state
- Transition:
  - Must read 1 char
- Acceptance:
  - If final state is accept state

NFAs

- Can be in multiple states
- Transition
  - Has empty transitions
- Acceptance:
  - If one of final states is accept state

Previously

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

Is Concatenation Closed?

THEOREM

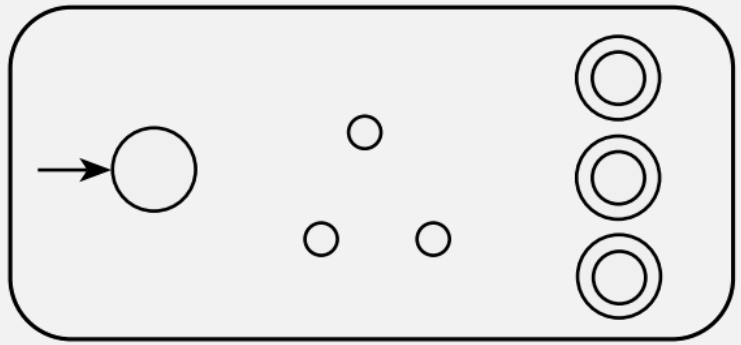
The class of regular languages is closed under the concatenation operation.

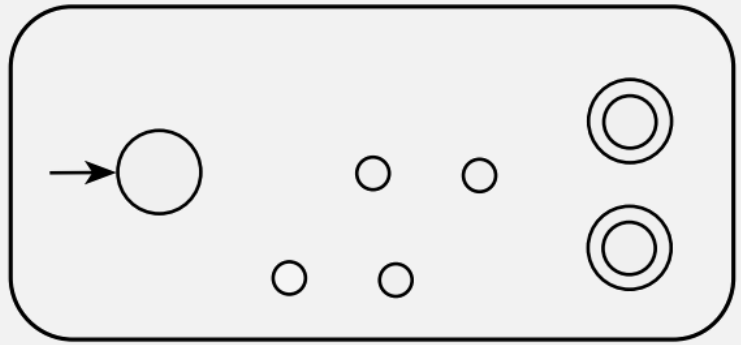
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

Proof requires: Constructing new machine

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

Concatentation





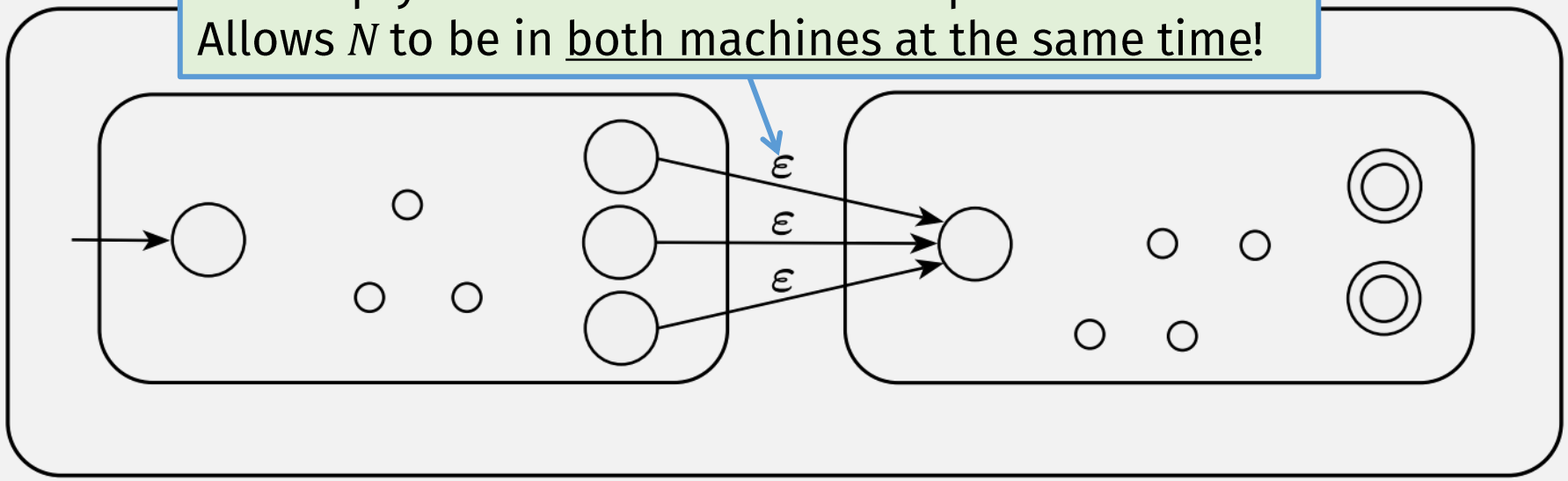
Let M_1 recognize A_1, and M_2 recognize A_2.

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

N is an **NFA**! It can:
- Keep checking 1st part with M_1
  and
- Move to M_2 to check 2nd part

