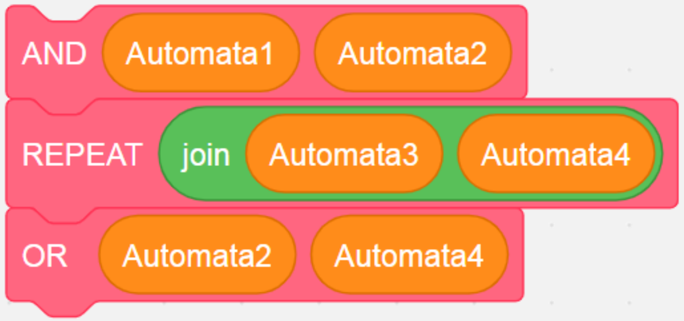
CS420

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

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

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

- 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

Languages can be infinite

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

$$\emptyset \qquad \{\ \}$$

Empty set is a language

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

"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

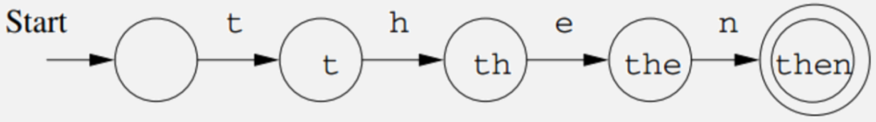
deterministic

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

- 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

DFA

- Passwords must have a minimum length of ten (10) characters - but more is better!
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z) DFA
  - 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 <u>all</u> requirements, combine smaller DFAs into one big DFA?

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

Password Checker DFAs

But what if this is not a DFA?

M_5: AND

M_3: OR

M_1: Check special chars

M_2: Check uppercase

M_4: Check length

Want to be able to easily <u>combine</u> DFAs

<u>We want:</u>

$$\text{OR, AND} : \text{DFA} \times \text{DFA} \rightarrow \text{DFA}$$

To <u>combine more than once,</u> operations must be **closed**!

"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 = $\{0, 1, 2, \ldots\}$
  - 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 = $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
  - 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 !=0

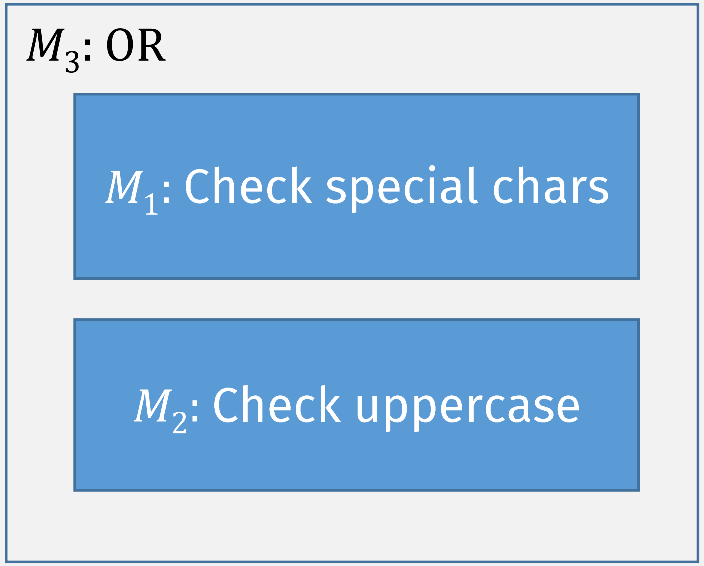
Why Care About Closed Ops on Reg Langs?

