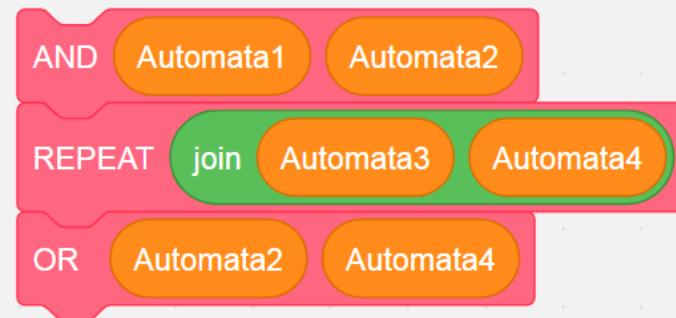


CS420

Combining Automata & Regular Languages

Monday, January 31, 2022

UMass Boston Computer Science



Announcements

- HW 0 in
- HW 1 out
 - Due Sun 2/6 11:59pm

Last Time: Alphabets, Strings, Languages

- An **alphabet** is a non-empty finite set of symbols

$$\Sigma_1 = \{0,1\}$$

$$\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$$

- A **string** is a finite sequence of symbols from an alphabet

01001

abracadabra

ϵ

Empty string (length 0)

- A **language** is a set of strings

$$A = \{\text{good, bad}\}$$

\emptyset { }

Empty set is a language

Languages can be infinite

$A = \{w \mid w \text{ contains at least one 1 and}$
an even number of 0s follow the last 1}

“the set of all ...”

“such that ...”

Last Time: Computers and Languages

- The **language of a machine** is the set of all strings that it accepts

E.g.,

- An DFA $M = (Q, \Sigma, \delta, q_0, F)$ **accepts** string w if $\hat{\delta}(q_0, w) \in F$
- M **recognizes** the language $L(M) = \{w \mid M \text{ accepts } w\}$

Last Time: Regular Languages

A language is called a *regular language* if some finite automaton recognizes it.

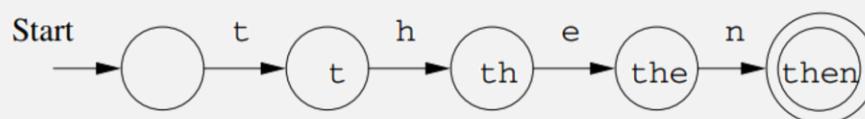
Last Time: Finite State Automaton, a.k.a. DFAs

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. $\delta: Q \times \Sigma \rightarrow Q$ 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**.

- Key characteristic:
 - Has a **finite** number of states
 - I.e., a computer or program with access to a single cell of memory,
 - Where: # states = the possible symbols that can be written to memory
- Often used for **text matching**



Combining DFAs?

Password Requirements

- » Passwords must have a minimum length of ten (10) characters - but more is better!
- » Passwords **must include at least 3** different types of characters:

DFA

» upper-case letters (A-Z) ← DFA

» lower-case letters (a-z)

» symbols or special characters (%,&,*,\$,etc.) ← DFA

» numbers (0-9) ← DFA

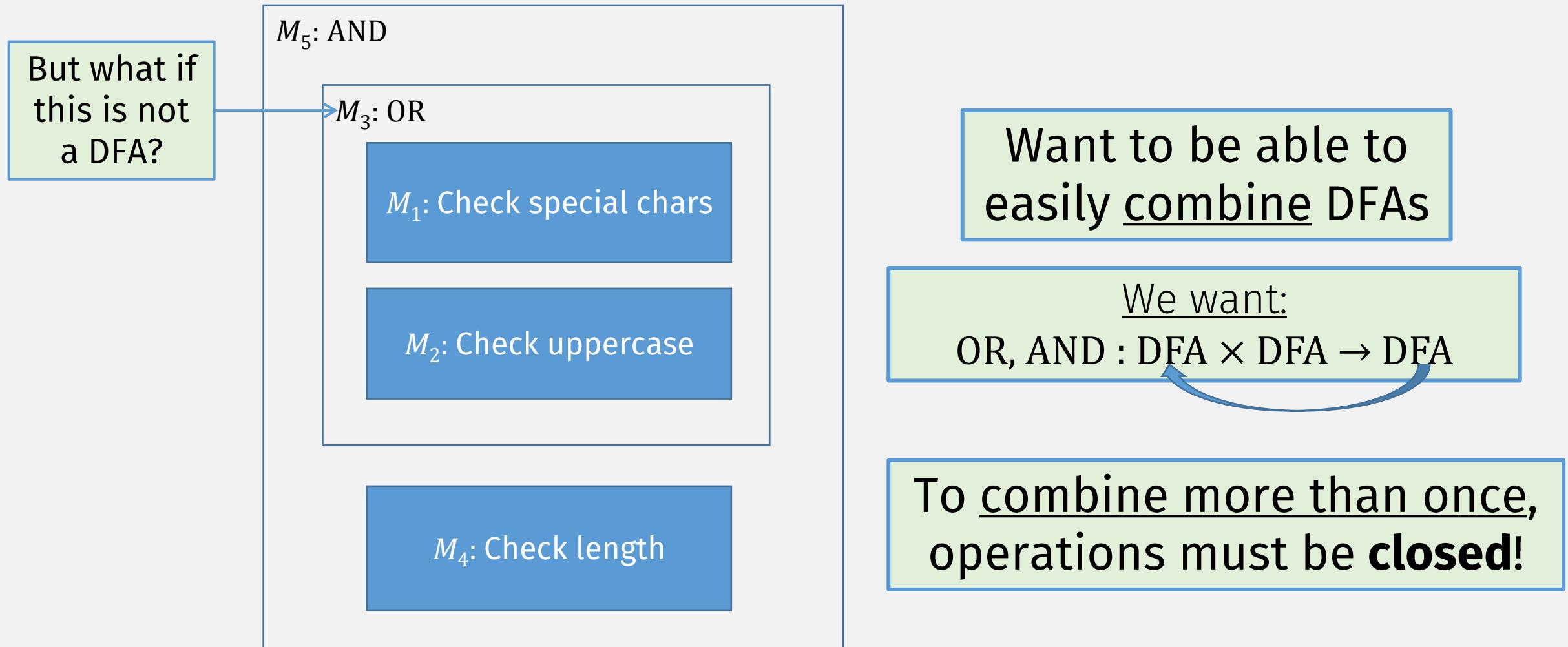
» Passwords cannot contain all or part of your email address ← DFA

» Passwords cannot be re-used ← DFA

To match all requirements, combine smaller DFAs into one big DFA?

<https://www.umb.edu/it/password>

Password Checker DFAs



“Closed” Operations

A set is closed under an operation if:
the result of applying the operation to
members of the set is in the same set

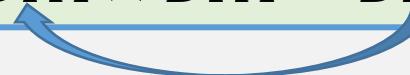
- Set of Natural numbers = {0, 1, 2, ...}
 - Closed under addition:
 - if x and y are Natural numbers,
 - then $z = x + y$ is a Natural number
 - Closed under multiplication?
 - yes
 - Closed under subtraction?
 - no
- Integers = {..., -2, -1, 0, 1, 2, ...}
 - Closed under addition and multiplication
 - Closed under subtraction?
 - yes
 - Closed under division?
 - no
- Rational numbers = { $x \mid x = y/z$, y and z are Integers}
 - Closed under division?
 - No?
 - Yes if $z \neq 0$

Why Care About Closed Ops on Reg Langs?