N

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



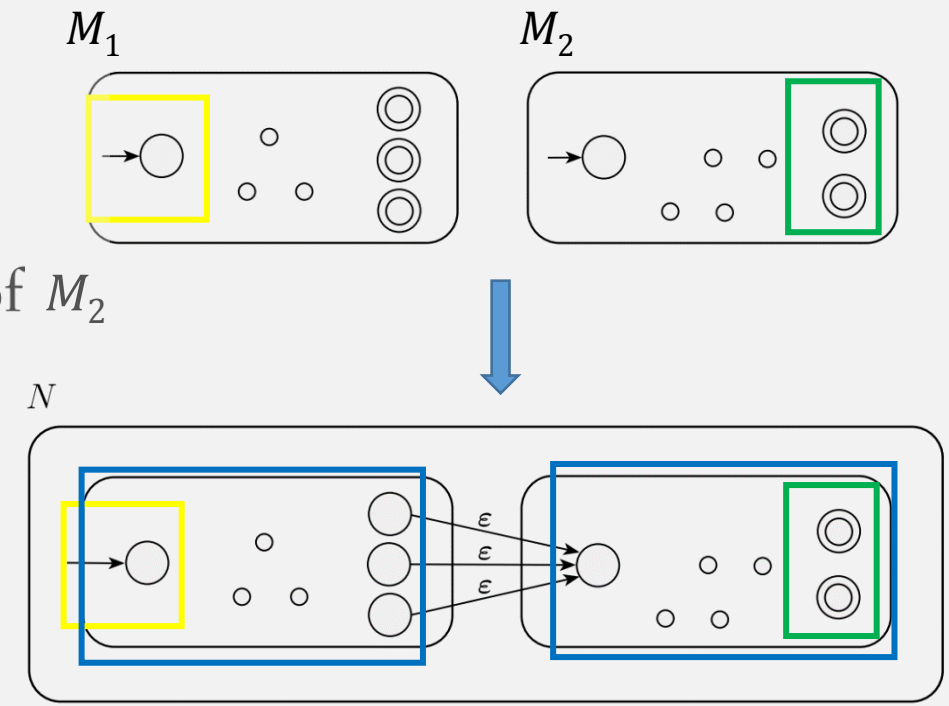
Concatenation is Closed for Regular Langs

PROOF (part of)

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

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

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



Concatenation is Closed for Regular Langs

PROOF (part of)

Wait, is this true?

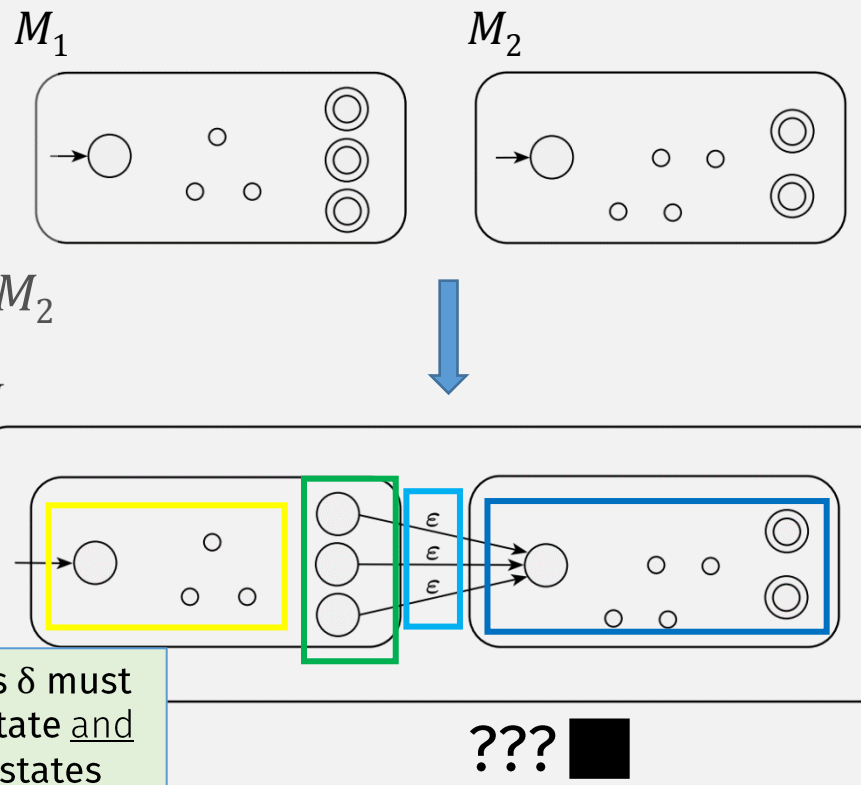
Let DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

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

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

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

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
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1	2. Def of Reg Lang (Coro)
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2	3. Def of Reg Lang (Coro)
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. See Examples Table
6. $A_1 \cup A_2$ is a regular language	6. Def of Regular Language
7. The class of regular languages is closed under the union operation. In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.	7. From stmt #1 and #6

Q.E.D. ■

Is Concat Closed For Regular Langs?

Proof?

Statements	**Justifications**
1. A_1 and A_2 are regular languages	1. Assumption
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1	2. Def of Reg Lang (Coro)
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2	3. Def of Reg Lang (Coro)
4. Construct NFA $M = (Q, \Sigma, \delta, q_0, F)$ ☑	4. Def of NFA
5. M recognizes ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$	5. See Examples Table
6. ~~$A_1 \cup A_2$~~ $A_1 \circ A_2$ is a regular language	6. ??? Does NFA recognize reg langs?
7. The class of regular languages is closed under (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

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 concatenation is closed …

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

Specifically, we must prove:

NFAs ⇔ regular languages

“If and only if” Statements

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

Represents <u>two</u> statements:

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

2. $\Leftarrow$ if Y, then X
   - “**reverse**” direction

How to Prove an "iff" Statement

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

Proof has <u>two</u> (If-Then proof) parts:

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

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof: 2 parts

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

(Easier)

- We know: if L is **regular**, then a **DFA** exists that recognizes it.
- So to prove this part: Convert that DFA → an equivalent NFA! (see HW 3)

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

Full Statements & Justifications?

