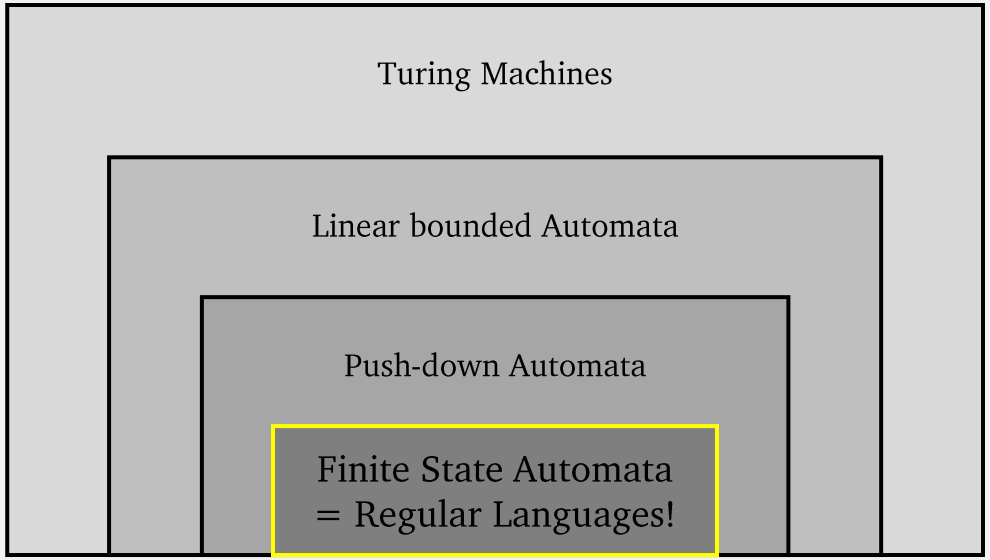
CS420

Regular Languages

Thursday, September 15, 2023

UMass Boston Computer Science



Announcements

- HW 0 in
  - ~~Due Wed 9/13 11:59pm EST~~

- HW 1 out
  - Due Sun 9/25 11:59pm EST

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

Last Time: Computation and Languages

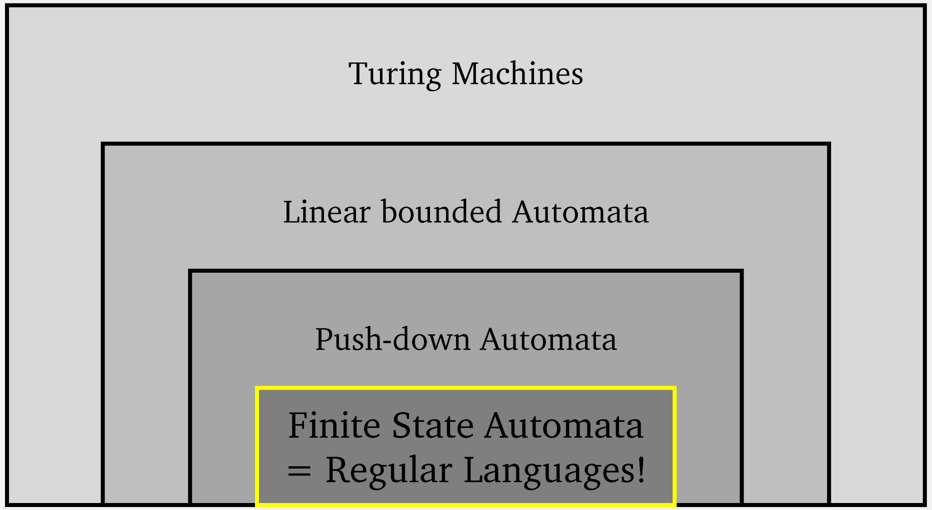
- The **language of a machine** is the set of all strings that it accepts

- A **computation model** is equivalent to the set of machines it defines
  - E.g., all possible Finite State Automata are a computation model

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***.



- Thus: a **computation model** is also equivalent to a set of languages

Last Time: Regular Languages: Definition

If **a finite automaton (FSM)** <u>recognizes</u> **a language,**

then **that language is called a regular language.**

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

M ***recognizes language*** A
if $A = \{w|\ M \text{ accepts } w\}$

Last Time: A Language, Regular or Not?

