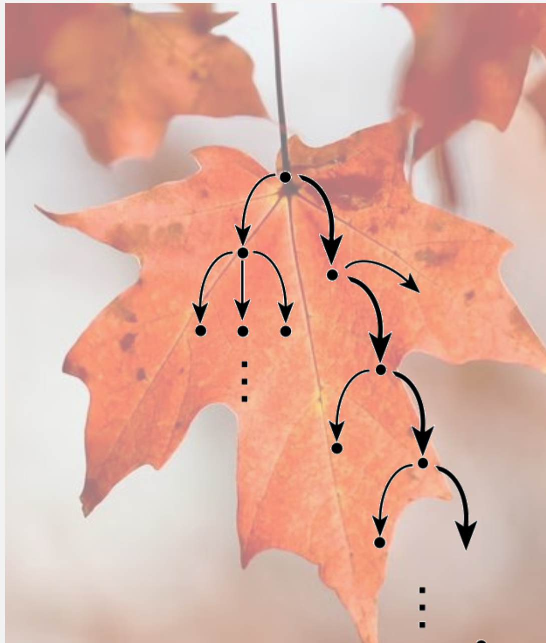
CS 420 / CS 620

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 <u>set of languages</u>** (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.

Want to prove this statement

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

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, $\boxed{\text{if } A_1 \text{ and } A_2 \text{ are regular languages, so is } A_1 \cup A_2}$.

Want to prove this statement

Or this (same) statement

Flashback: Mathematical Statements: **IF-THEN**

Using:

- If we know: $P \rightarrow Q$ 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: $P \rightarrow Q$ is TRUE:
  - Prove P is FALSE (**usually hard** or **impossible**)
  - Assume P is TRUE, then prove Q is TRUE

p	q	$p \rightarrow q$
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**

Stat Do we know anything about A_1 and A_2?

1. A_1 and A_2 are regular languages	1. Assumption of If part of If-Then
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 $A_1 \cup A_2$	
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 $A_1 \cup A_2$.	7. From stmt #1 and #6

Corollary

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

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

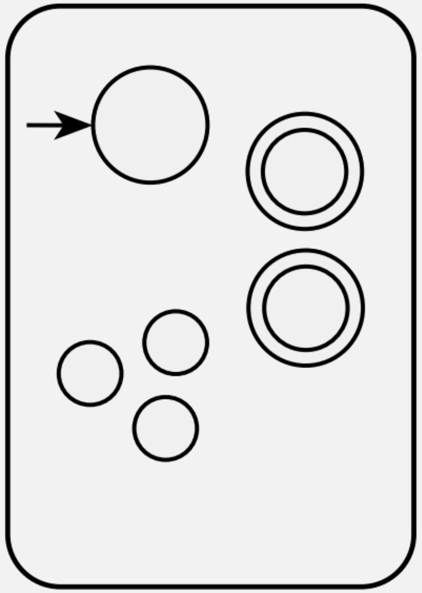
Is Union Closed For Regular Langs?

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. $\boxed{\text{Construct DFA } M = (Q, \Sigma, \delta, q_0, F) \text{ (todo)}}$	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. 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 M? Don't know what A_1 and A_2 are!

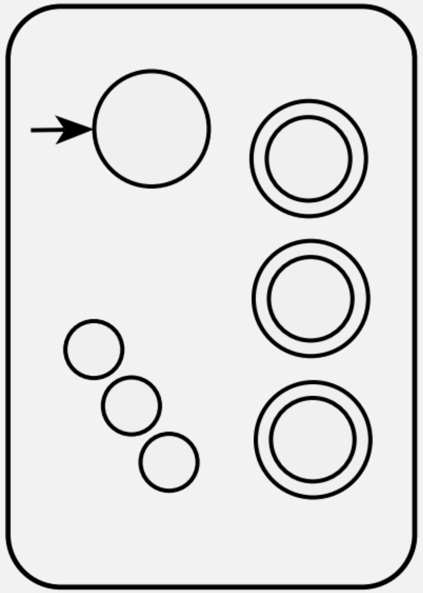
M_1

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

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>...

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

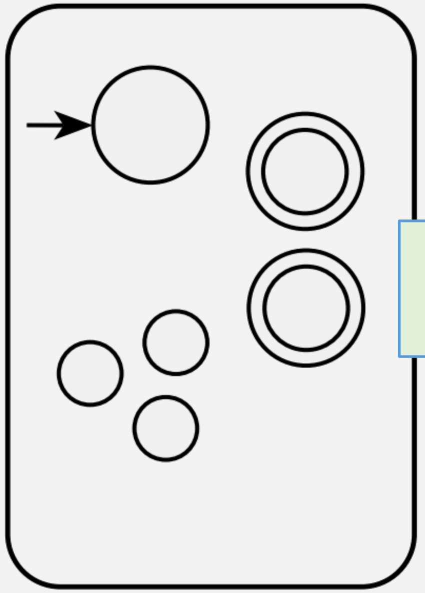
Definition of Regular Language (Corollary)