"equivalent" = "recognizes the same language"

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

Statements	**Justifications**
1. L is a regular language	1. Assumption
2. A DFA M recognizes L	2. Def of Regular lang (Coro)
3. Construct NFA N = `convert`(M)	3. See hw ~~2~~ 3!
4. DFA M is **equivalent** to NFA N	4. See Equiv. table!
5. An NFA N recognizes L	5. ???
6. If L is a regular language, then some NFA N recognizes it	6. By Stmts #1 and # 5

Assume the "if" part ...

... use it to prove "then" part

“Proving” Machine Equivalence (Table)

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

NFA N = convert(M)

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

Note:
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

$$r_1 = q_0 \text{ and } r_n \in F$$

Then N accepts?/rejects? w because …

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

“Proving” Machine Equivalence (Table)

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

NFA $N = \text{convert}(M)$

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

$\hat{\delta}(q_0, w') \notin F$ 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 …

Then N accepts?/rejects? w' because …

Justification #2?

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

☑ ⇒ If L is regular, then some NFA N recognizes it. (Easier)

- We know: if L is **regular**, then a **DFA** exists that recognizes it.
- So to prove this part: Convert that DFA → an equivalent NFA! (see HW 3)

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

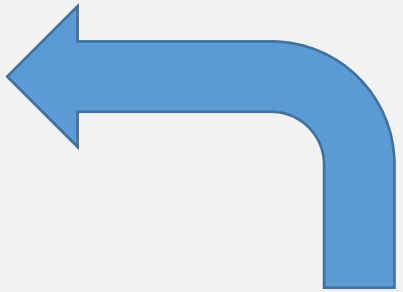
- We know: for L to be **regular**, there must be a **DFA** recognizing it
- Proof Idea for this part: Convert given NFA N → an equivalent DFA

"equivalent" =
"recognizes the same language"

How to convert **NFA→DFA**?

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

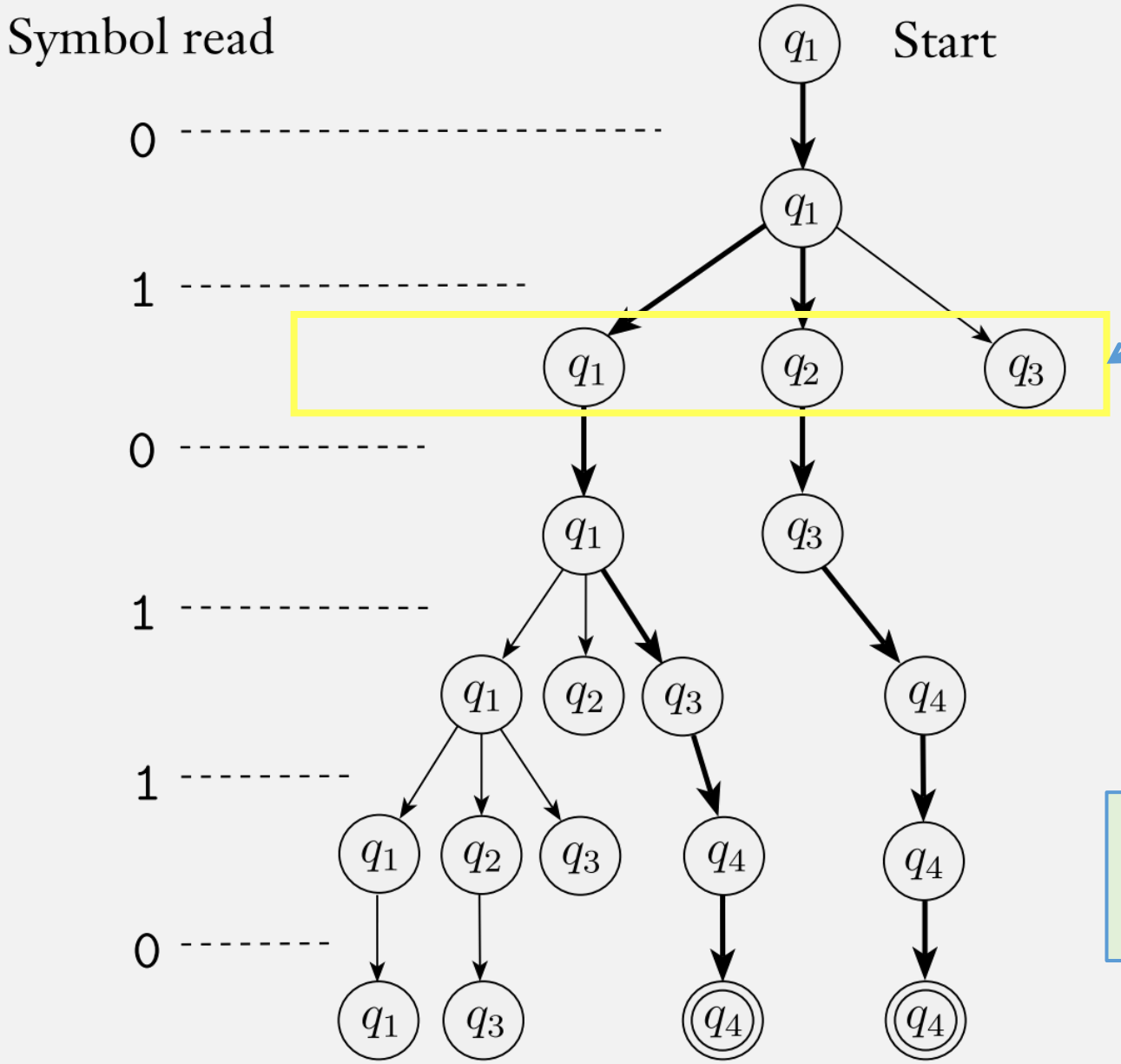
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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>