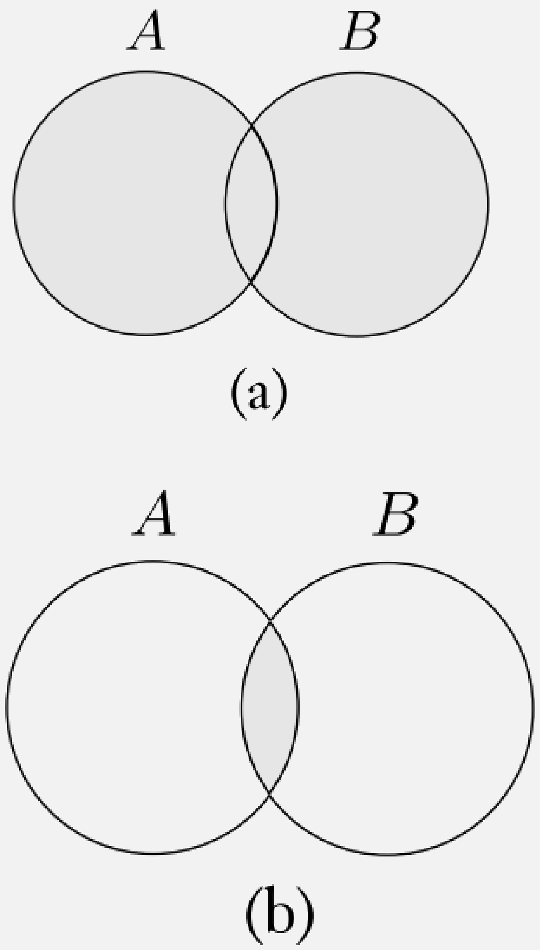
- Closed operations preserve “regularness”
- I.e., it preserves the same computation model!
- This way, a “combined” machine can be “combined” again!

We want:

OR, AND : DFA × DFA → DFA

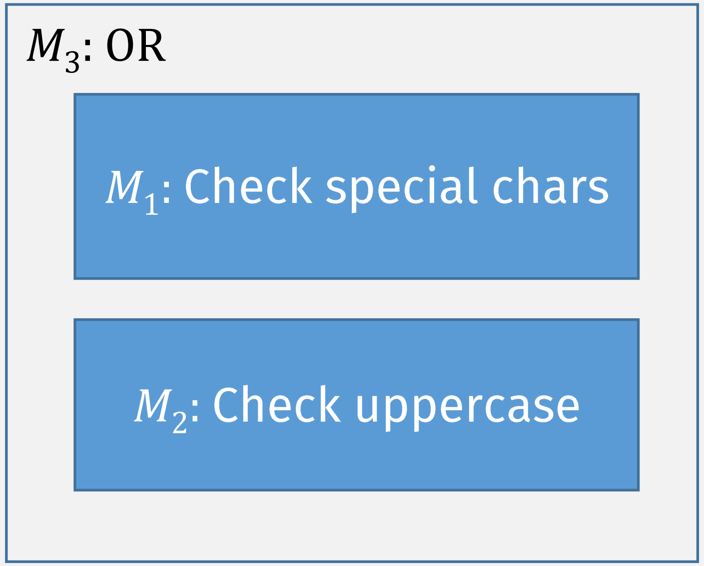
Password Checker: “OR” = “Union”

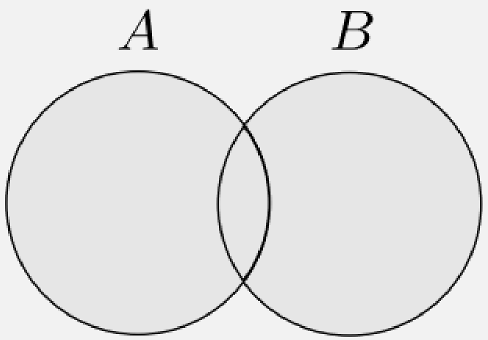




???

Password Checker: “**OR**” = “Union”





(a)

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

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

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

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)

Set of languages

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 $A_1 \cup A_2$.