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

Union

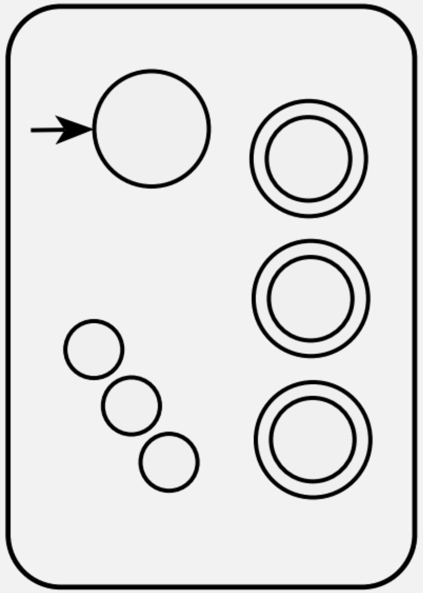
M_1

recognizes A_1



M_2

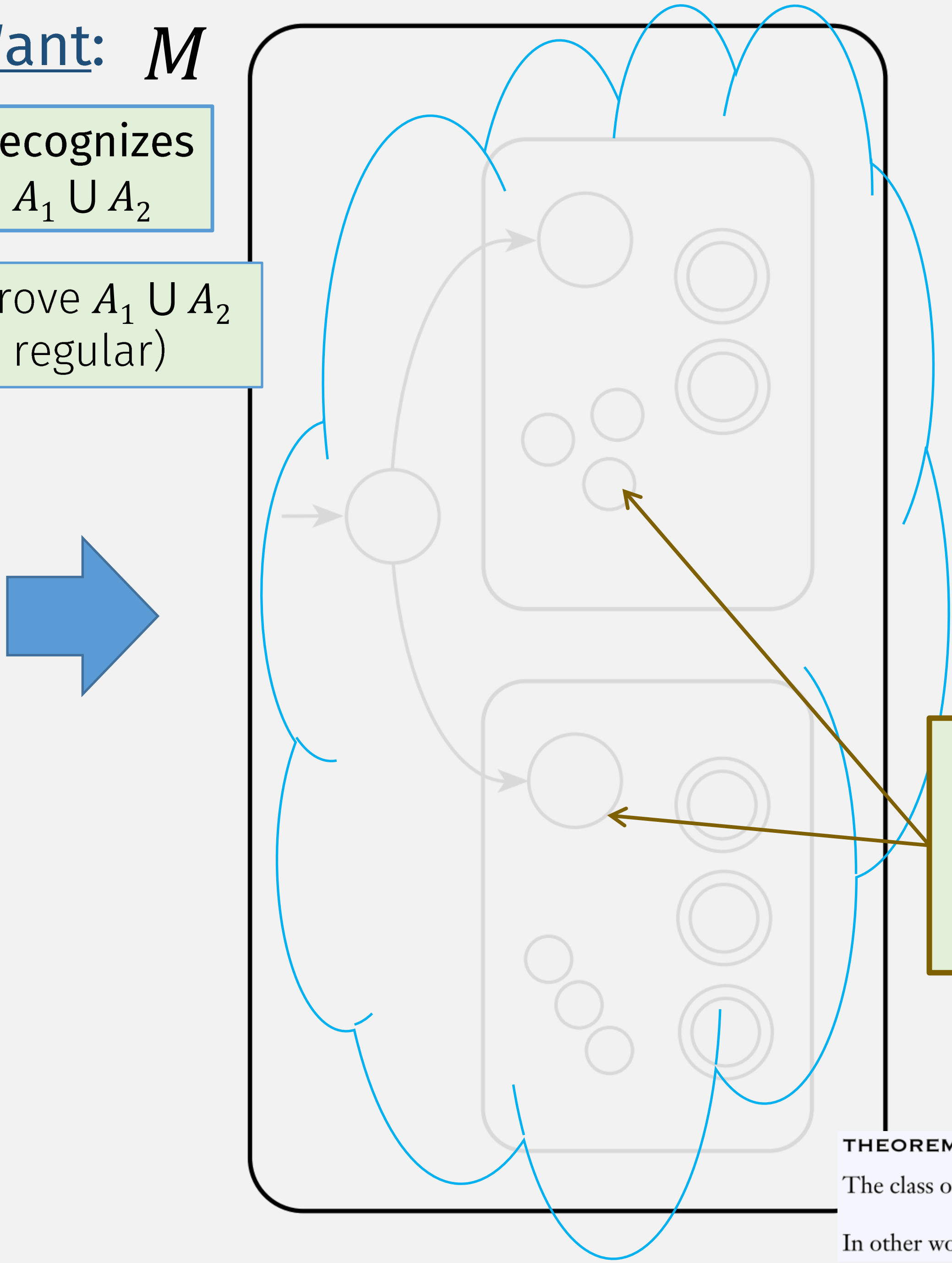
recognizes A_2



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

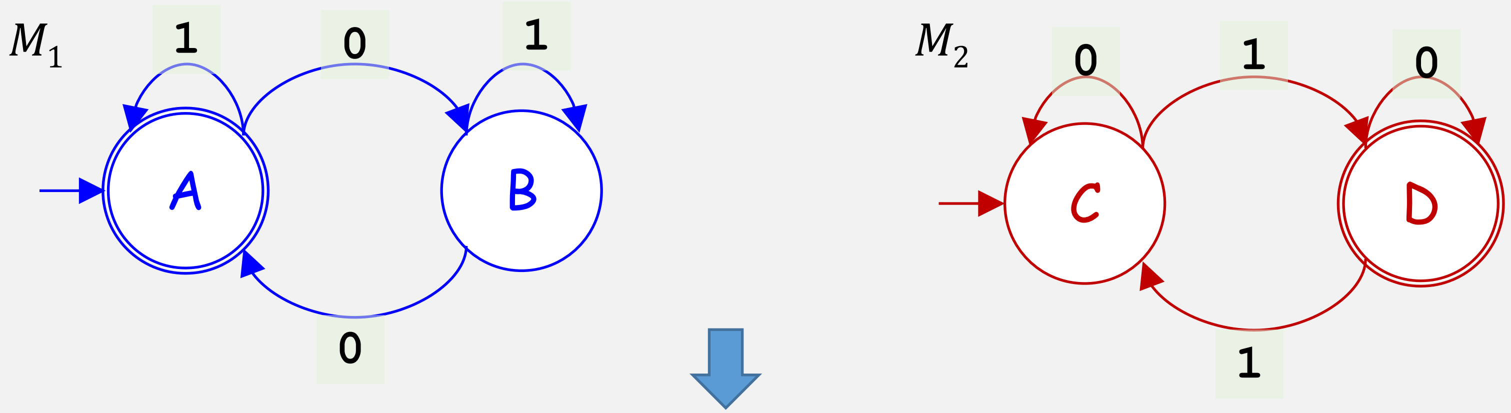
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

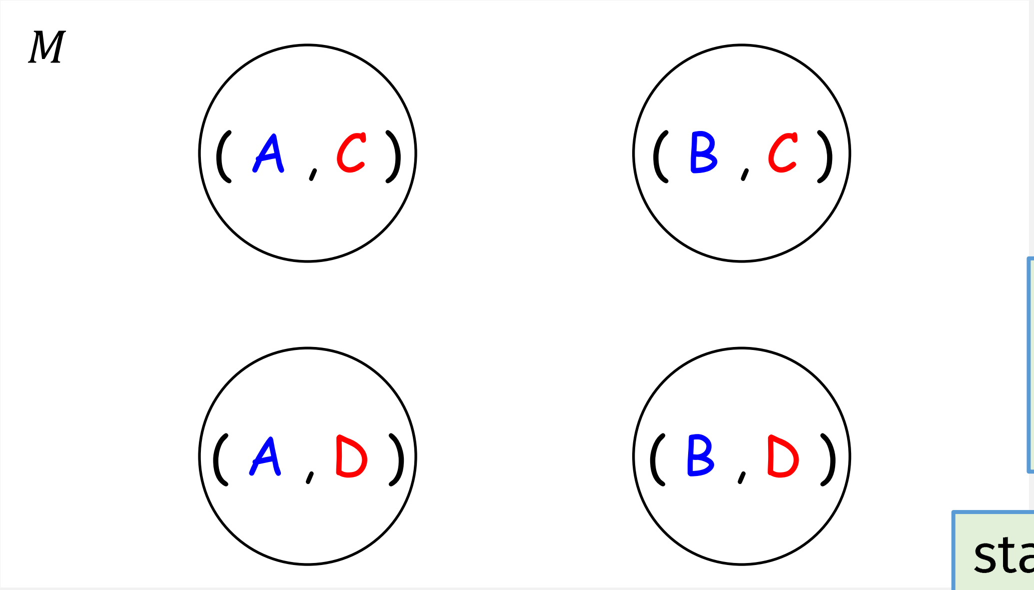
A **state** of M is a <u>**pair**</u>:
- <u>**first**</u> **part: state** of M_1
- <u>**second**</u> **part: state** of M_2

states of M:
<u>**all pair combos**</u> 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)