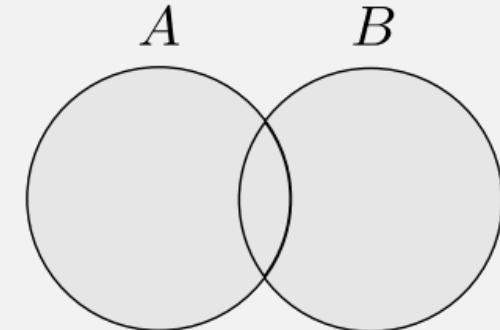
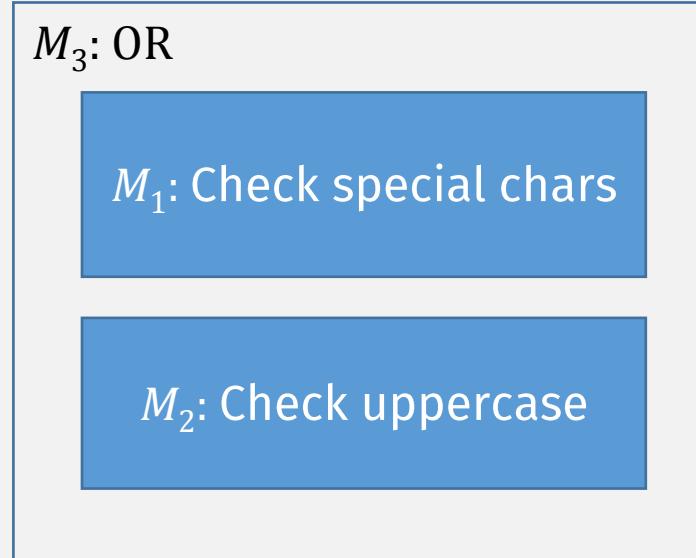
- Closed operations preserve “regularness”
- I.e., it preserves the same computation model!
- This way, a “combined” machine can be “combined” again!

We want:

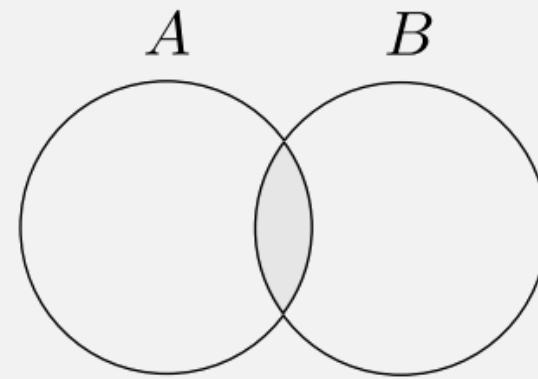
OR, AND : DFA \times DFA \rightarrow DFA



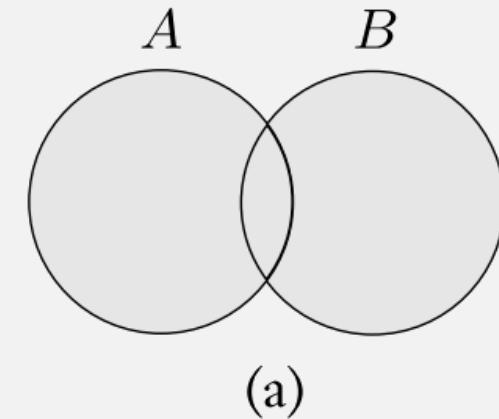
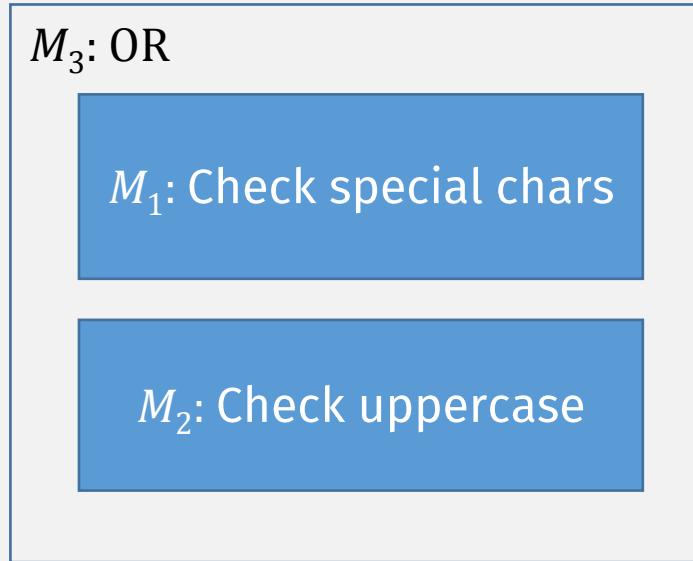
Password Checker: “OR” = “Union”



???



Password Checker: “OR” = “Union”



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

Union of Languages

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

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

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$

A Closed Operation: Union

(A set is **closed** under an operation if the result of applying the operation to members of the set is in the same set)

THEOREM

The **Set of languages** is closed under the union operation.

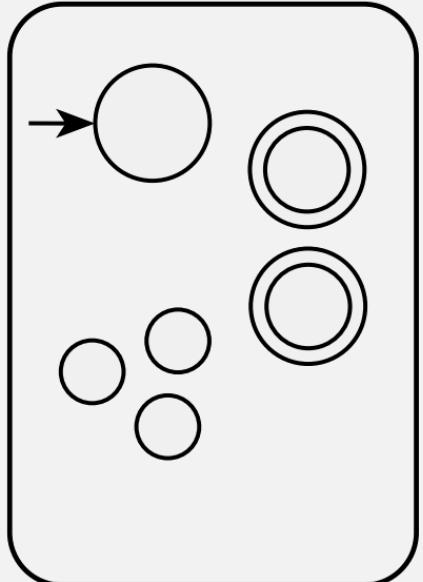
In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

- How do we prove that a language is regular?
 - Create a DFA recognizing it!
- So to prove this theorem ...
create a DFA that recognizes $A_1 \cup A_2$

A language is called a *regular language* if some finite automaton recognizes it.