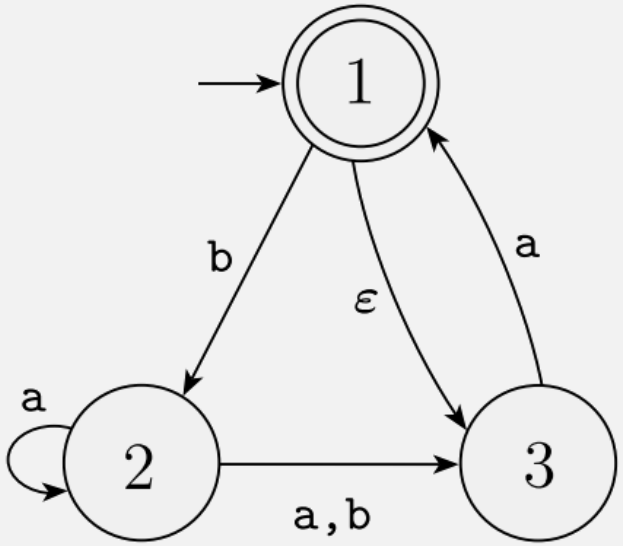
This is similar to the proof strategy from **"Closure of union"** where:
a state = a pair of states

Convert **NFA→DFA**, Formally

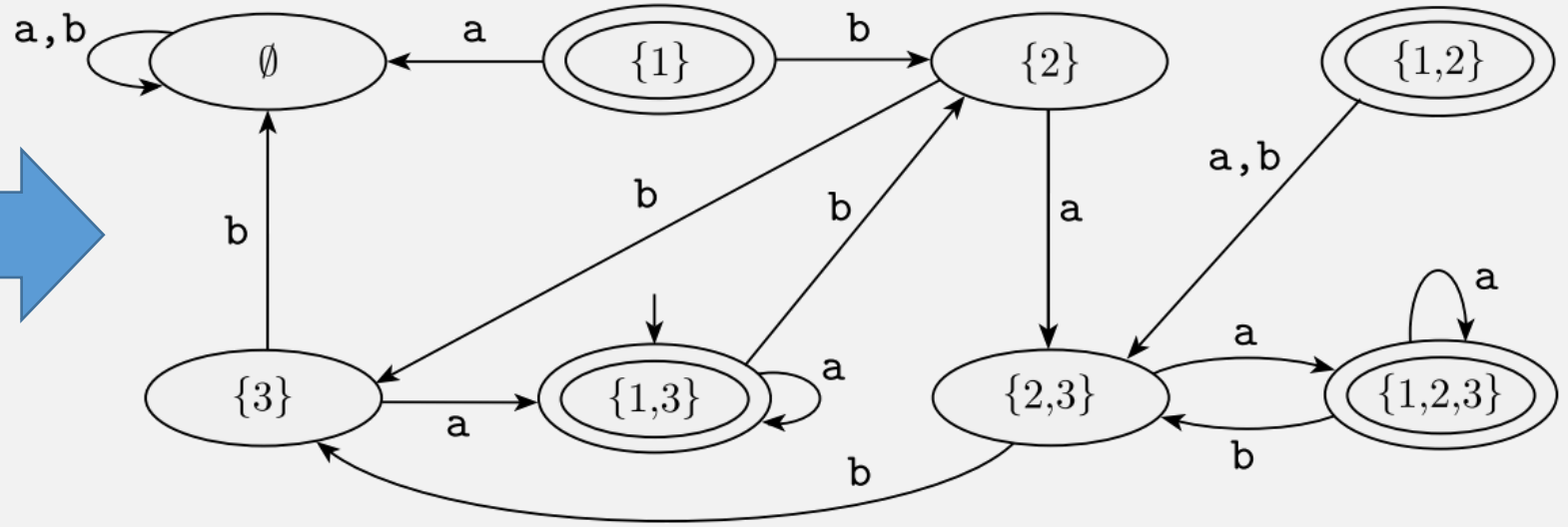
- Let **NFA** N = $(Q, \Sigma, \delta, q_0, F)$
- **An equivalent DFA** M **has states** Q' = $\mathcal{P}(Q)$ (power set of Q)

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 $qs \subseteq Q_{DFA}$ 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 3)

⇐ 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 ☑

Wait, is this true?