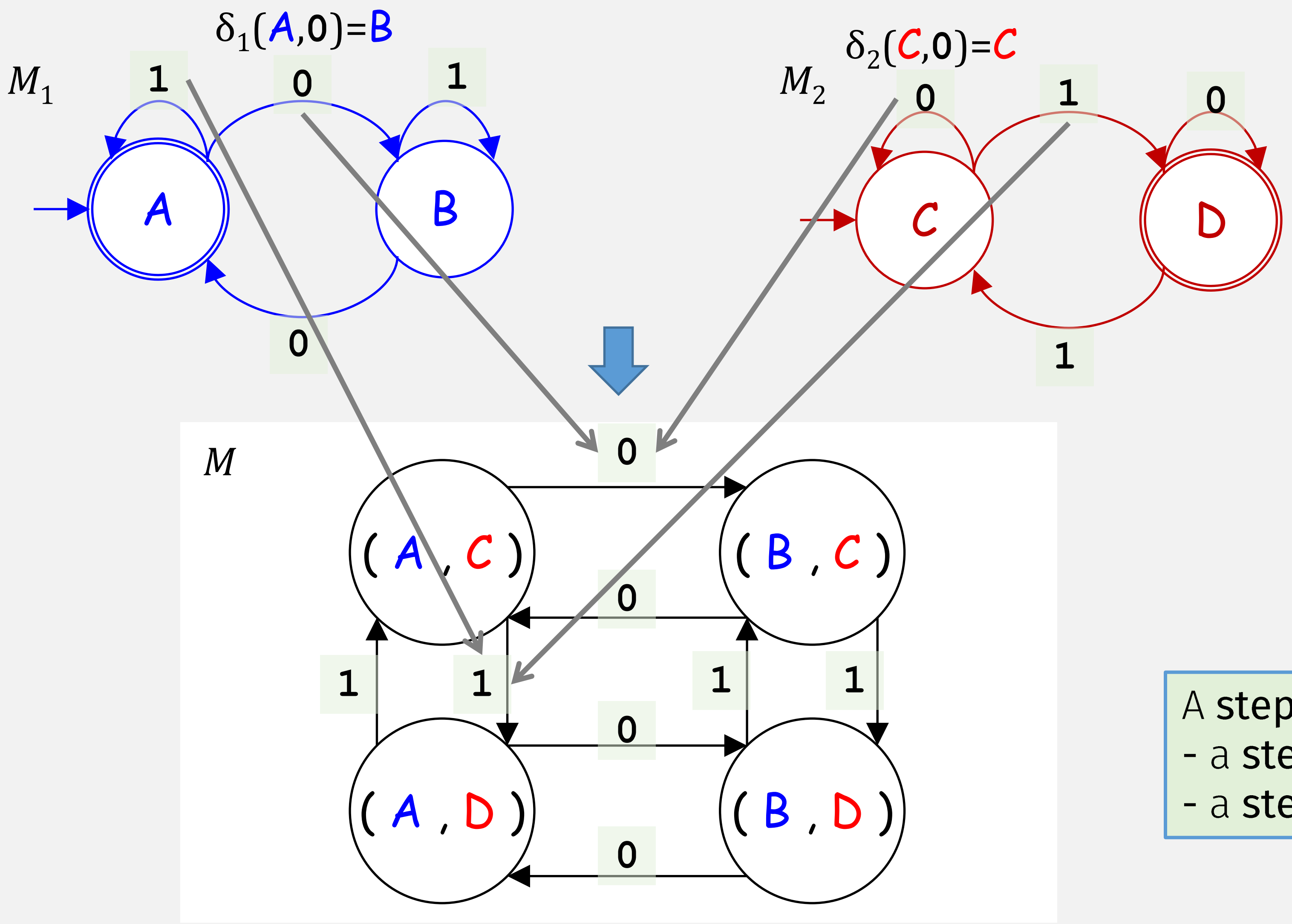
- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  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 <u>both</u>:
- 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. $\boxed{\delta \colon Q \times \Sigma \longrightarrow Q \text{ is the \textit{transition function},}}$[1]
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

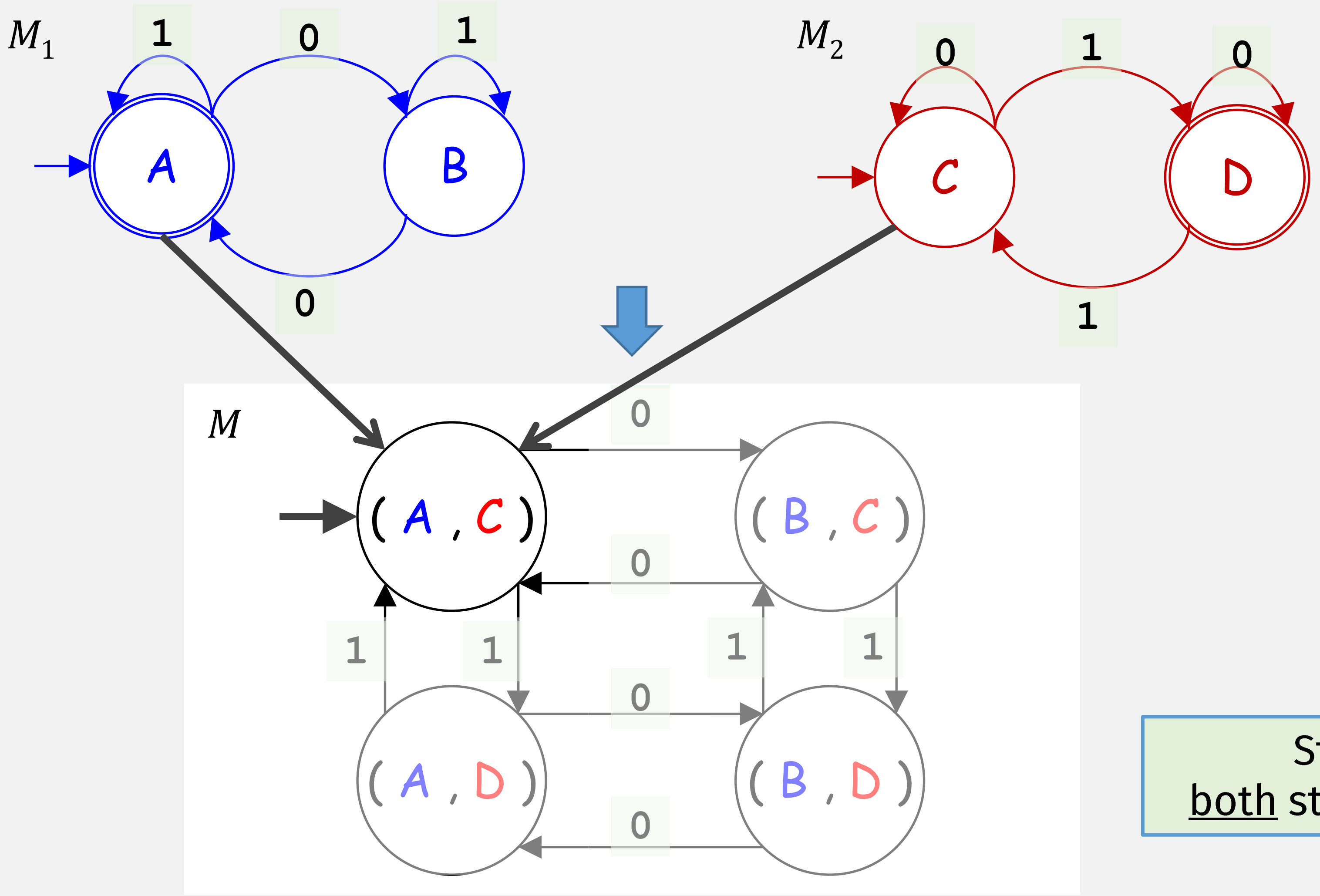
Union is Closed For Regular Languages

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: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$
- 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)