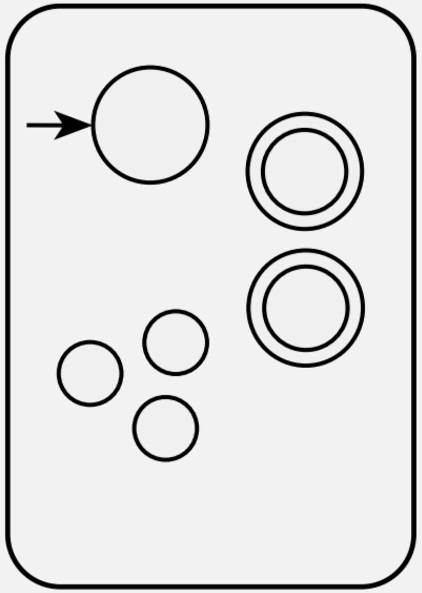
A language is called a ***regular language*** if some finite automaton recognizes it.

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

Union

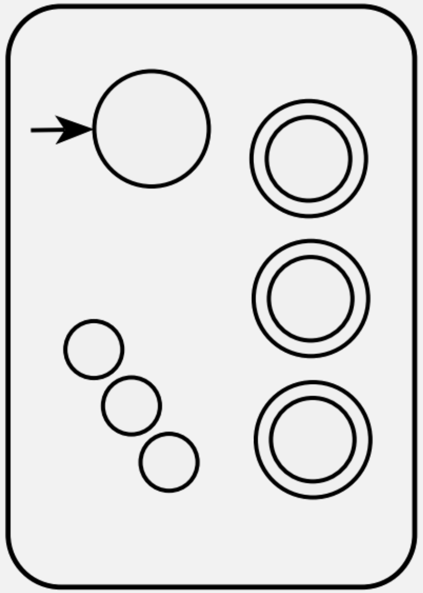
M_1

recognizes A_1



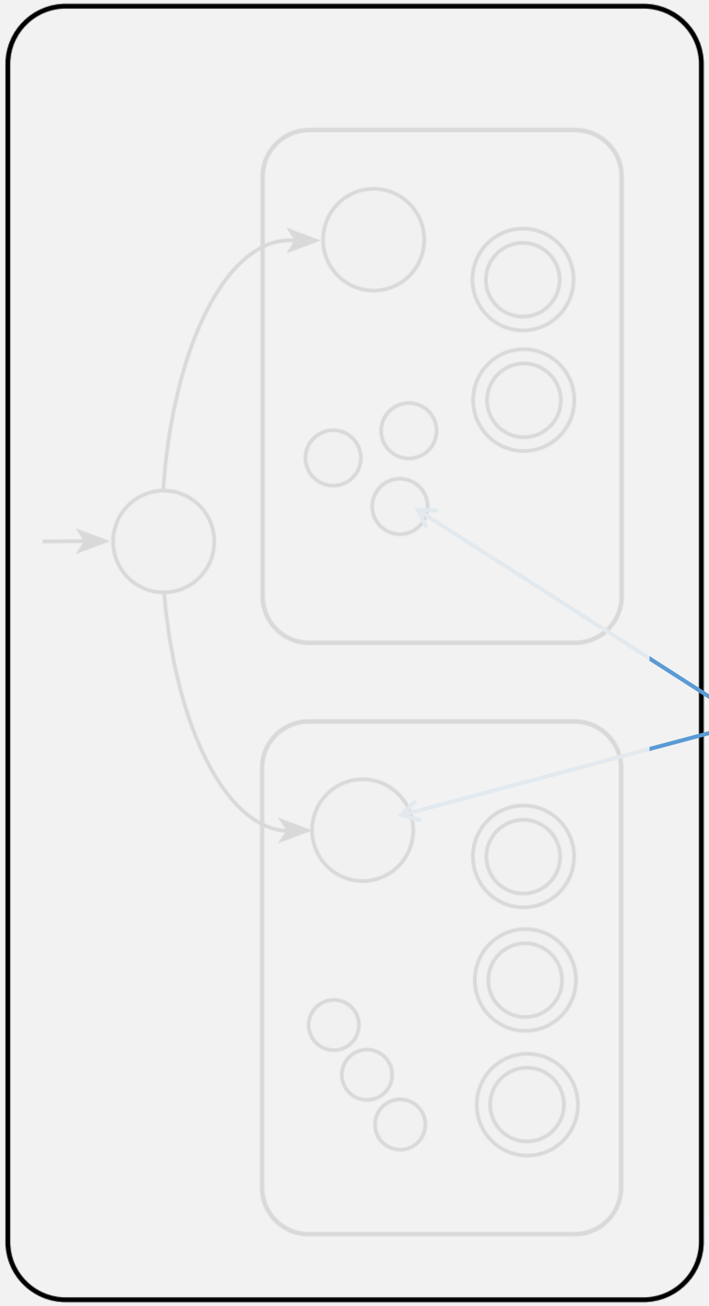
M_2

recognizes A_2



Want: M

Recognizes $A_1 \cup A_2$



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

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

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

  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. $\delta \colon Q \times \Sigma \longrightarrow Q$ 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***.

Union is Closed For Regular Languages

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

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