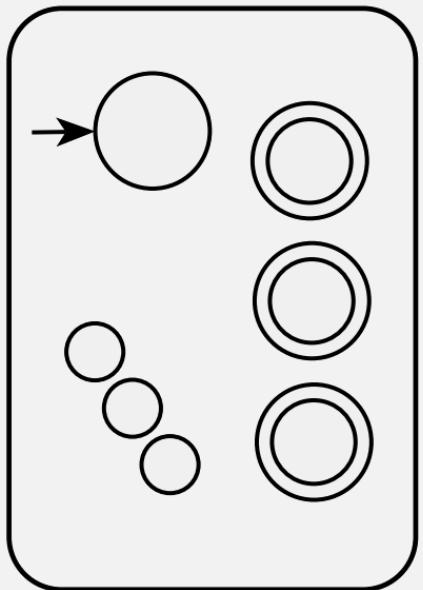
M_1

recognizes A_1



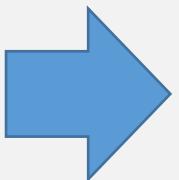
M_2

recognizes A_2



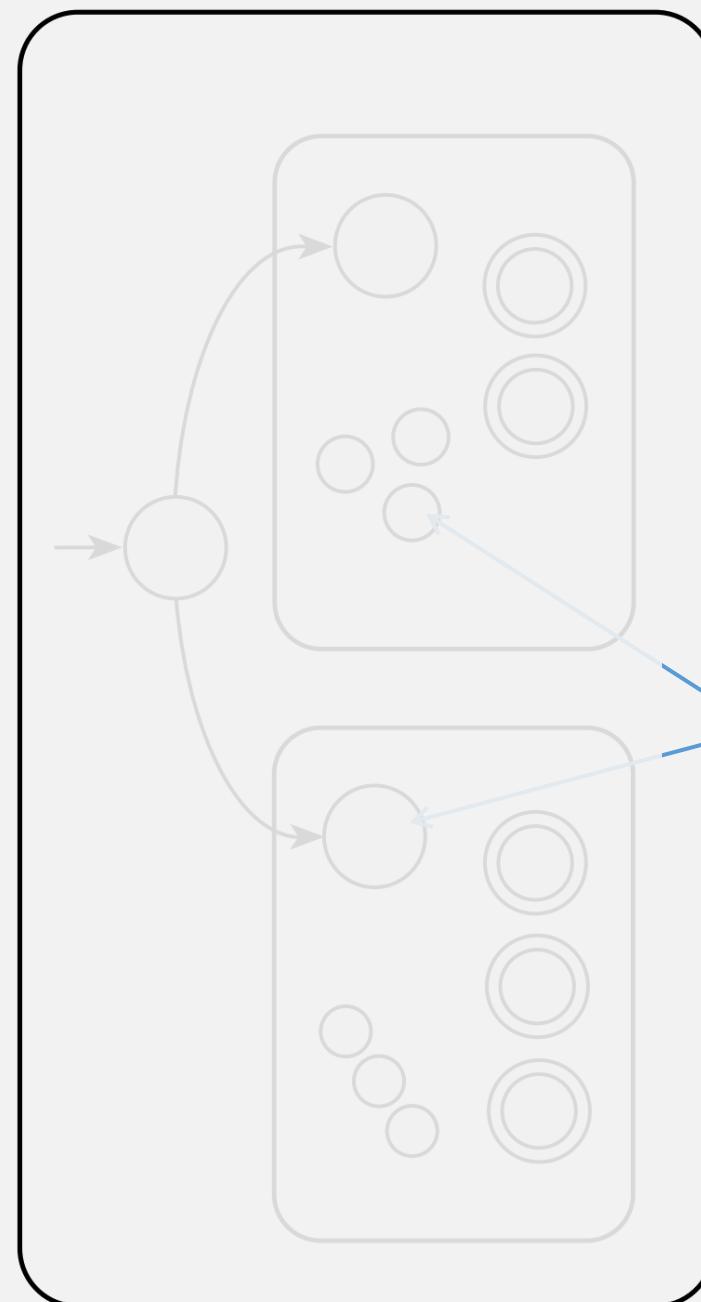
Want: M

Recognizes
 $A_1 \cup A_2$



Union

Rough sketch Idea:
 M is a combination of M_1 and M_2 that “runs” its input on both M_1 and M_2 in parallel



M needs to be “in”
both an M_1 and M_2
state simultaneously

And then accept if
either accepts

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

- 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 ,
Idea: M “runs” its input on both M_1 and M_2 in parallel
- Construct: a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using M_1 and M_2
- states of M : $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$
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. $\delta: Q \times \Sigma \rightarrow Q$ 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**.

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

THEOREM

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

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 : $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$
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 $a) = (\delta_1(r_1, a), \delta_2(r_2, a))$ a step in M_1 , a step in M_2

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

Union is Closed For Regular Languages

THEOREM

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

Proof

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

Union is Closed For Regular Languages

THEOREM

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

Proof

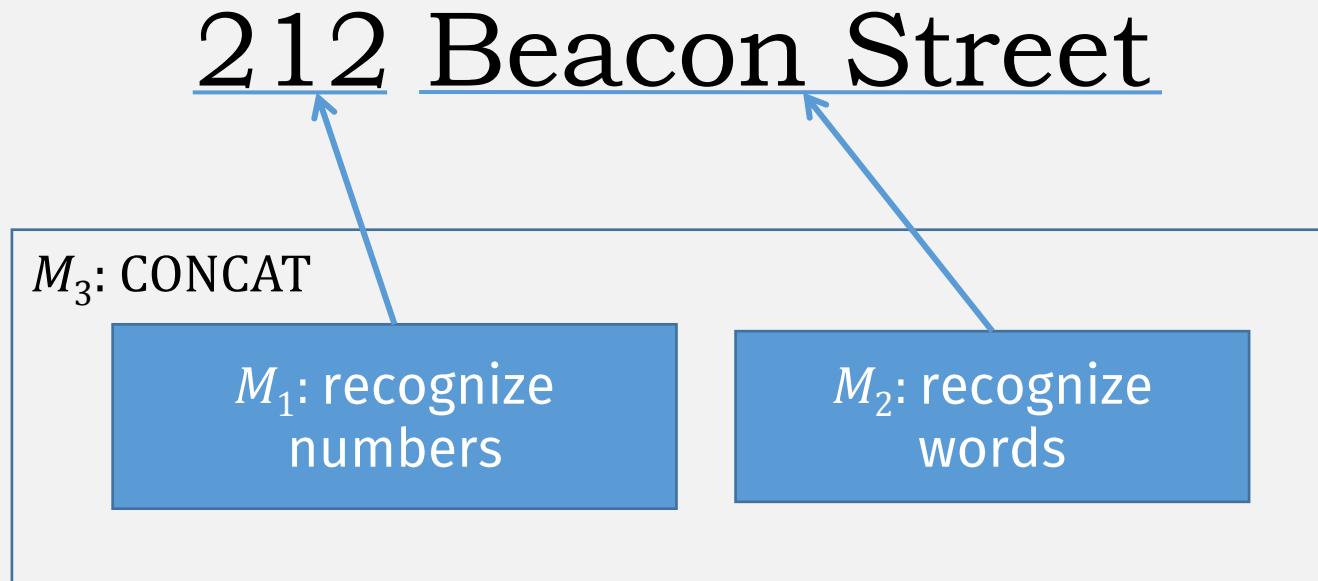
- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1 ,
- Given: $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2 ,
- Construct: a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using M_1 and M_2
- states of M : $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times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

Remember:
Accept states must
be subset of Q

(Q.E.D.) ■

Another 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 $\{a, b, \dots, z\}$.

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

$$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

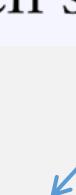
Is Concatenation Closed?