- If given: **a Finite Automaton M**
  - We know: $L(M)$**, the language recognized by** M**, is a regular language**
  - Because:

> If **a finite automaton** (FSM) recognizes **a language,**
> then **that language is called a regular language.**

- If given: **a Language A**
  - **Is A is a regular language?**
    - **Not necessarily!**
  - **How do we determine,** i.e., ***prove*, that A is a regular language?**

An Inference Rule: Modus Ponens

Premises

- If P then Q
- P is true

Conclusion

- Q is true

<u>Example</u> **Premises**

- If an FSM recognizes language A, then A is a regular language
- There is an FSM M where $L(M) = A$

Conclusion

- A is a regular language!

… then we need to show

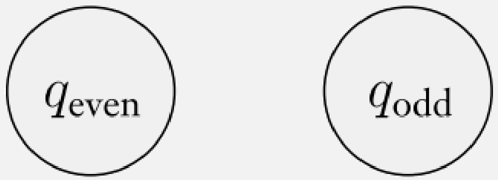
If we want to prove …

Last Time: Designing Finite Automata: Tips

- States = the machine's **memory**!
  - So think about what information must be remembered.
  - (# states must be decided in advance)
- Input may only be read once, one char at a time
- Must decide accept/reject after that
- Every state/symbol pair must have a transition (for DFAs)

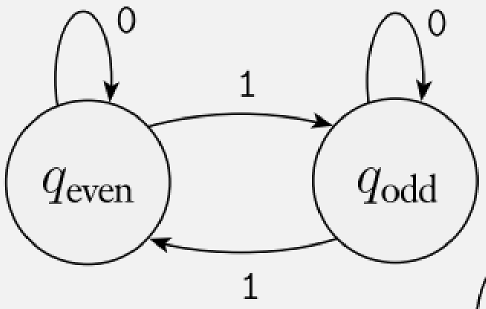
Design a DFA: accept strings with odd # **1**s

- States:
  - 2 states:
    - seen even **1**s so far
    - seen odds **1**s so far

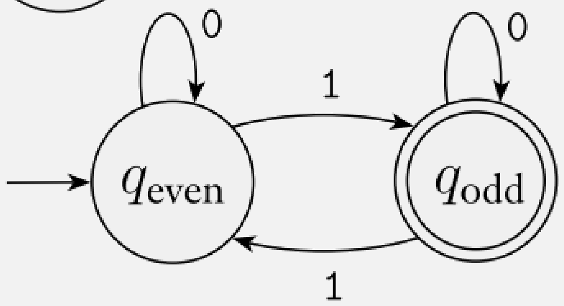


- Alphabet: `0` and `1`

- Transitions:



- Start / Accept states:



In-class exercise

- Prove: the following language is a regular language:
  - $A = \{w \mid w \text{ has exactly three 1's}\}$
  - i.e., design a finite automata that recognizes it!

- Where $\Sigma = \{0, 1\}$,

- Remember:

DEFINITION

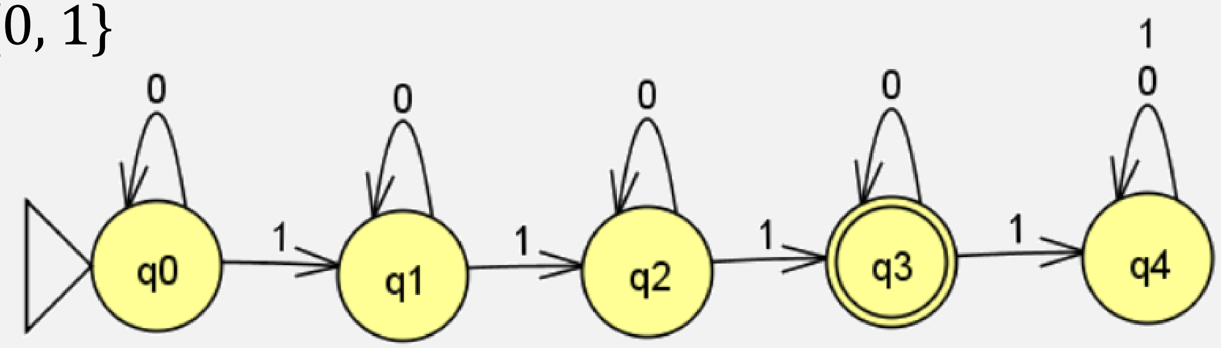
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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***.