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

PROOF

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

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

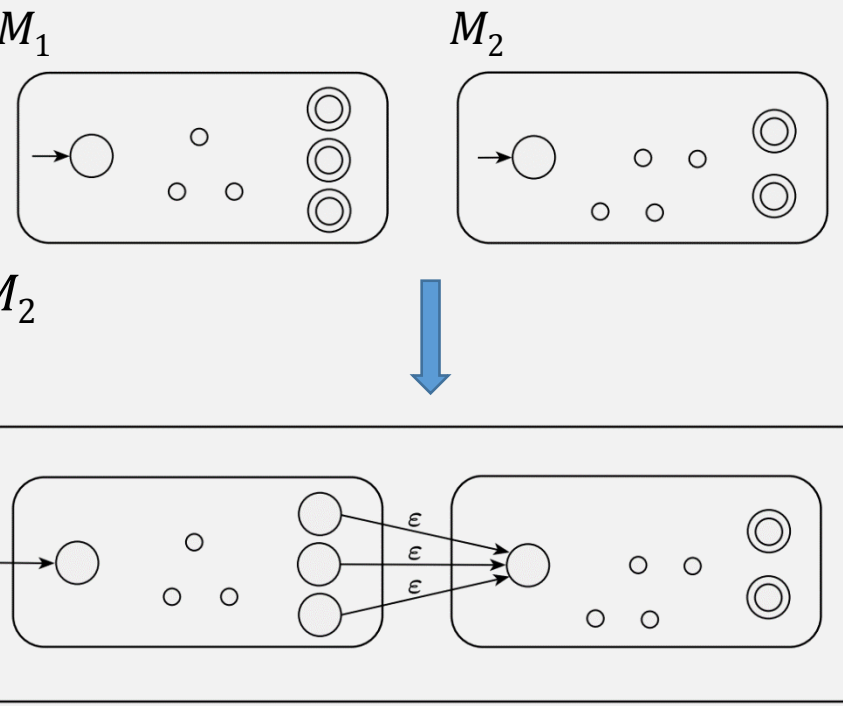
1. $Q = Q_1 \cup Q_2$

2. The state q_1 is the same as the start state of M_1

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

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

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



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

New possible proof strategy!

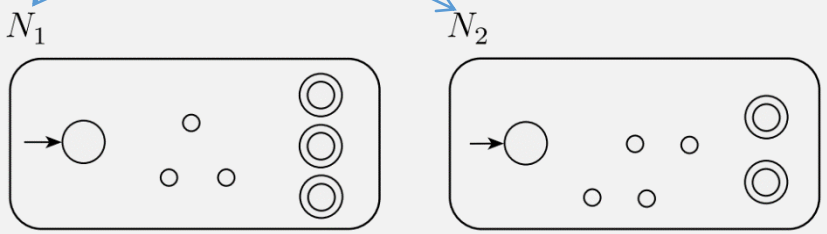
Concat Closed for Reg Langs: Use NFAs Only

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.

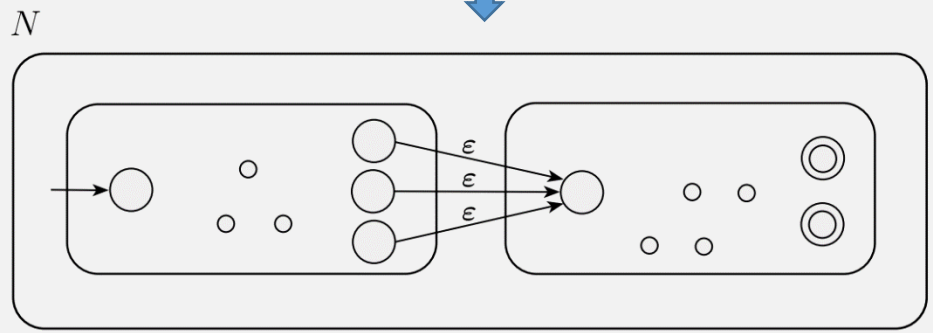
N = CONCAT$_{\text{NFA}}$ (N_1,N_2) $\Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$



1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of N_1
3. The accept states F_2 are the same as the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

????

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



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

Flashback: Union is Closed For Regular Langs

THEOREM

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

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

Proof:

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

Flashback: Union is Closed For Regular Langs

Proof

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

State in M = M_1 state + M_2 state

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

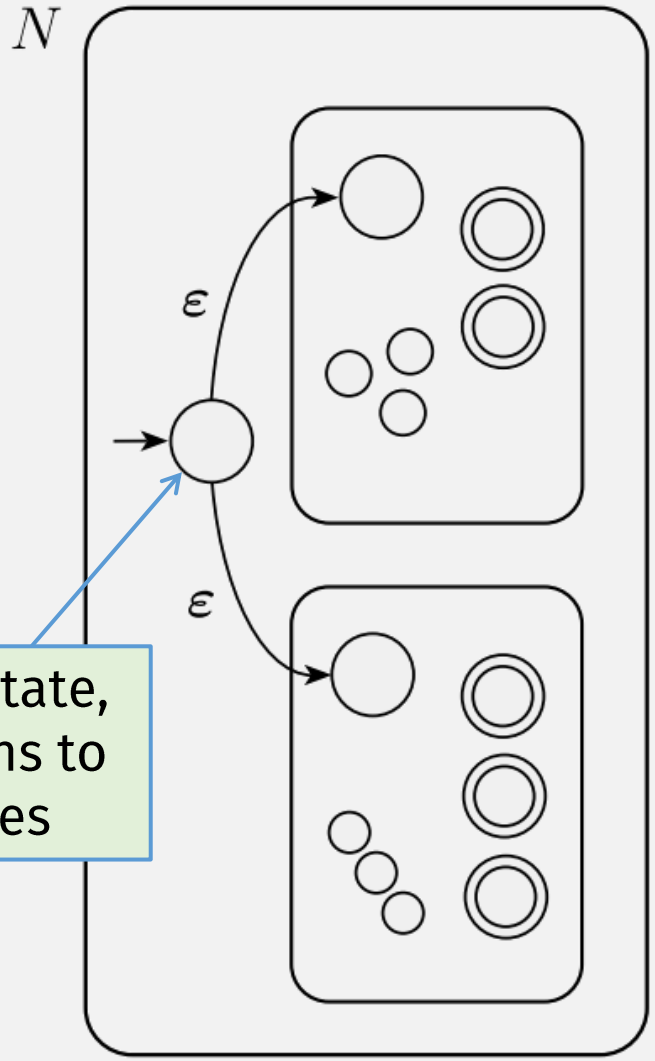
M step = a step in M_1 + a step in M_2

- M start state: (q_1, q_2)