$a) = (\delta_1(r_1, a), \delta_2(r_2, a))$ a step in M_1, 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***.

Union is Closed For Regular Languages

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__

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

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

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
- 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)

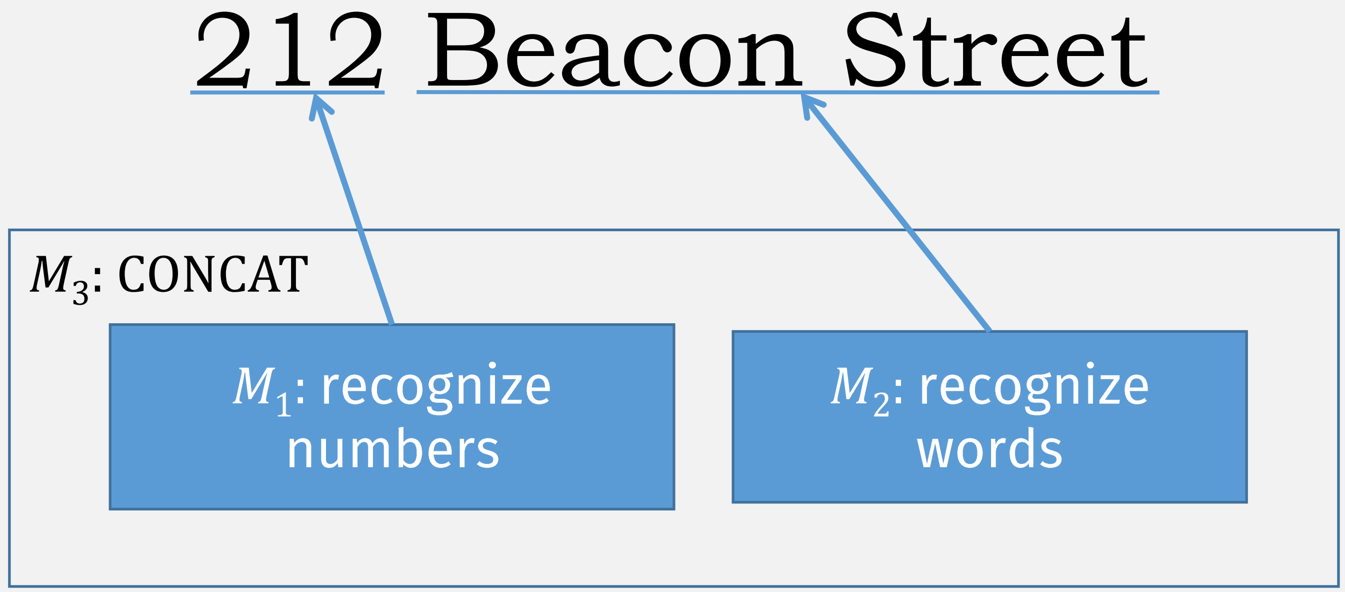
  Accept if either M_1 or M_2 accept

  Remember: Accept states must be subset of Q
- M accept states: $F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}$

(Q.E.D.) ■

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

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

Is Concatenation Closed?