In-class exercise Solution

- Design finite automata recognizing:
  - $\{w \mid w$ has exactly three 1's$\}$

- *States*:
  - Need one state to represent how many 1's seen so far
  - $Q = \{q_0, q_1, q_2, q_3, q_{4+}\}$
- *Alphabet*: $\Sigma = \{0, 1\}$
- *Transitions*:



- *Start state*:
  - q_0
- *Accept states*:
  - $\{q_3\}$

So finite automata are used to recognize simple string patterns?

Yes!

Have you ever used a programming language feature to recognize simple string patterns?

So Far: Finite State Automaton, a.k.a. DFAs

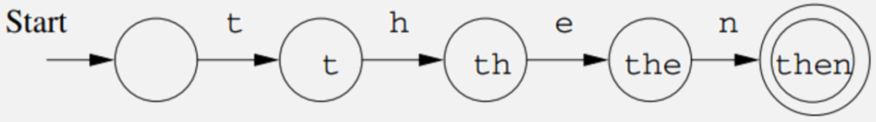
deterministic

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

1. Q is a $\boxed{\text{finite}}$ 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***.

- <u>Key characteristic</u>:
  - 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

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

To match <u>all</u> requirements, <u>combine</u> smaller DFAs into one big DFA?

(We do this with programs all the time)

Password Checker DFAs

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, i.e., <u>composability</u>

<u>We want these operations:</u>

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

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

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!

<u>We want:</u>

OR, AND : DFA $\times$ DFA $\rightarrow$ DFA

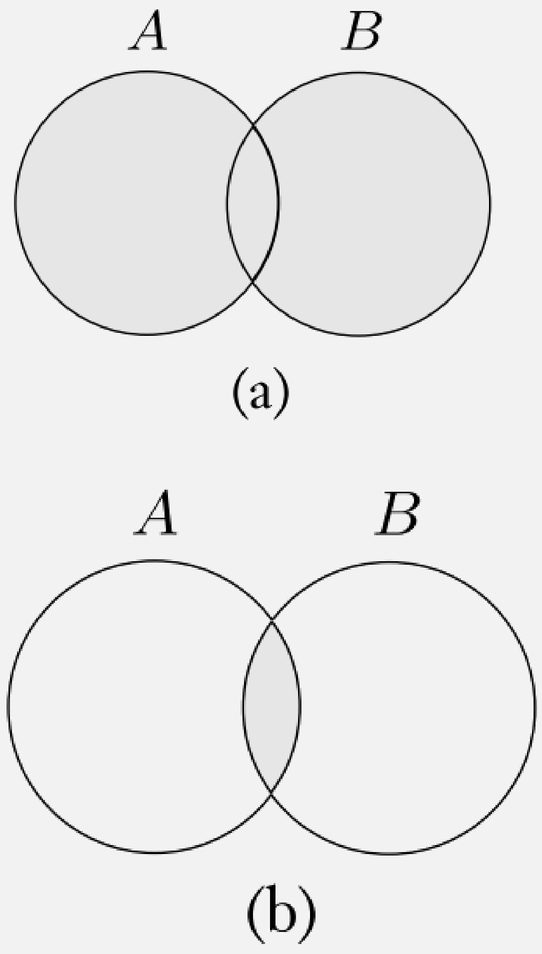
- So this semester, we will look for operations that are <u>closed</u>!