Accept if either M_1 or M_2 accept

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

Union is Closed for Regular Languages



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

Alternate Proof, with NFAs

Union is Closed for Regular Languages

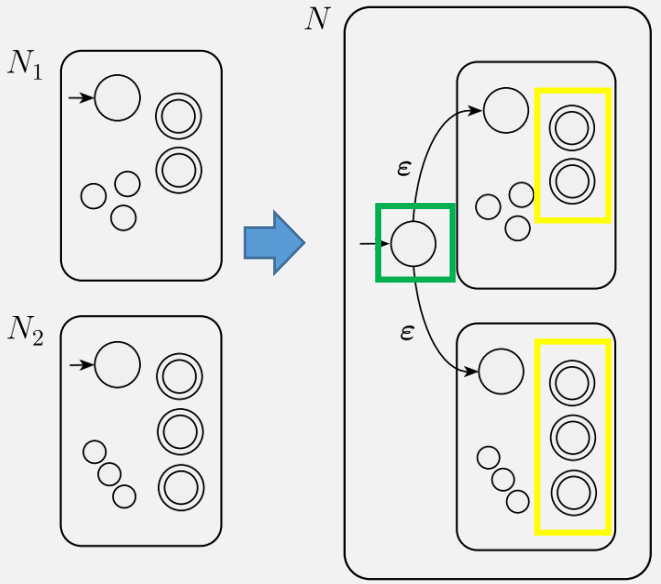
PROOF

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.

$N = \text{UNION}_{\text{NFA}}(N_1, N_2)$

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 $F = F_1 \cup F_2$.



Alternate Proof, with NFAs

Union is Closed for Regular Languages

PROOF

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

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

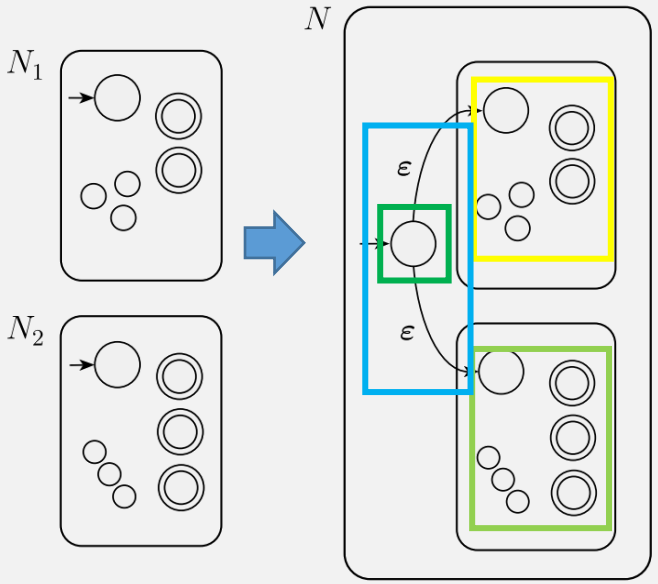
1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.

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

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

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 $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$.

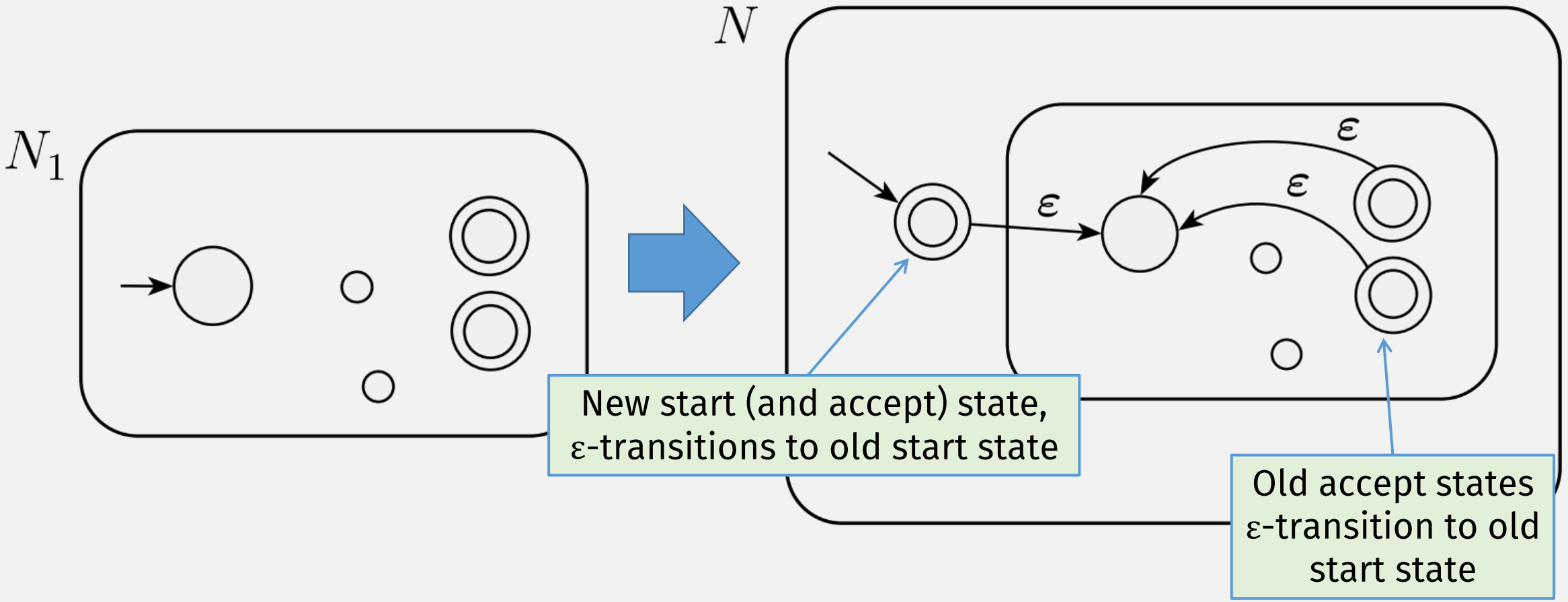
If $A = \{\mathtt{good}, \mathtt{bad}\}$