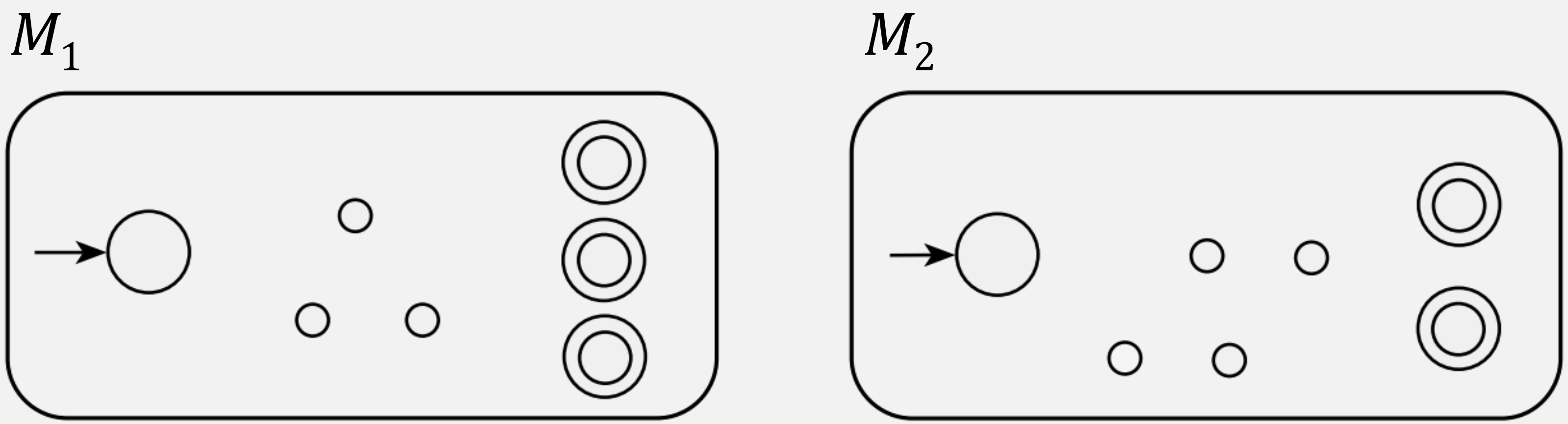
THEOREM

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 <u>new</u> 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)