Password Checker: “**OR**” = “Union”

M_3: OR

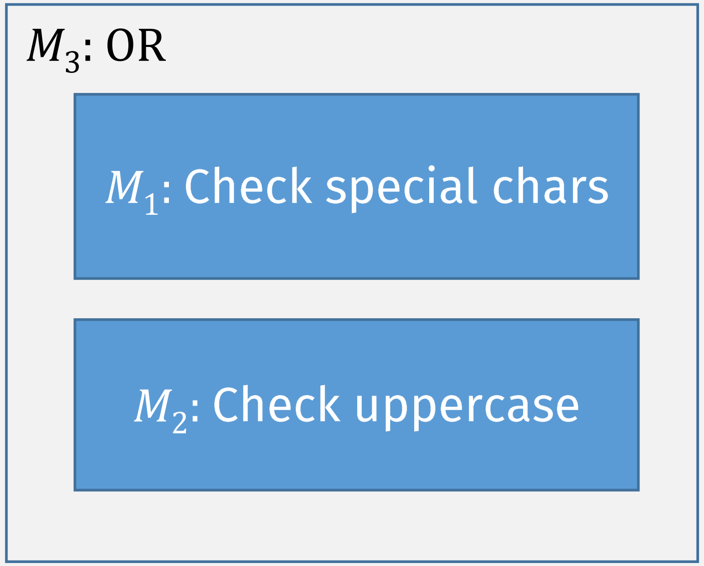
M_1: Check special chars

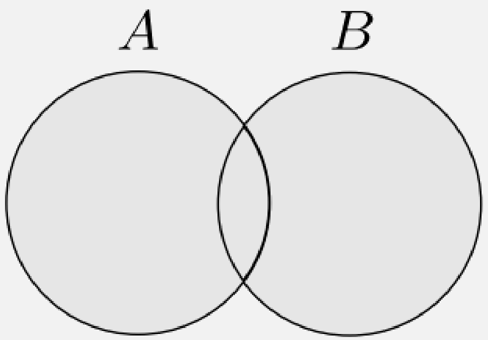
M_2: Check uppercase



???

Password Checker: "OR" = "Union"





(a)

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

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 $A_1 \cup A_2$.

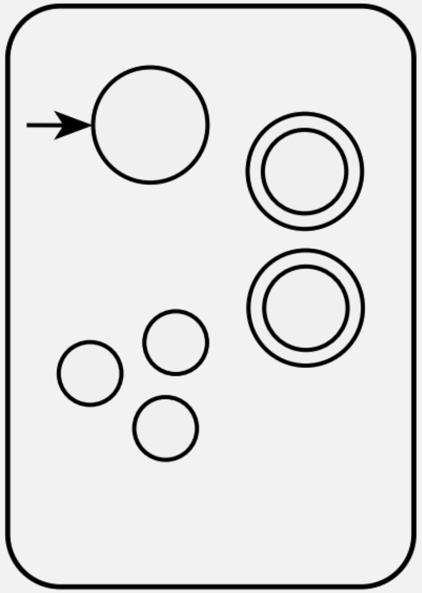
A language is called a ***regular language*** if some finite automaton recognizes it.