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

Note: repeat zero or more times

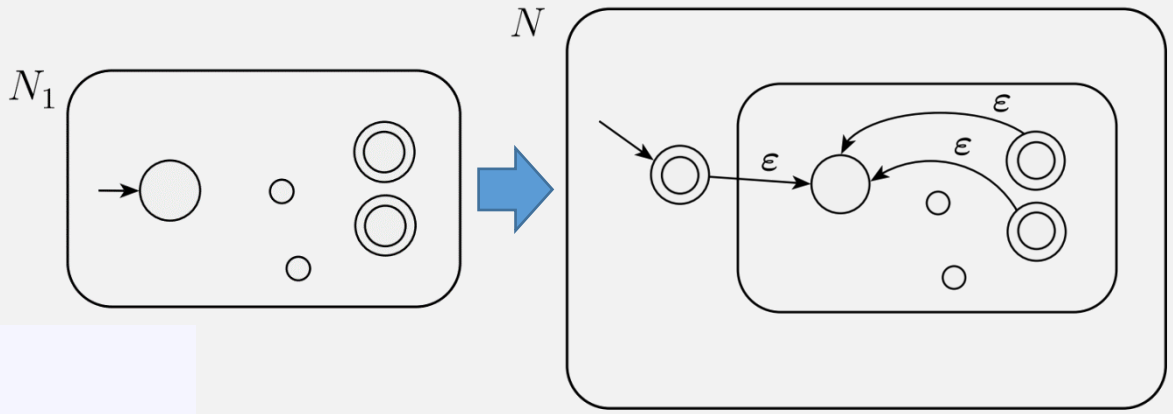
(this is an infinite language!)

Kleene Star


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

In-class exercise

Kleene Star is Closed for Regular Langs



THEOREM

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

In-class solution

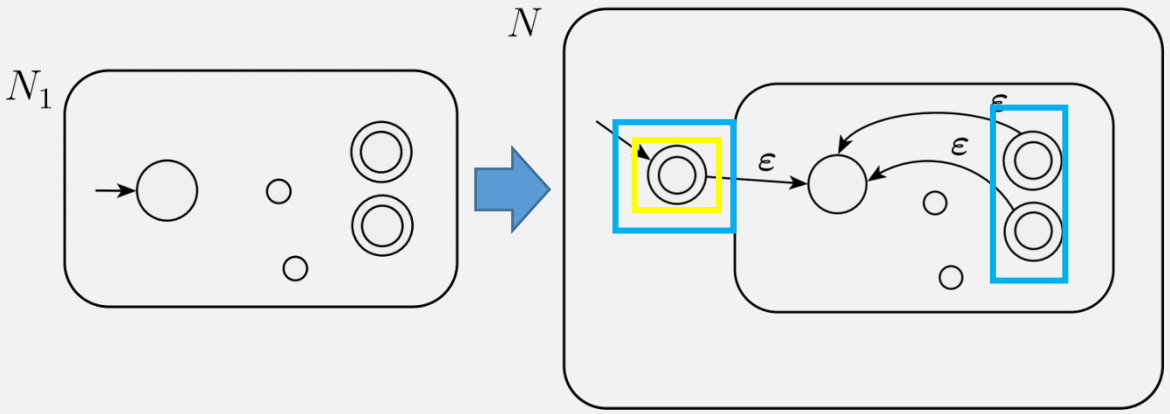
Kleene Star is Closed for Regular Langs

(part of)

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

$N = \text{STAR}_{\text{NFA}}(N_1)$ $\Sigma, \delta, q_0, F)$ 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!

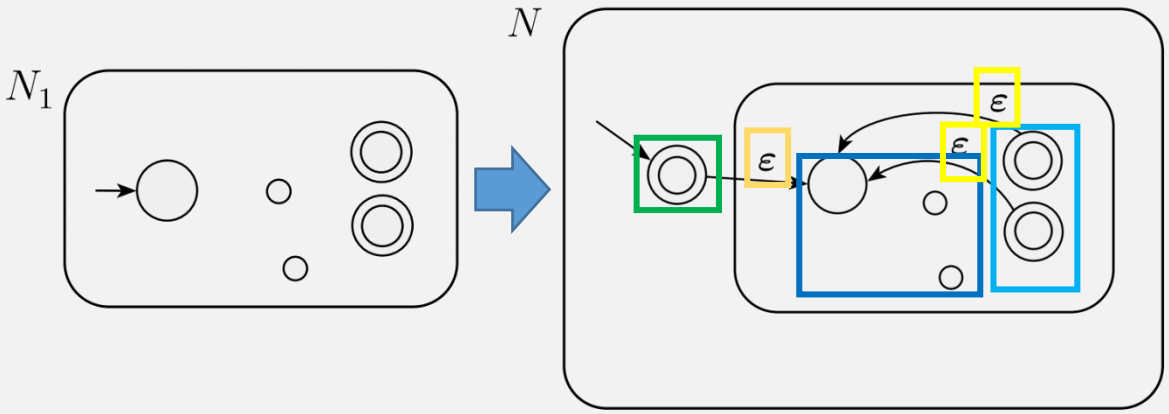
In-class solution

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.

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

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

Next Time: Why These Closed Operations?

- Union
- Concat
- Kleene star

All regular languages can be constructed from:
- single-char strings, and
- these three combining operations!