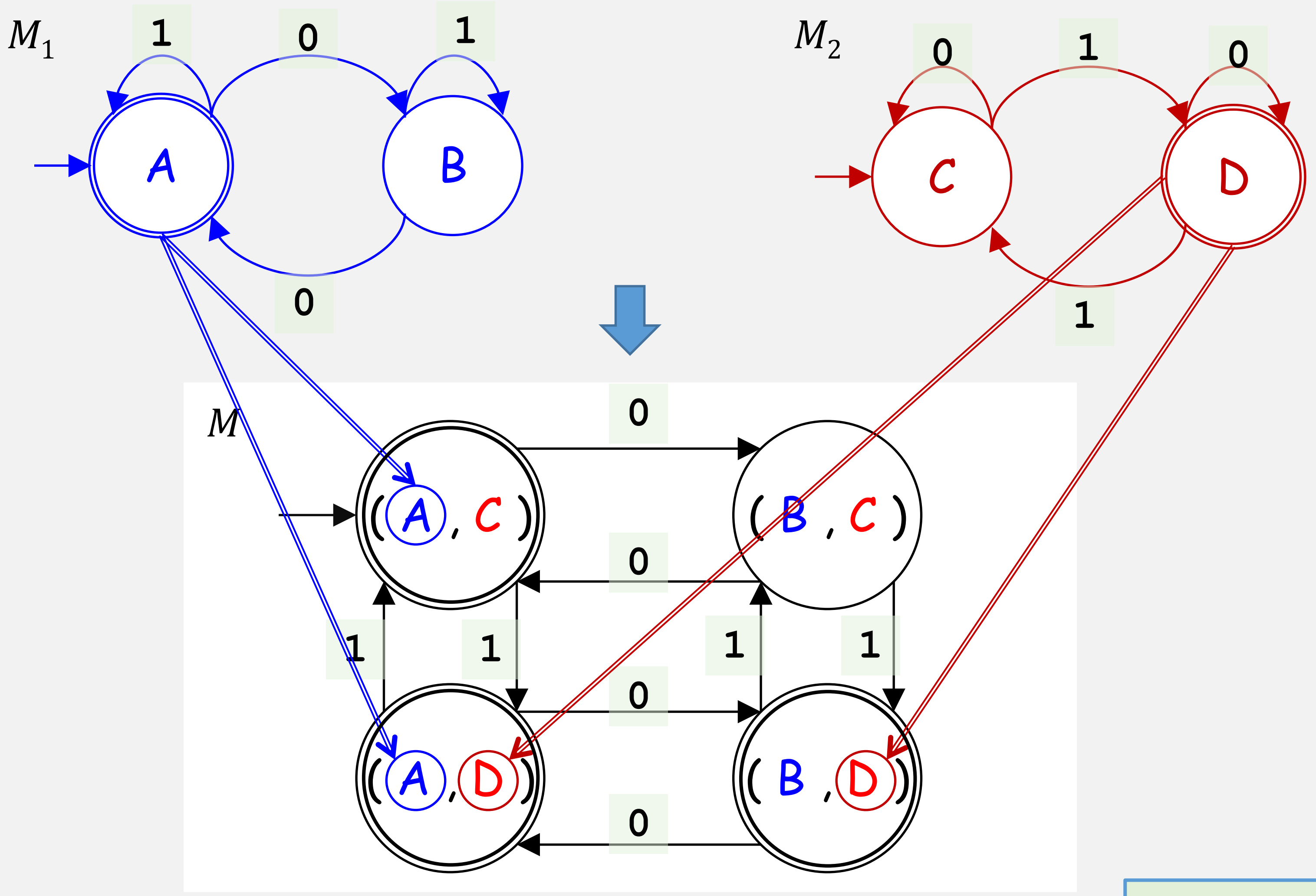
- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$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

Remember:
Accept states must be subset of Q

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

Accept if either M_1 or M_2 accept

DFA Union Example



Accept if <u>either</u> 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:

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

- states of M: $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- M transition fn: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$
- M start state: (q_1, q_2)
- M accept states: $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

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 $A_1 \cup A_2$ — How to create this? Don't know what A_1 and A_2 are!	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

"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 $A_1 \cup A_2$?	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 $A_1 \cup A_2$?

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,
- 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: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$
- M start state: (q_1, q_2)

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

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

"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 $\in A_{\boxed{2}} \rightarrow$ accepted by $M_{\boxed{2}} \rightarrow$ 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 <u>either</u> M_1 or M_2 accept | Else reject

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

Alternate Examples Table

"Prove" that DFA recognizes a language

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

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 lang $A_1 \cup A_2$?	Accepted by $M = \mathrm{UNION}_{\mathrm{DFA}}(M_1,M_2)$	n
s_1	Yes	Accept	(J1)
s_2	Yes	Accept	(J1)
$\cancel{s_3}$	~~???~~	???	
$\cancel{s_4}$	~~???~~	???	
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,

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

$s_1 \in A_1 \rightarrow$ 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

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

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

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

String	$\in A_1$?	$\in A_2$?	M_1 result?	M_2 result?	$\in A_1 \cup A_2$?	$M = \text{UNION}_{\text{DFA}}(M_1,M_2)$ 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