THEOREM

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

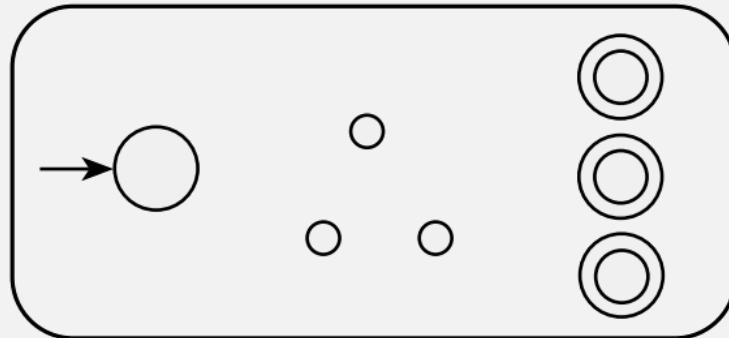
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

- Construct a new machine M recognizing $A_1 \circ A_2$? (like union)
 - From DFA M_1 (which recognizes A_1),
 - and DFA M_2 (which recognizes A_2)

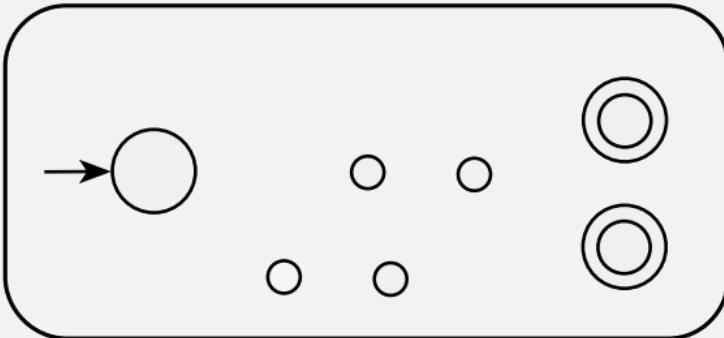


Concatenation

M_1



M_2

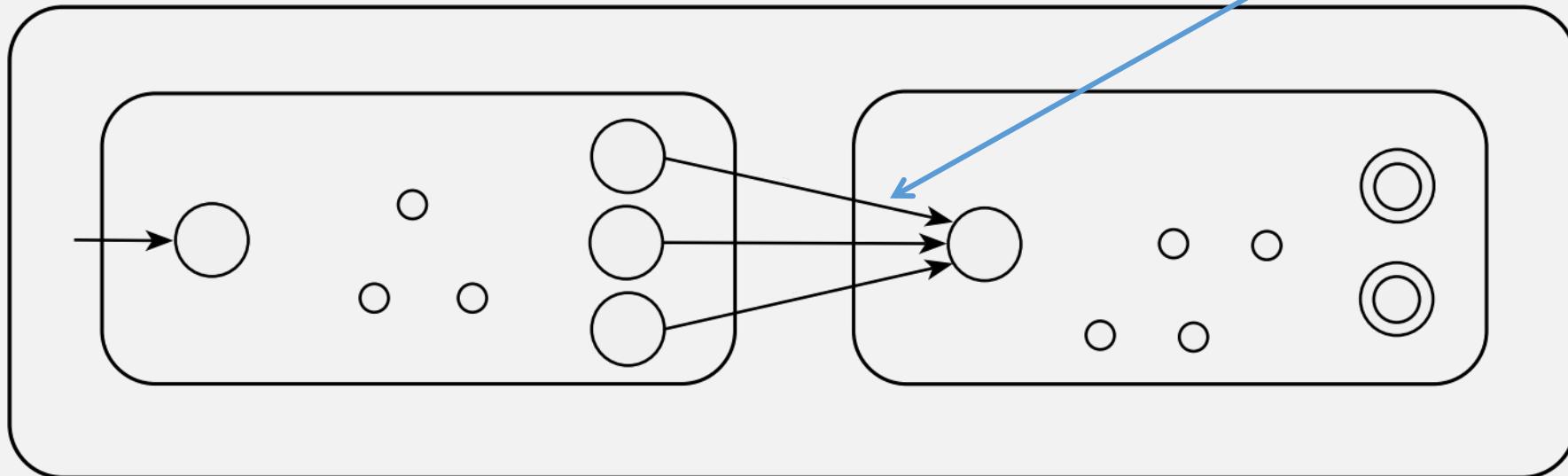


PROBLEM:
Can only
read input
once, can't
backtrack

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 $A = \{ab, abc\}$
- And M_2 recognize language $B = \{cde\}$
- Want: Construct M to recognize $A \circ B = \{\boxed{abcde}, \boxed{abccde}\}$
- But if M sees ab as first part of input ...
- M must decide to either:

Overlapping Concatenation Example

- Let M_1 recognize language $A = \{ab, \boxed{abc}\}$
- And M_2 recognize language $B = \{\boxed{cde}\}$
- Want: Construct M to recognize $A \circ B = \{abcde, abccde\}$

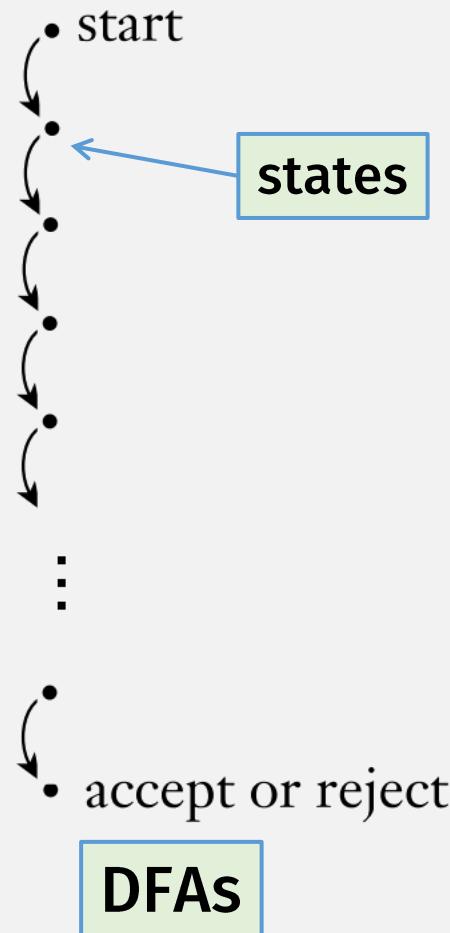
- But if M sees ab as first part of input ...
- M must decide to either:
 - stay in M_1 (correct, if full input is \boxed{abccde})
 - or switch to M_2 (correct, if full input is \boxed{abcde})
- But it needs to handle both cases!