Concatentation

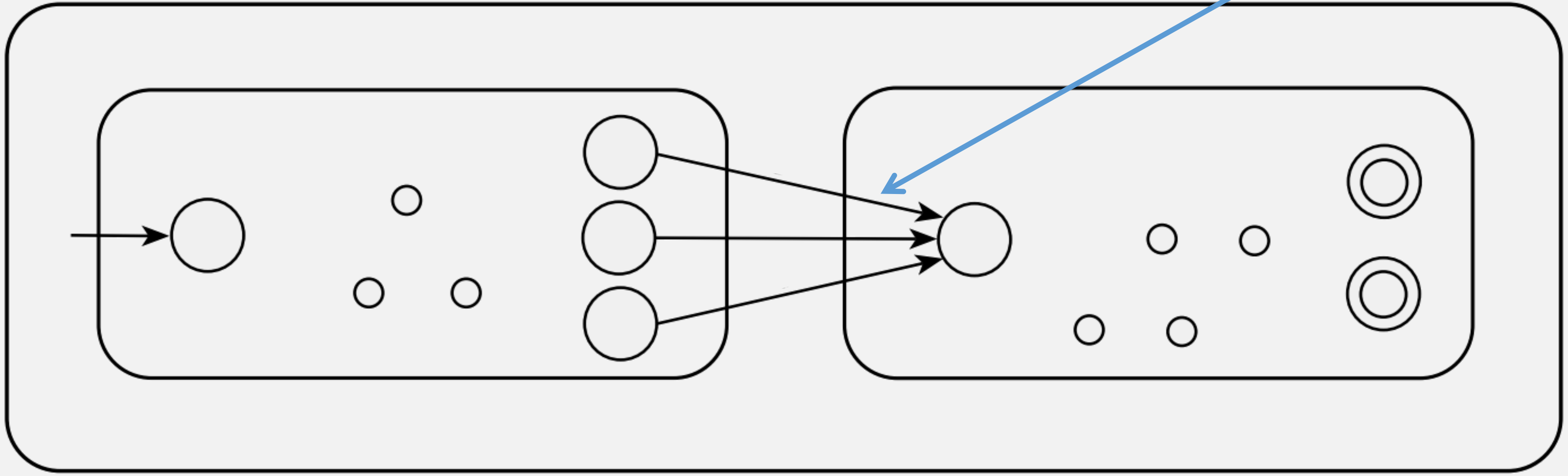


Let M_1 recognize A_1, and M_2 recognize A_2.

<u>Want</u>: Construction of M to recognize $A_1 \circ A_2$

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

Need to switch machines at some point, but when?



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

Overlapping Concatenation Example

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

- But if M sees **ab** as first part of input …
- M must decide to either:

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

Overlapping Concatenation Example

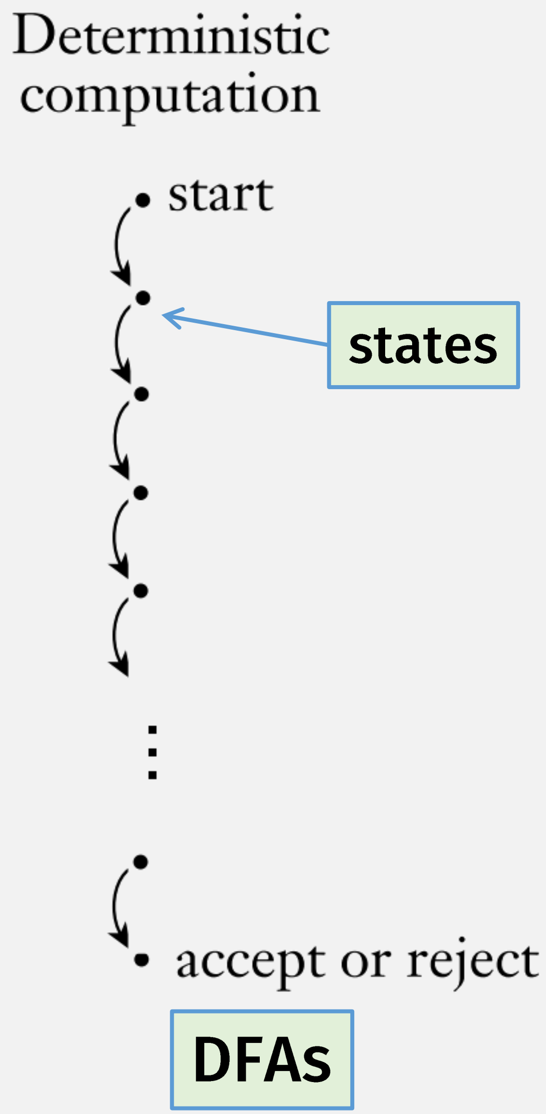
- Let M_1 recognize language A = {**ab**, **abc**}
- And M_2 recognize language B = {**cde**}
- Want: **Construct** M **to recognize** $A{\circ}B$ = {**abcde**, **abccde**}

- **But if** M **sees ab as first part of input …**
- M **must decide to either:**
  - **stay in** M_1 **(correct, if full input is abccde)**
  - **or switch to** M_2 **(correct, if full input is abcde)**
- **But it needs to handle <u>both cases</u>!**