$s_1 \in A_1 \rightarrow$ 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

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

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

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

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

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

Previously

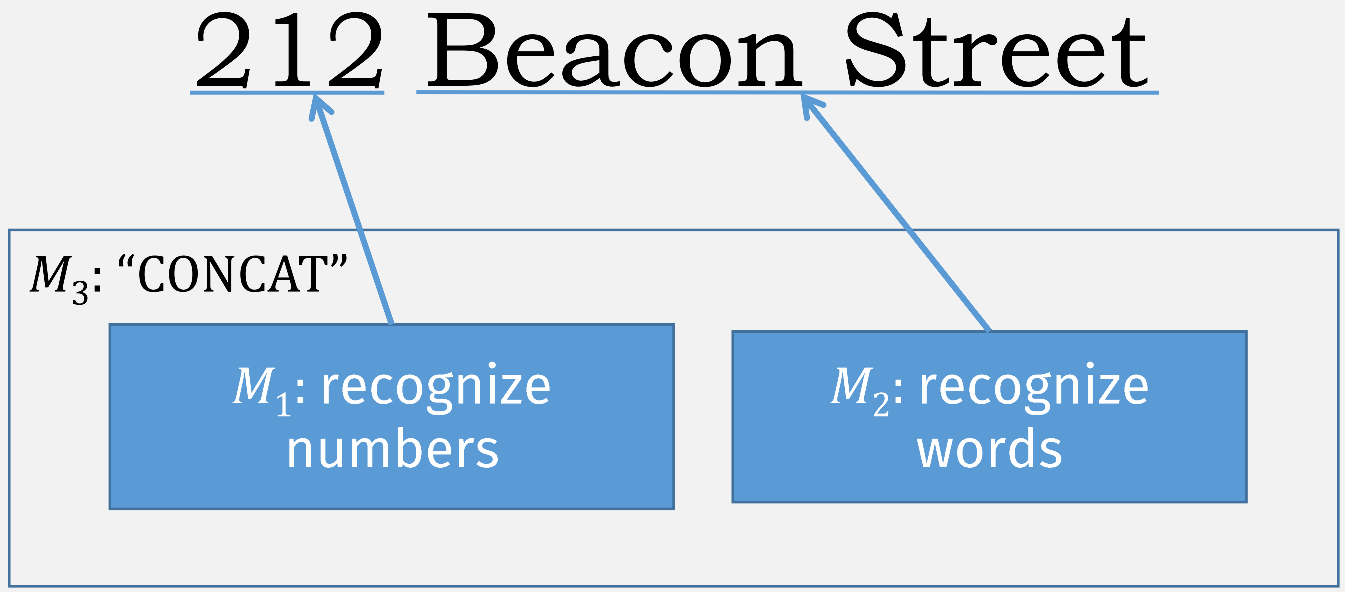
Is Union Closed For Regular Langs?

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

Concatenation of Languages

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

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

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

Is Concatenation Closed?

THEOREM

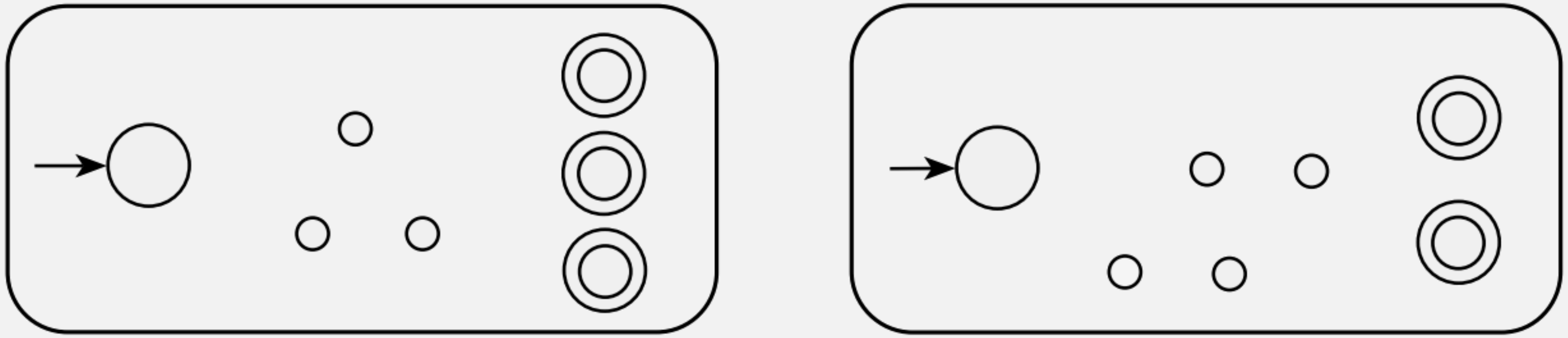
The class of regular languages is closed under the concatenation operation.

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

M_1 M_2

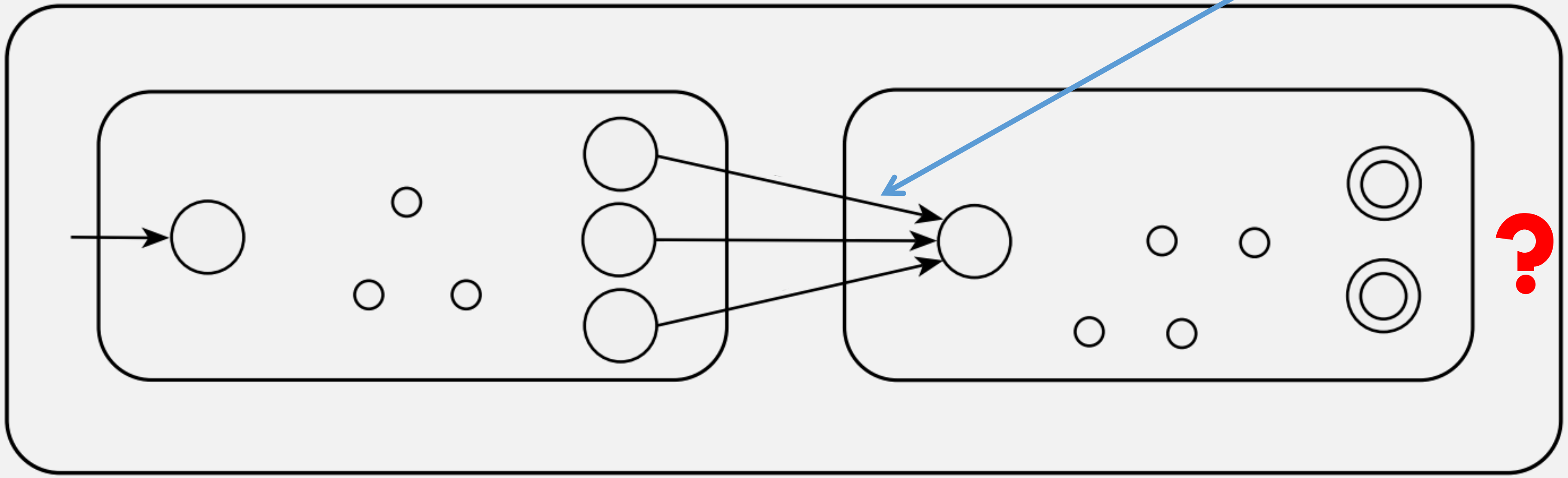


<u>PROBLEM</u>: Can only read input once, can't backtrack

Let M_1 recognize A_1, and M_2 recognize A_2.

<u>Want</u>: 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 $A \circ B$ = { `jensmith`, `jenssmith` }

- If M sees `jen` ...
- M must decide to <u>either</u>:

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

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 <u>either</u>:
  - stay in M_1 (correct, if full input is `jenssmith`)

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

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 }

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

- 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

Is Concatenation Closed?

FALSE?