A DFA can't do this!
(We need a new kind of machine)

Nondeterminism

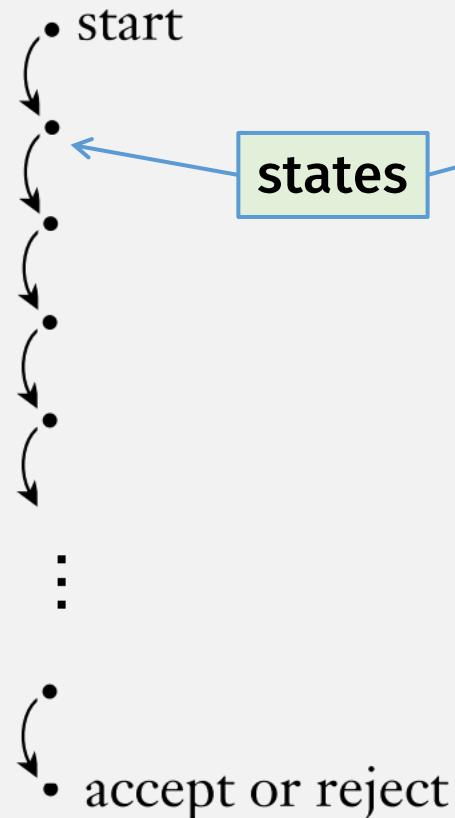
Deterministic vs Nondeterministic

Deterministic
computation



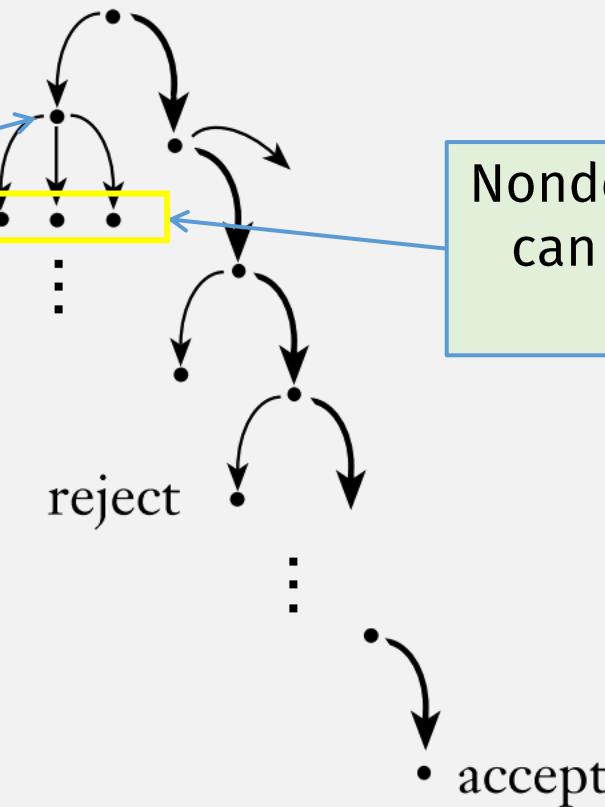
Deterministic vs Nondeterministic

Deterministic
computation



DFAs

Nondeterministic
computation



New FA

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

Difference

Power set, i.e. a transition results in set of 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. $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states**.

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

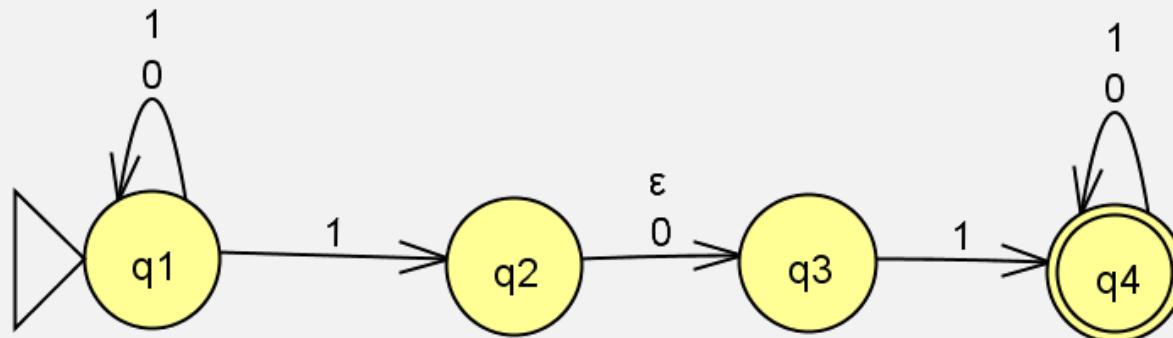
1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

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

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

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. $\delta: Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ 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

Empty transition
(no input read)

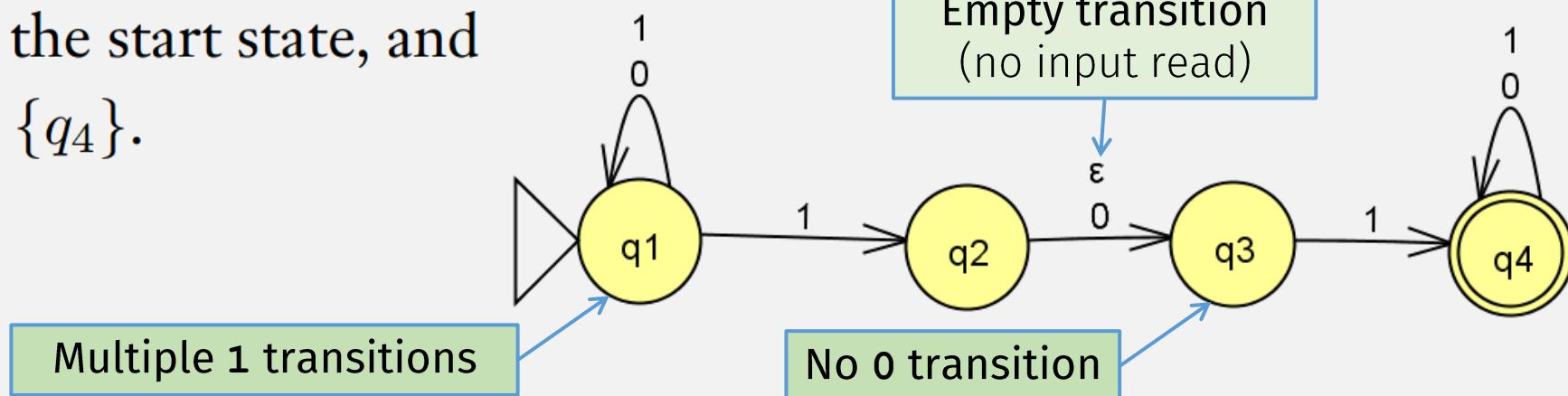
$$\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$$

	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

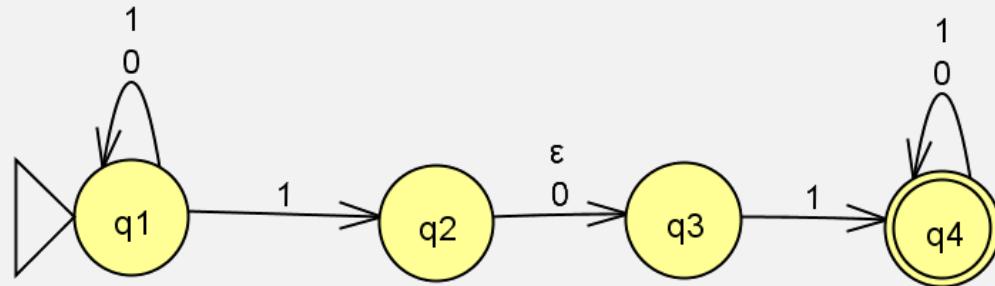
Result of transition
is a set

Empty transition
(no input read)