List of Closed Ops for Reg Langs (so far)

☑ • Union $A \cup B = \{x | \, x \in A \text{ or } x \in B\}$

☑ • Concatentation $A \circ B = \{xy | \, x \in A \text{ and } y \in B\}$

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

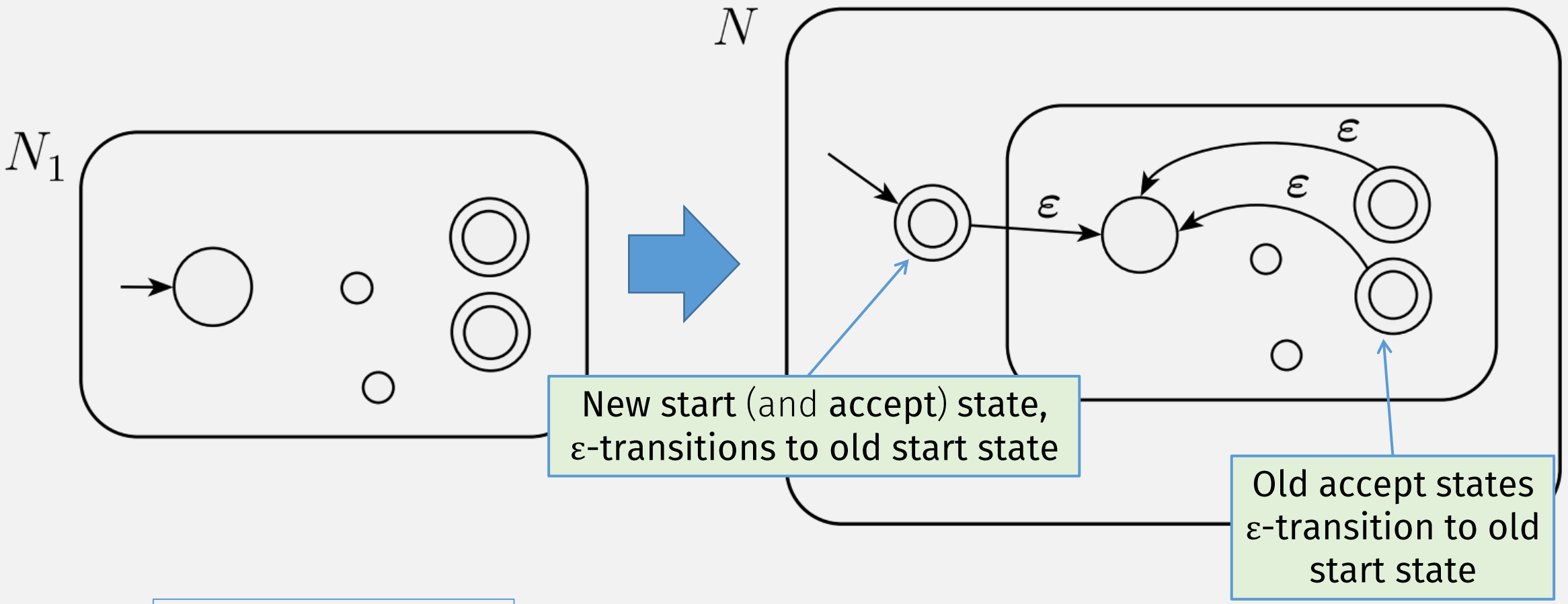
$$A^* = \{\varepsilon, \mathtt{good}, \mathtt{bad}, \mathtt{goodgood}, \mathtt{goodbad}, \mathtt{badgood}, \mathtt{badbad}, \mathtt{goodgoodgood}, \mathtt{goodgoodbad}, \mathtt{goodbadgood}, \mathtt{goodbadbad}, \ldots\}$$

Note: repeat zero or more times

(this is an infinite language!)

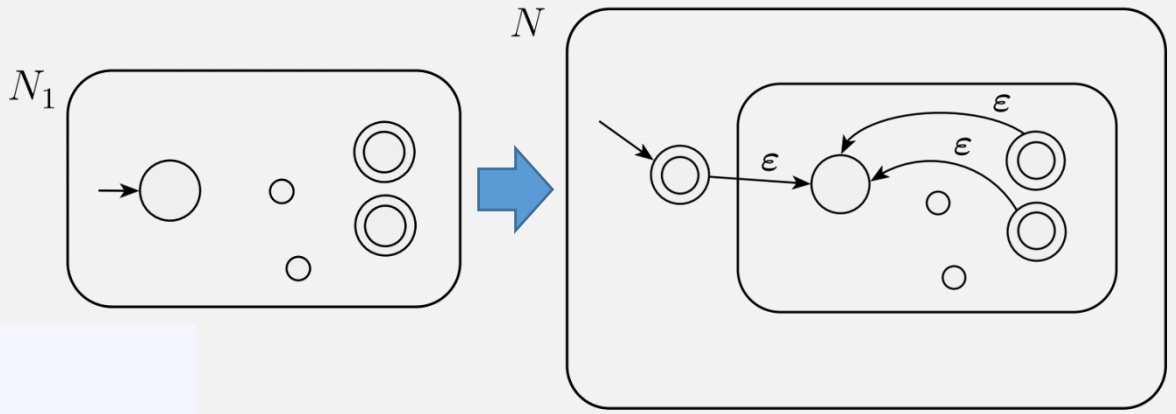
Star: $A^* = \{x_1 x_2 \ldots x_k | \; k \geq 0 \text{ and each } x_i \in A\}$

Kleene Star is Closed for Regular Langs?



HW4 exercise

Kleene Star is Closed for Regular Langs



THEOREM

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

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!**