THEOREM

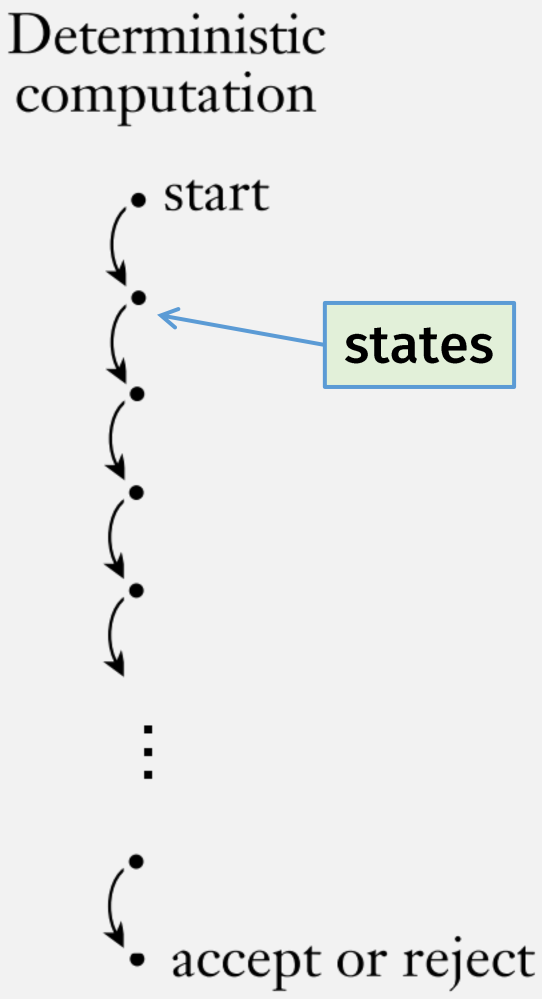
The class of regular languages is closed under the concatenation operation.

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?**

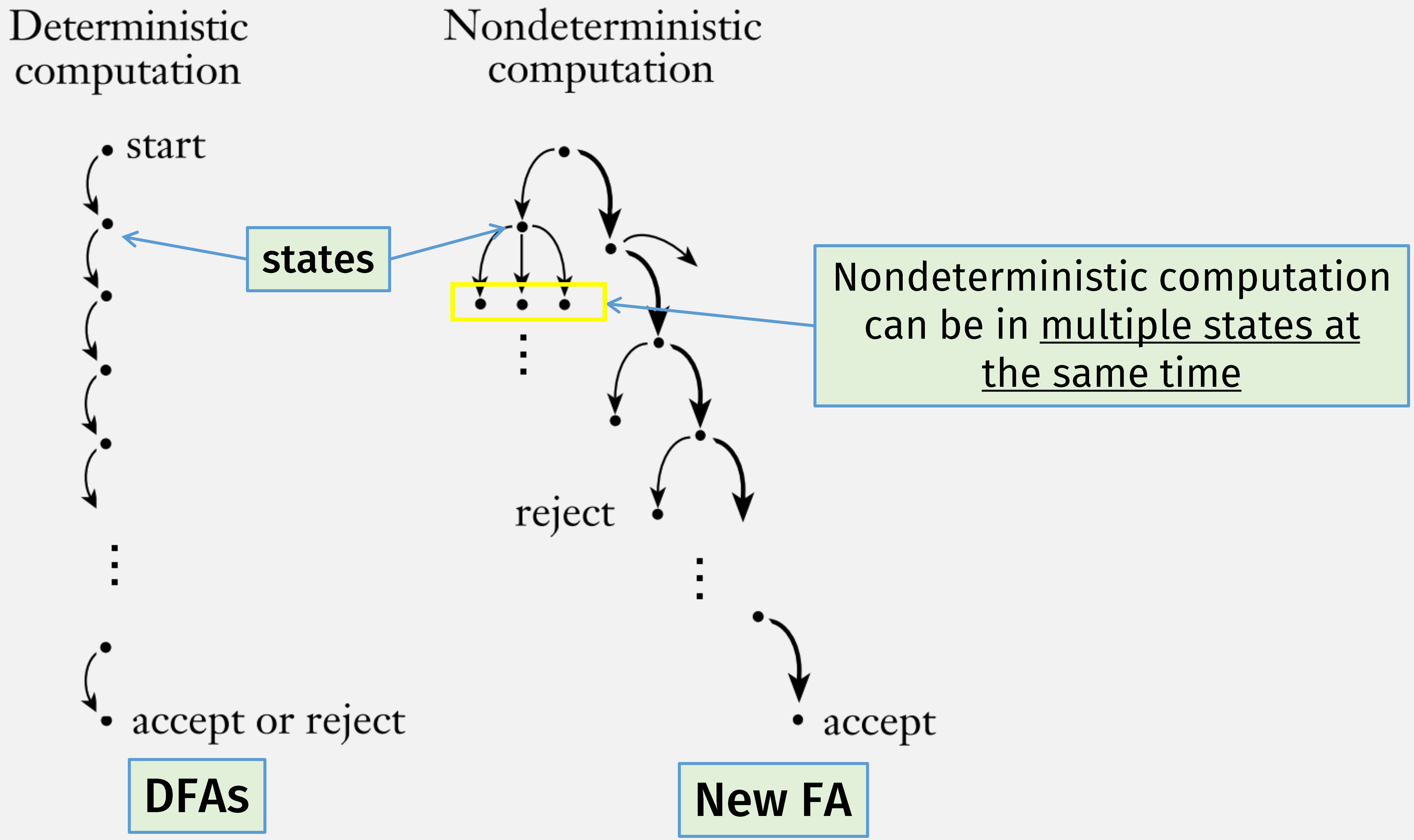
Nondeterminism

Deterministic vs Nondeterministic



DFAs

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

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. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Difference

Power set, i.e. a transition results in <u>set</u> 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. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

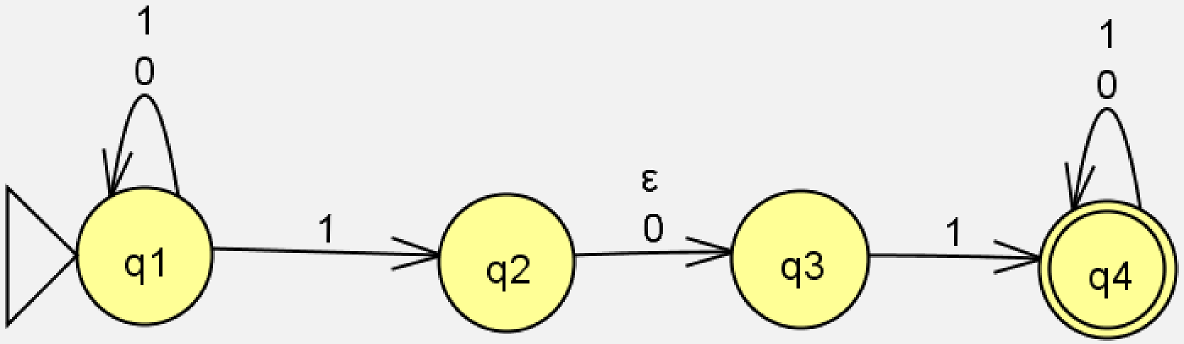
Transition label can be "empty", i.e., machine can transition without reading input