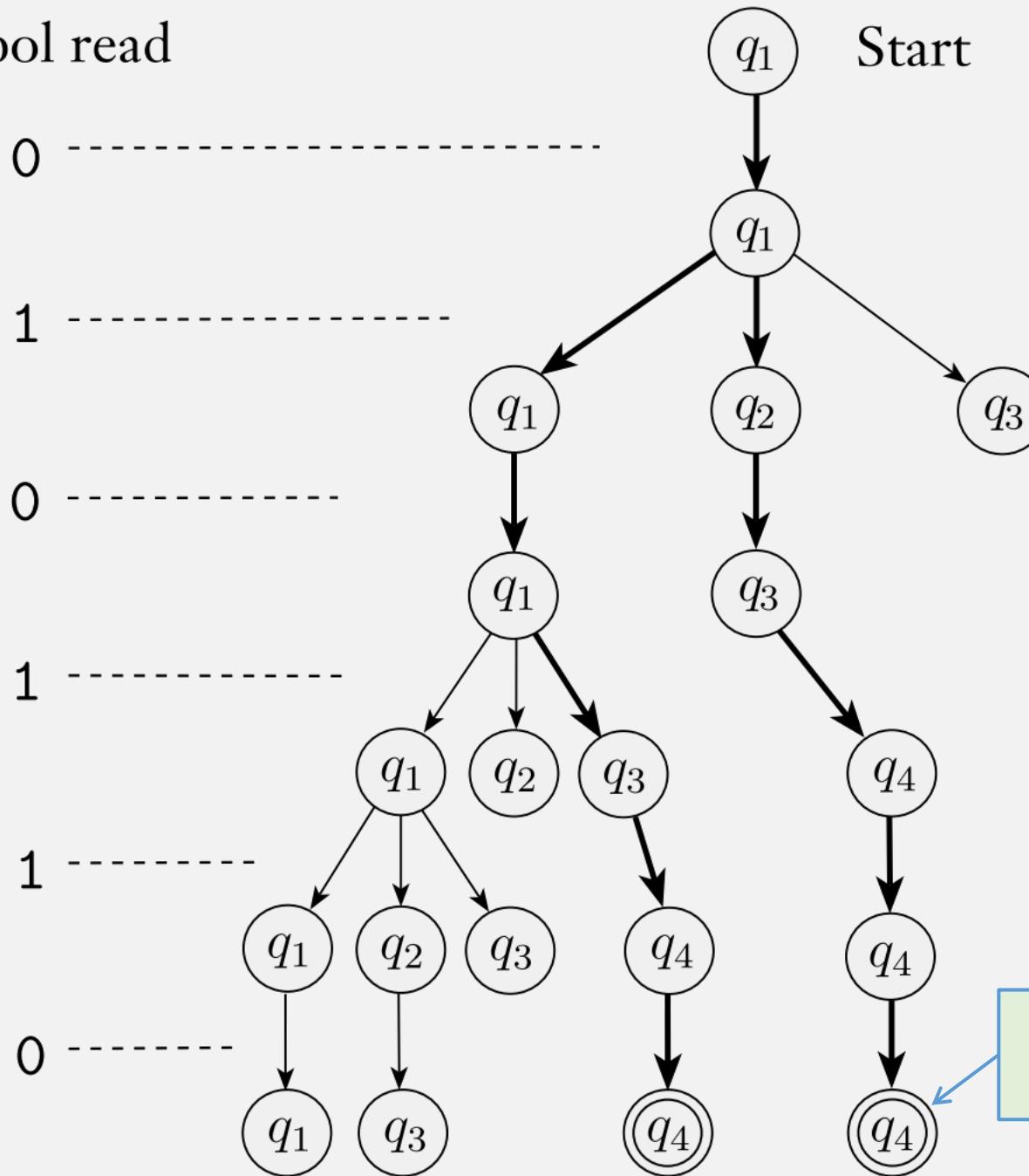
4. q_1 is the start state, and
5. $F = \{q_4\}$.



Running Programs, NFAs (JFLAP demo): 010110



Symbol read



An NFA accepts its input if at least one path ends in an accept state

A nondeterministic machine can be in multiple states at the same time!

So this is an accepting computation

Flashback: DFA Computation Model

Informally

- Machine = a DFA
- Input = string of chars

Machine accepts input if:

- Start in “start state”
- Repeat: Read 1 char, change state according to transition table
- Result =
 - “Accept” if last state is “Accept” state
 - “Reject” otherwise

Formally

- $M = (Q, \Sigma, \delta, q_0, F)$
 - $w = w_1 w_2 \cdots w_n$
- M **accepts** w if sequence of states r_0, r_1, \dots, r_n in Q exists with
- $r_0 = q_0$
 - $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n - 1$
 - $r_n \in F$

NFA

~~Flashback:~~ ~~DFA~~ Computation Model

Informally

- Machine = a ~~DFA~~ NFA
- Input = string of chars

Machine accepts input if:

- Start in “start state”
- Repeat: Read 1 char, change state according to transition table
- Result =
 - “Accept” if last state is “Accept” state
 - “Reject” otherwise

Formally

- $M = (Q, \Sigma, \delta, q_0, F)$
 - $w = w_1 w_2 \cdots w_n$
- M **accepts** w if sequence of states r_0, r_1, \dots, r_n in Q exists with
- $r_0 = q_0$
 - $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n - 1$
 - $r_n \in F$
- $r_{i+1} \in \delta(r_i, w_{i+1})$ ← This is now a set
-
- 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. $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is the transition function,
 4. $q_0 \in Q$ is the start state, and
 5. $F \subseteq Q$ is the set of accept states.

Flashback: DFA Extended Transition Fn

Define the extended transition function: $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

- Inputs:
 - Beginning state $q \in Q$
 - Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Output:
 - Ending state

(Defined recursively)

- Base case: $\hat{\delta}(q, \varepsilon) = q$
 - Recursive case: $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n)$
-
- The diagram illustrates the recursive definition of the extended transition function. It shows two main components: the base case and the recursive case. In the base case, an empty string leads to a state q . In the recursive case, a string w is processed by first taking its first character (labeled 'First char') and then making a recursive call on the remaining characters (labeled 'Remaining chars'). This recursive call is shown as a 'Recursive call' to the base case. A 'Single transition step' is also indicated between the recursive call and the final result.