- 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

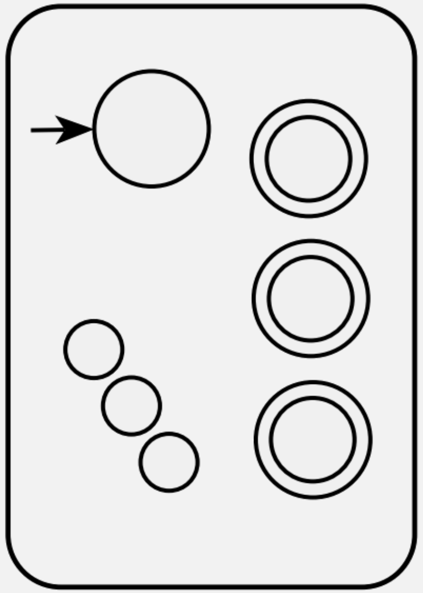
M_1

recognizes A_1



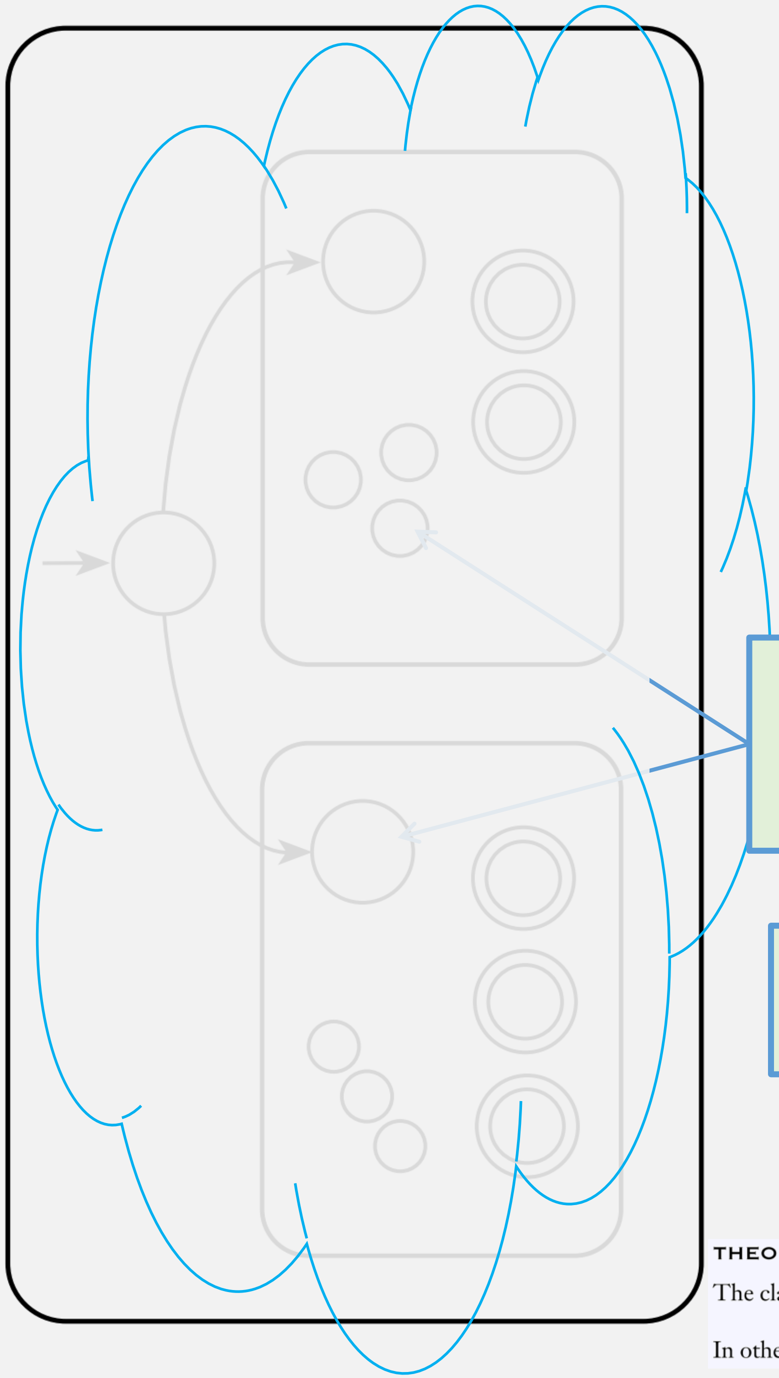
M_2

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

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,

Want: M "runs" its input on both M_1 and M_2 at the same time

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

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

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: $M = (Q, \Sigma, \delta, q_0, F)$, 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: $\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big)$ a step in M_1, a step in M_2
- M start state: (q_1, q_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,
- 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
- 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)