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

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. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

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

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

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

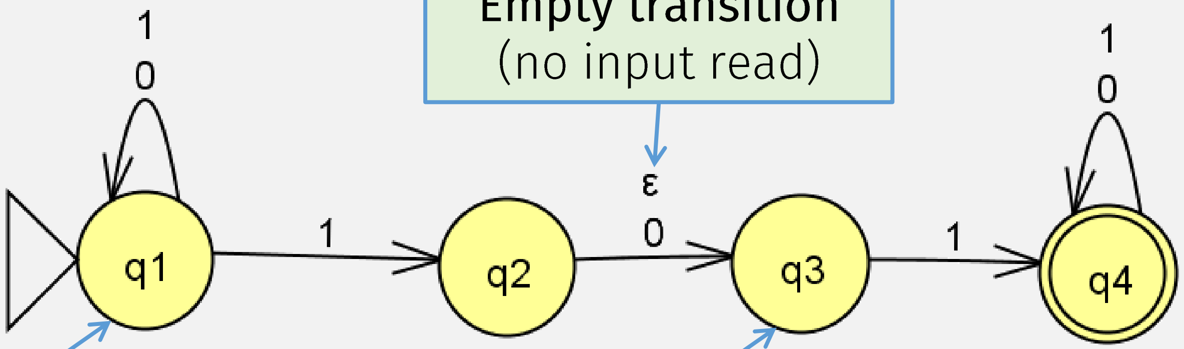
Empty transition (no input read)

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. $F = \{q_4\}$.

Empty transition (no input read)

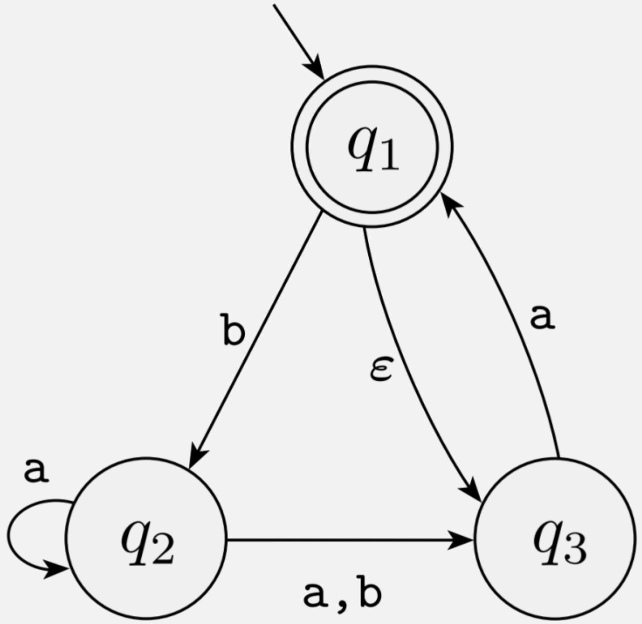


Multiple 1 transitions

No 0 transition

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. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

In-class Exercise Solution

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)