NFA

~~Flashback:~~ DFA - Extended Transition Fn

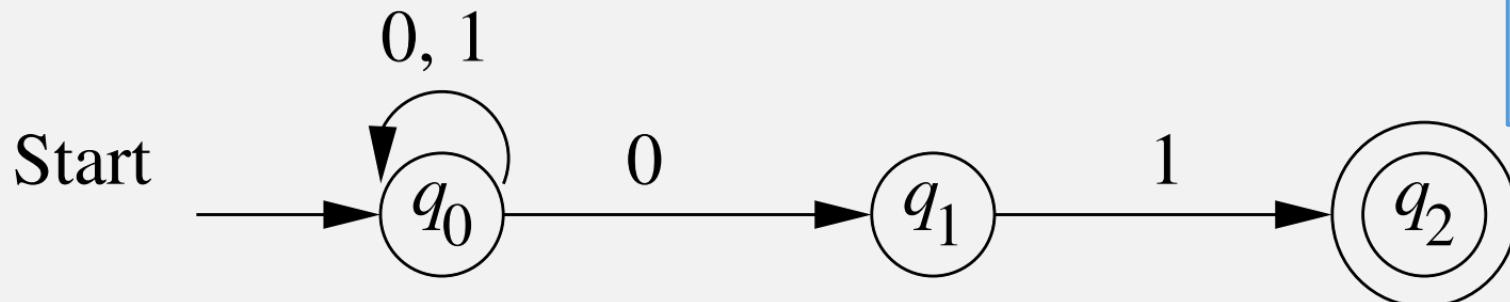
Define the extended transition function: $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

- Inputs:
 - Beginning state $q \in Q$
 - Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Output:
 - Ending state Set of ending states

(Defined recursively)

- Base case: $\hat{\delta}(q, \epsilon) = q$ $\hat{\delta}(q, \epsilon) = \{q\}$
 - Recursive case: $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), u)$ $\hat{\delta}(q, w) = \bigcup_{i=1}^k \hat{\delta}(q_i, w_2 \cdots w_n)$
- Combine results of recursive calls for each q_i
- First char
- Result is set of states

NFA Extended delta Example



We haven't considered empty transitions!

- $\hat{\delta}(q_0, \epsilon) = \{q_0\}$
- $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$

Read one char, do recursive call for each result, then combine
- $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

Adding Empty Transitions

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

(Defined recursively)

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

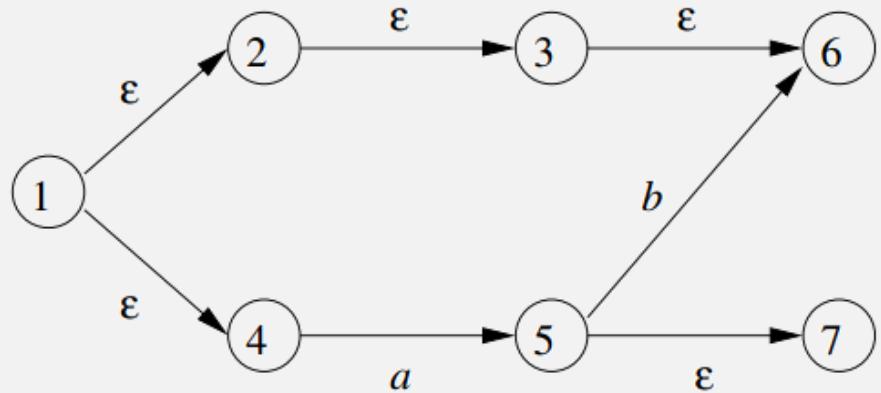
- **Inductive case:**

A state is in the reachable set if ...

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

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

ε -REACHABLE Example



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

NFA Extended Transition Function

Define the extended transition function: $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$

- *Inputs:*
 - Some beginning state q
 - Input string $w = w_1 w_2 \cdots w_n$
- *Output:*
 - Set of ending states

(Defined recursively)

- Base case: $\hat{\delta}(q, \epsilon) = \{q\}$
- Recursive case:
 - If: $\delta(q, w_1) = \{q_1, \dots, q_k\}$

- Then:
$$\hat{\delta}(q, w) = \bigcup_{i=1}^k \hat{\delta}(q_i, w_2 \cdots w_n)$$

NFA Extended Transition Function

Define the extended transition function: $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$

- *Inputs:*
 - Some beginning state q
 - Input string $w = w_1 w_2 \cdots w_n$
- *Output:*
 - Set of ending states

(Defined recursively)

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

- Recursive case:

- If: $\delta(q, w_1) = \{q_1, \dots, q_k\} \xrightarrow{\quad} \bigcup_{i=1}^k \varepsilon\text{-REACHABLE}(q_i) = \{r_1, \dots, r_m\}$

- Then: $\hat{\delta}(q, w) = \bigcup_{i=1}^k \hat{\delta}(q_i, w_2 \cdots w_n)$

NFAs vs DFAs

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
 - Can read no chars
 - i.e., empty transition
- Acceptance:
 - If one of final states is accept state

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

Try # 2: Concatenation is Closed?

THEOREM

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

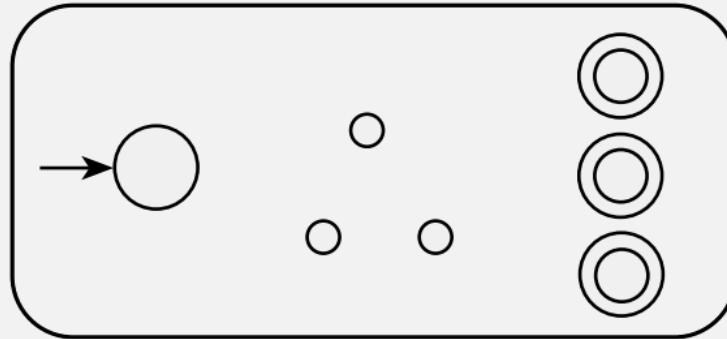
In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

Proof: Construct a new machine?

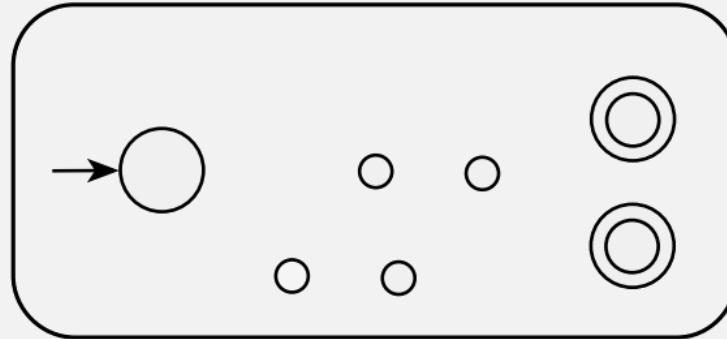
- How does it know when to switch machines
 - Can only read input once

Concatenation

N_1



N_2



Let N_1 recognize A_1 , and N_2 recognize A_2 .