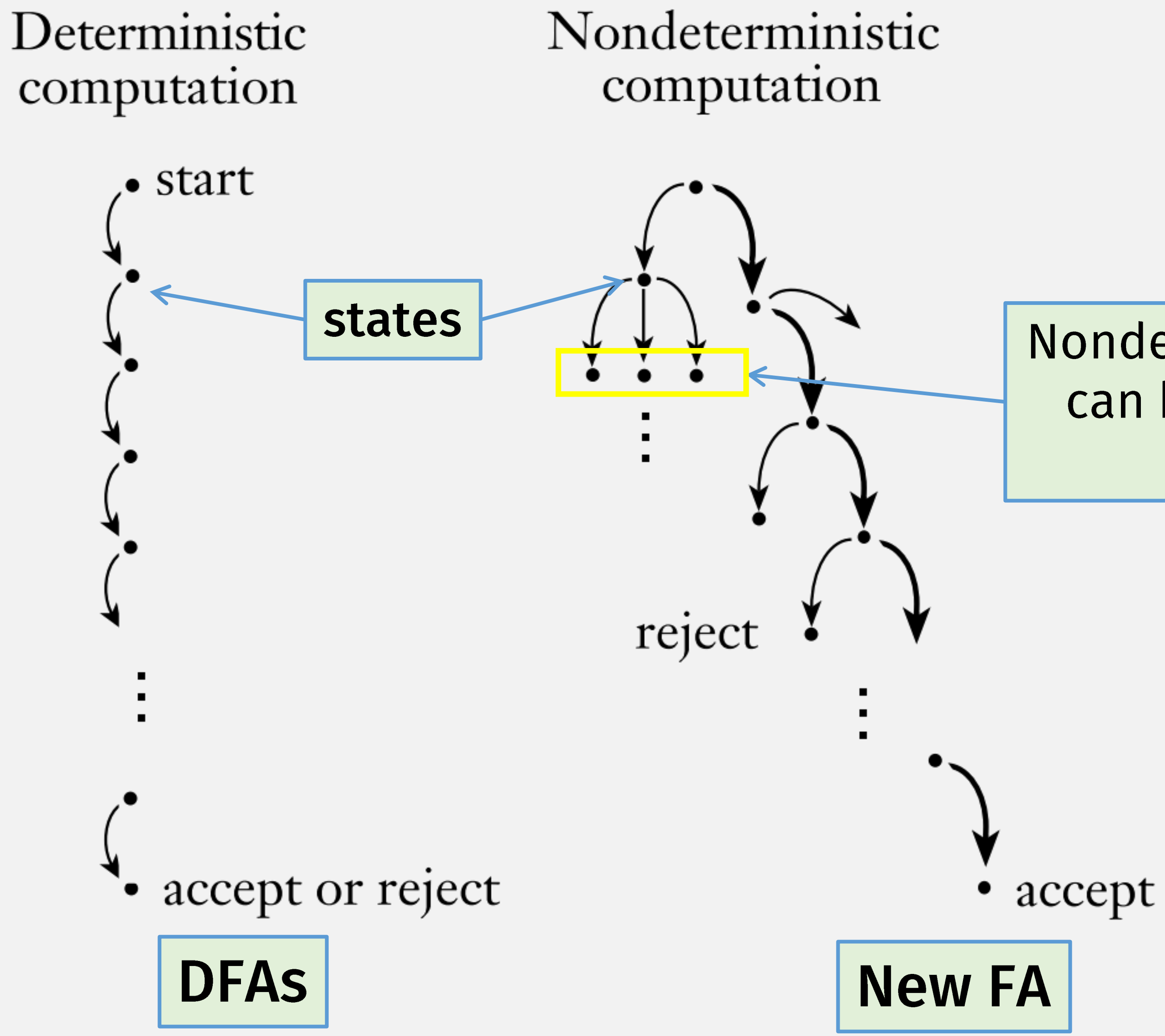
A DFA can't do this!
(We need a new kind of machine)