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

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

$\delta(q_1, \mathtt{a}) = \{ \}$

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

$\delta(q_1, \varepsilon) = \{ q_3 \}$

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

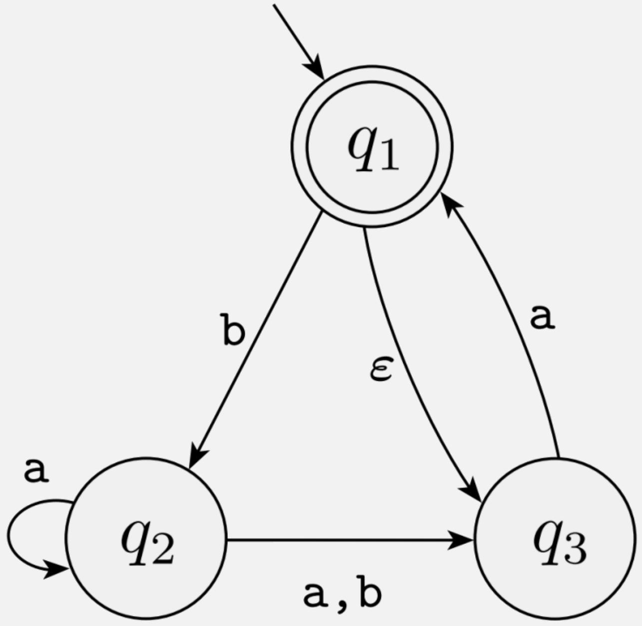
$\delta(q_2, \mathtt{b}) = \{ q_3 \}$

$\delta(q_2, \varepsilon) = \{ \}$

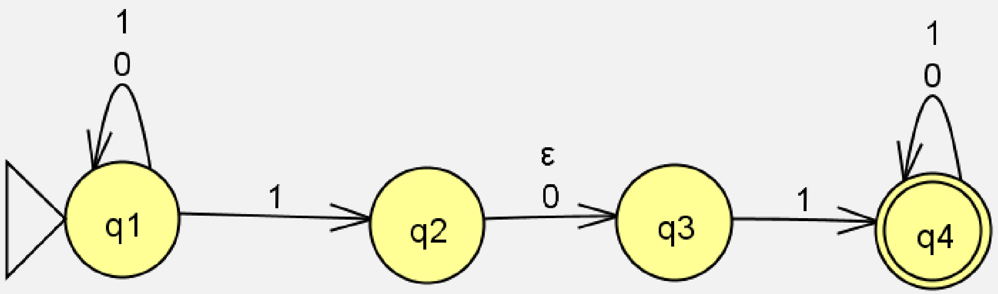
$\delta(q_3, \mathtt{a}) = \{ q_1 \}$

$\delta(q_3, \mathtt{b}) = \{ \}$

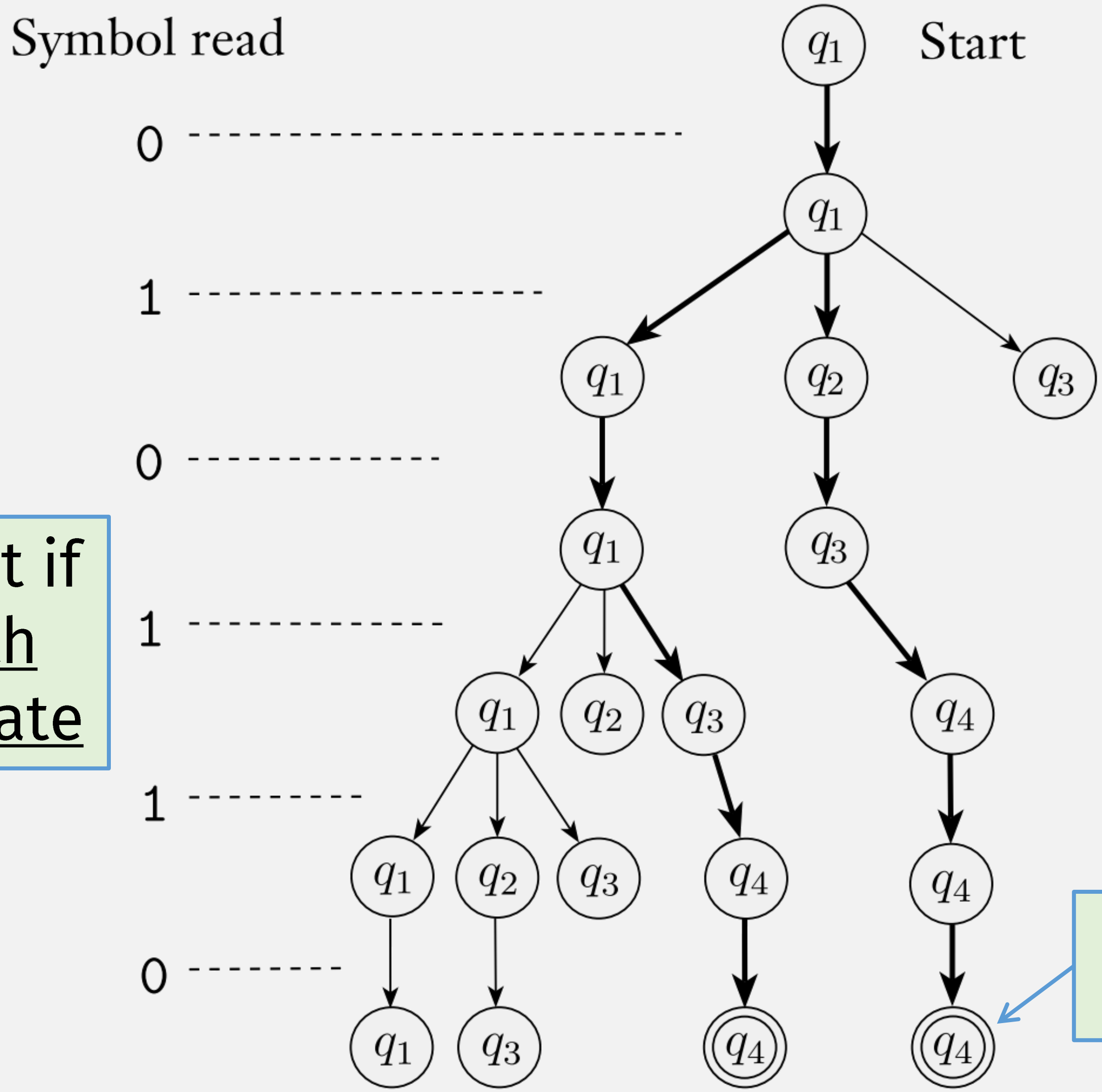
$\delta(q_3, \varepsilon) = \{ \}$



NFA Computation (JFLAP demo): `010110`



NFA Computation Sequence



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

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

So this is an **accepting computation**

Flashback

DFA Computation 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	*Formally (i.e., mathematically)*

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

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

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
  - For each "current" state, go to next states according to *transition rules*
  - ... 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

For each "current" state, go to next states according to *transition rules*

... 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

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

- 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)

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)

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

string

char

(Defined recursively)

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

Recursion on string

"smaller" argument

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

string

char

"second to last" state

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

Flashback

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

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)

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

(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

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

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

Recursively Defined Input
needs
Recursive Function

(Defined recursively)

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

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

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

(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

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

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

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 $qs \subseteq Q$

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

(Defined recursively)

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

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

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

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

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

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

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

Still ignoring ε transitions!

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

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

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

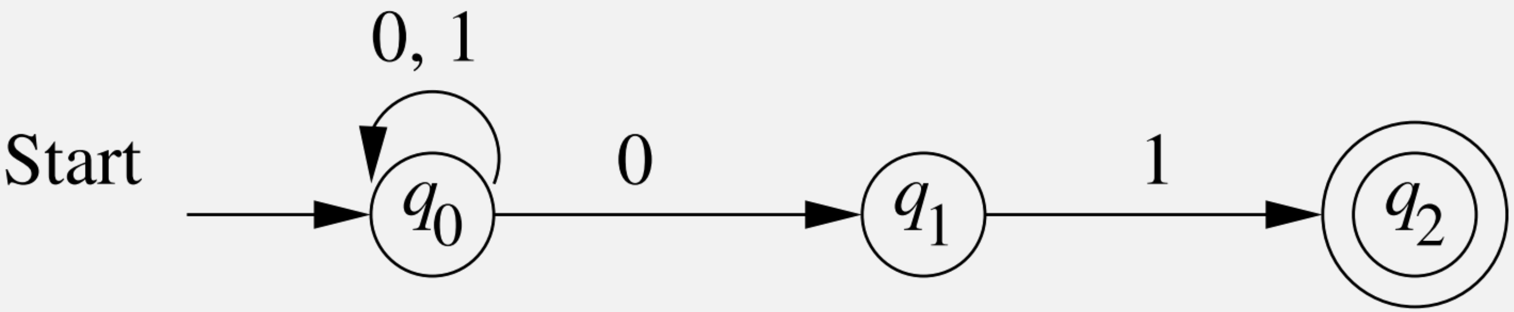
NFA Multi-Step δ Example

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 <u>zero or more empty transitions</u>

(Defined recursively)

- <u>Base</u> case: $q \in \varepsilon\text{-REACHABLE}(q)$

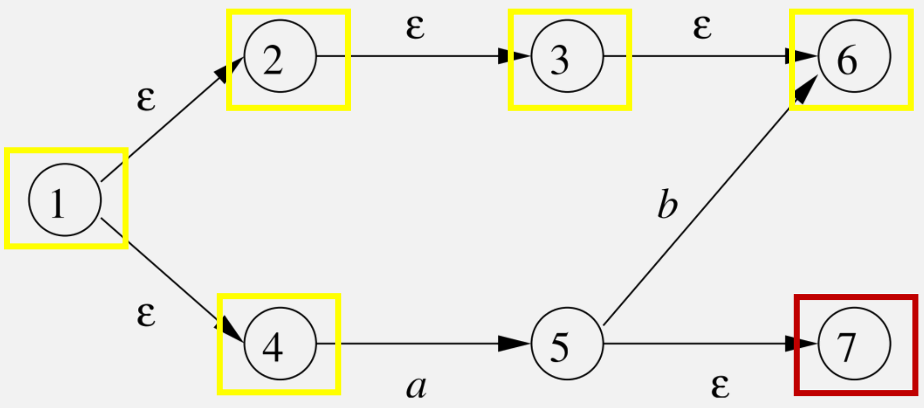
- <u>Recursive</u> case:

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

ε-REACHABLE Example



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

Handling ε transitions now!

NFA Multi-Step Transition Function

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

- 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 $qs \subseteq Q$

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

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

- 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 $qs \subseteq Q$

All chars except last

"second to last" set of states

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 <u>one</u> state
- Transition:
  - <u>Must read 1 char</u>
- Acceptance:
  - If final state <u>is</u> accept state

NFAs

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

Previously

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

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