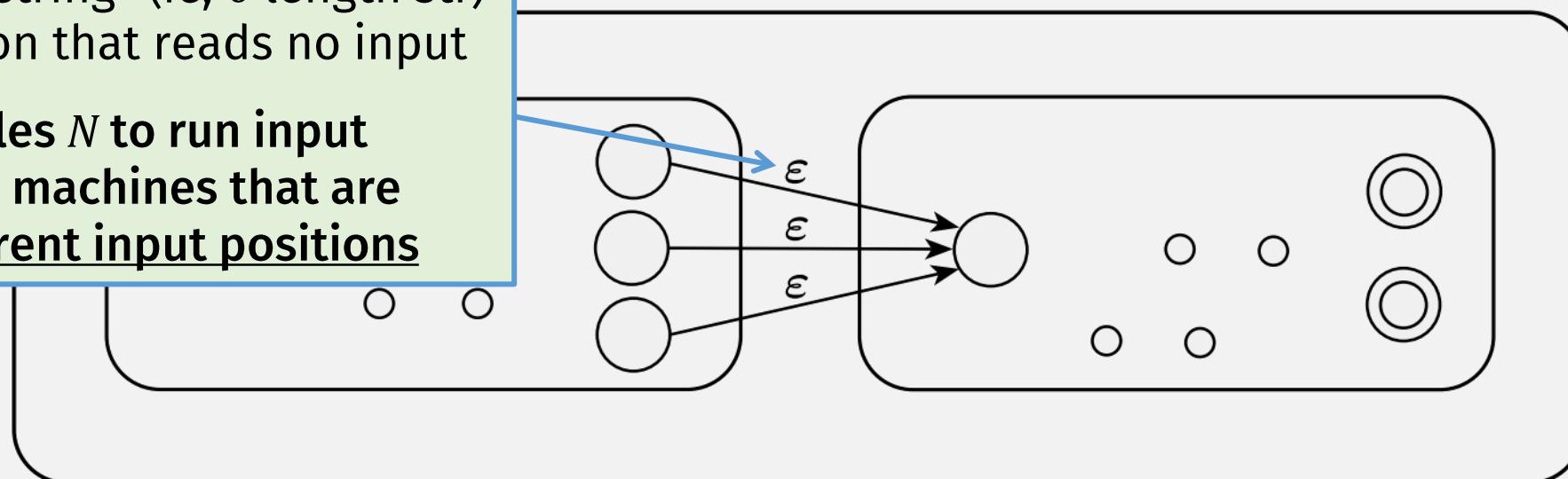
Want: Construction of N to recognize $A_1 \circ A_2$

N must simultaneously:

- Keep checking with N_1 **and**
- Move to N_2 to check 2nd part

ϵ = “empty string” (ie, 0 length str)
= transition that reads no input

**Enables N to run input
on two machines that are
at different input positions**



Concatenation is Closed for Regular Langs

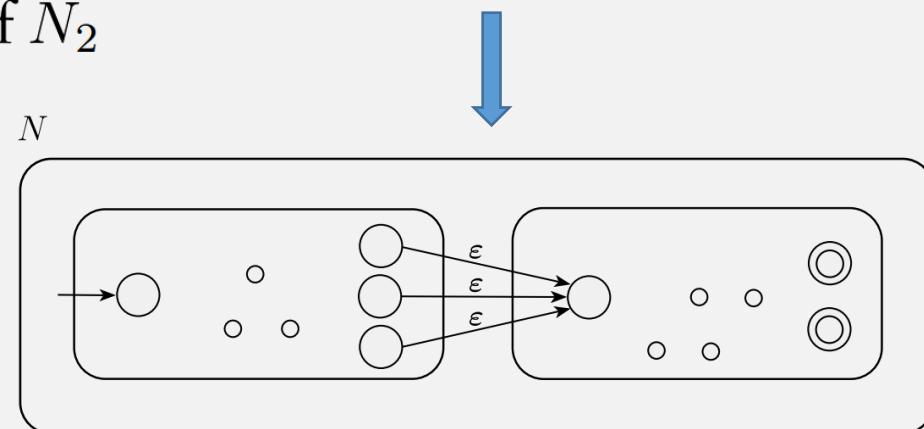
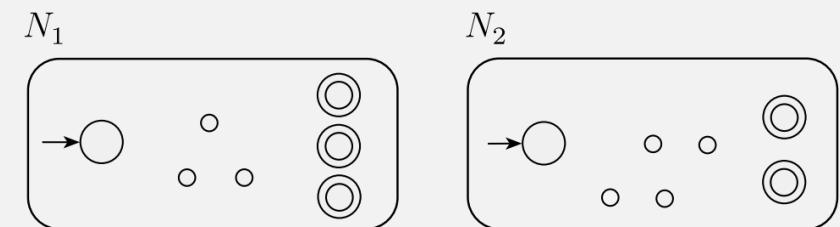
PROOF

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

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

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 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 $a \in \Sigma_\epsilon$,



Concatenation is Closed for Regular Langs

PROOF

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

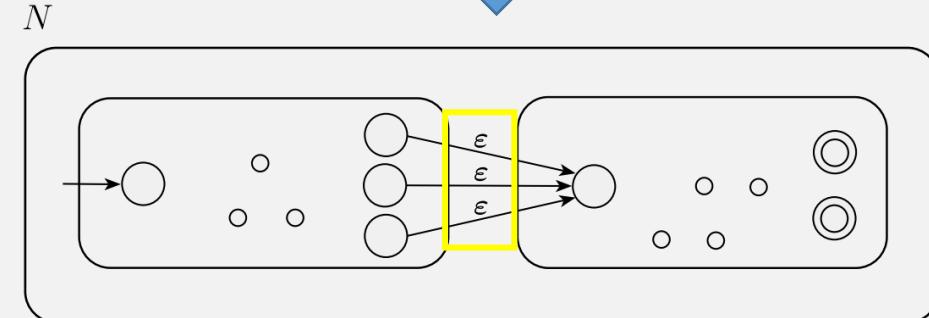
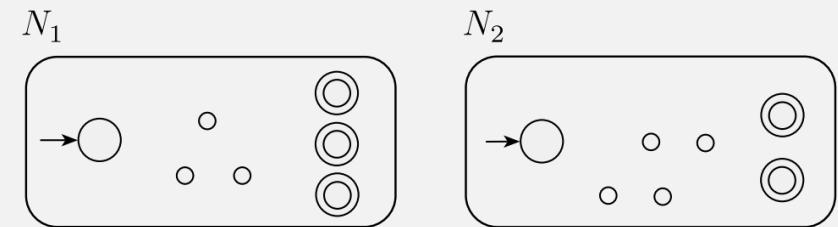
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2 .

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 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 $a \in \Sigma_\epsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) \\ \delta_1(q, a) \\ \delta_1(q, a) \cup \{q_2\} \\ \delta_2(q, a) \end{cases}$$

Wait, is this true?



???

Flashback: 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** A if $A = \{w \mid M \text{ accepts } w\}$
- A language is a **regular language** if a DFA recognizes it

An NFA's Language

- For NFA $N = (Q, \Sigma, \delta, q_0, F)$
- N **accepts** w if $\hat{\delta}(q_0, w) \cap F \neq \emptyset$
 - i.e., if the final states have at least one accept state
- Language of $N = L(N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$
- Q: How does an NFA's language relate to regular languages
 - Reminder: A language is regular if a DFA recognizes it

So is Concatenation Closed for Reg Langs?

- Concatenation of DFAs produces an NFA
- But: A language is called a *regular language* if some DFA recognizes it.
- To finish the proof,
we must prove that NFAs also recognize regular languages.
- Specifically, we must prove:
 - NFAs \Leftrightarrow regular languages

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

- $X \Leftrightarrow Y$ = “ X if and only if Y ” = X iff Y = $X \Leftrightarrow Y$
- Proof at minimum has 2 parts:
 1. \Rightarrow if X , then Y
 - “forward” direction
 - assume X , then use it to prove Y
 2. \Leftarrow if Y , then X
 - “reverse” direction
 - assume Y , then use it to prove X

Proving NFAs Recognize Regular Langs

Theorem:

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

Proof:

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

- Easier
- We know: if A is regular, then a **DFA** recognizes it.
- Easy to convert DFA to an NFA! (how?)

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

- Harder
- Idea: Convert NFA to DFA

Check-in Quiz 1/31

On gradescope