Nondeterminism

Deterministic vs Nondeterministic



Deterministic vs Nondeterministic



Nondeterministic computation can be in <u>multiple states at the same time</u>

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

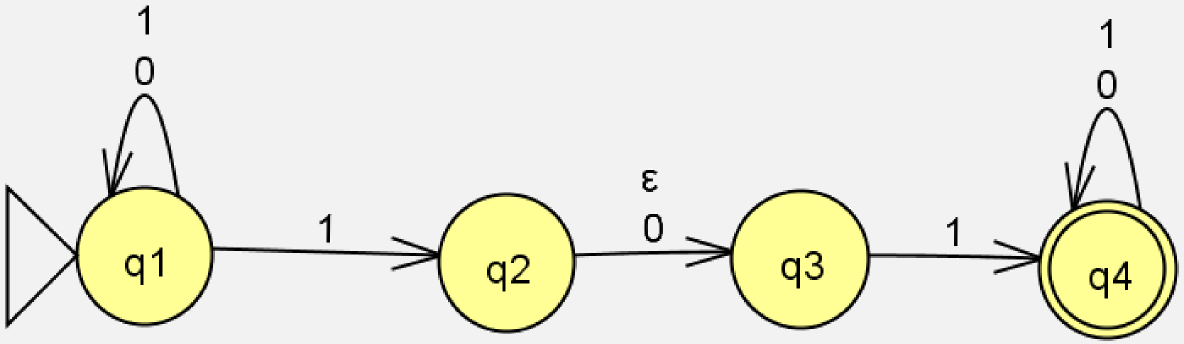
1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

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

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

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

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

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

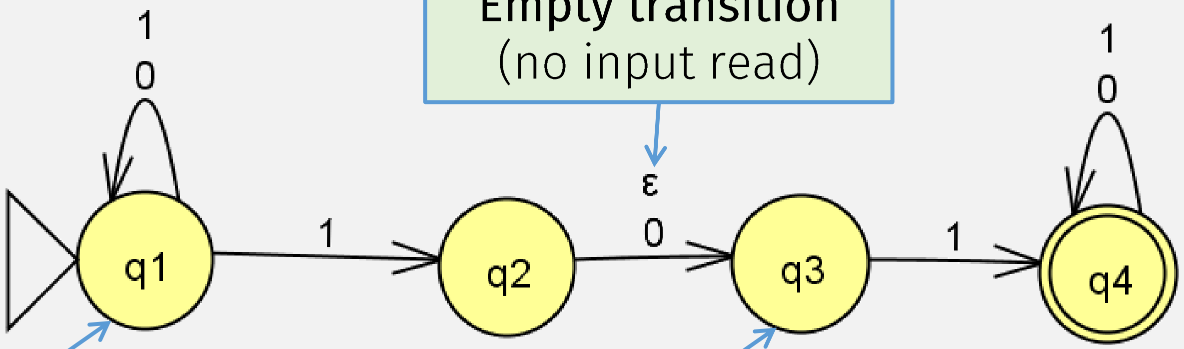
Empty transition (no input read)

	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

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

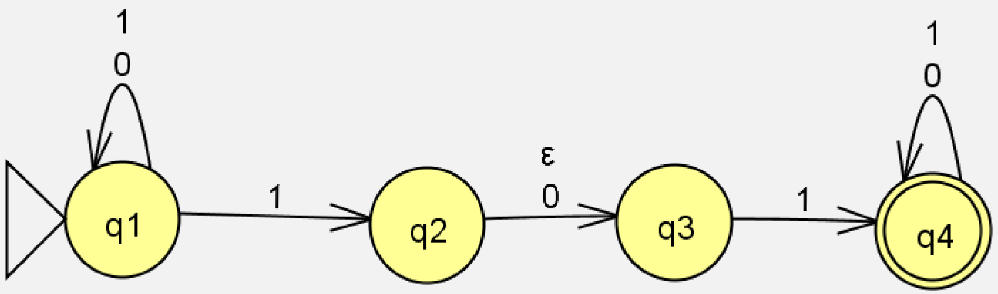
Empty transition (no input read)



Multiple 1 transitions

No 0 transition

Next Time: Running Programs, NFAs (JFLAP demo): `010110`



Check-in Quiz 1/31

On gradescope