Remember:
Accept states must be subset of Q

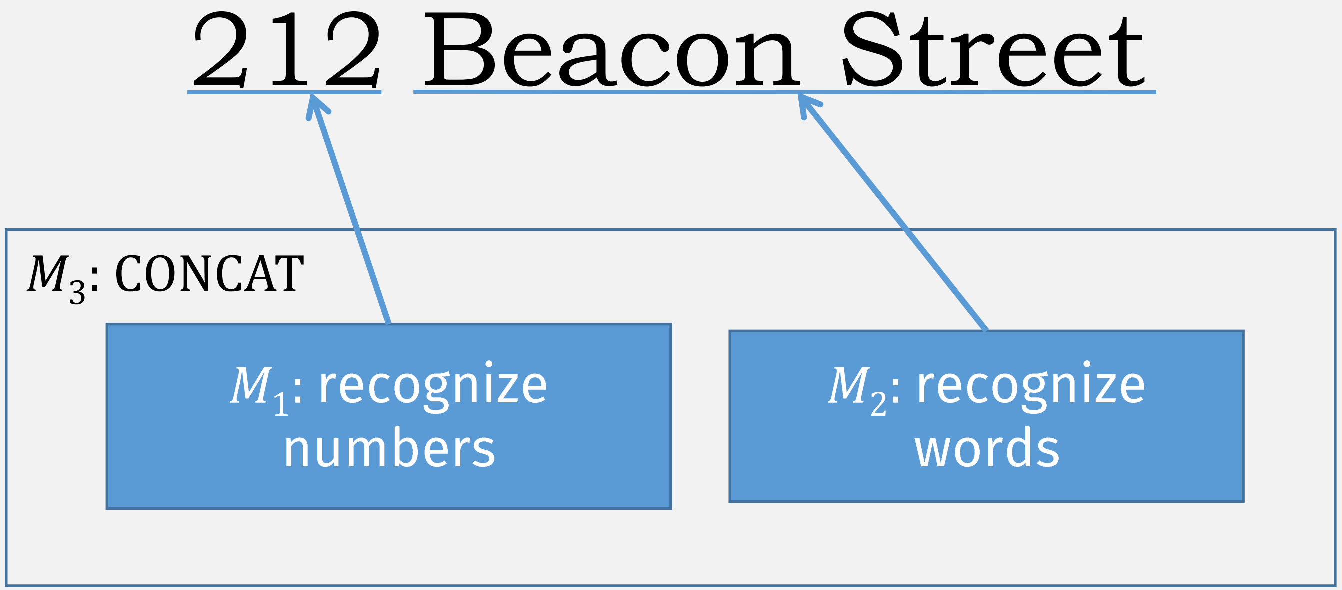
Accept if <u>either</u> M_1 or M_2 accept

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

(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

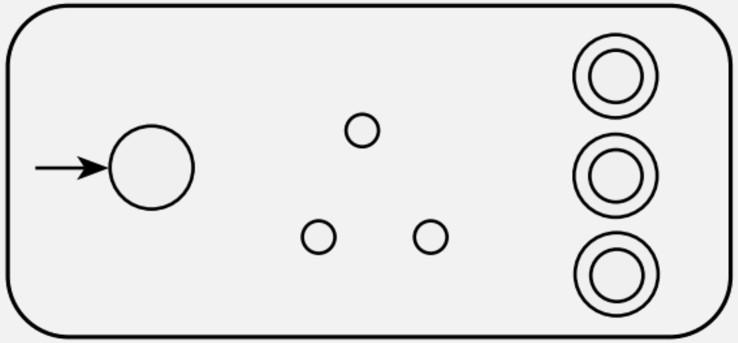
The class of regular languages is closed under the concatenation operation.

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

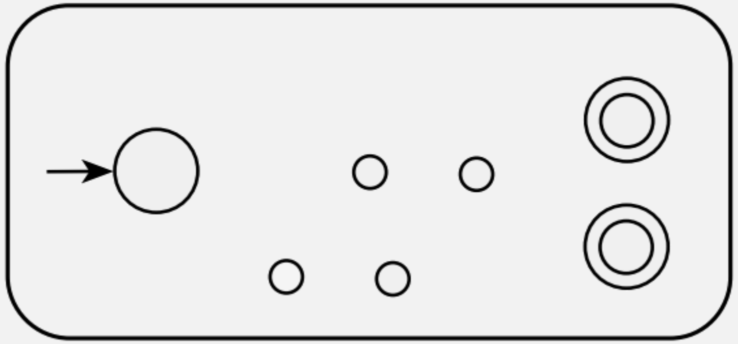
- Construct a new machine M recognizing $A_1 \circ A_2$? (like union)
  - From DFA M_1 (which recognizes A_1),
  - and DFA M_2 (which recognizes A_2)

Concatentation

M_1



M_2

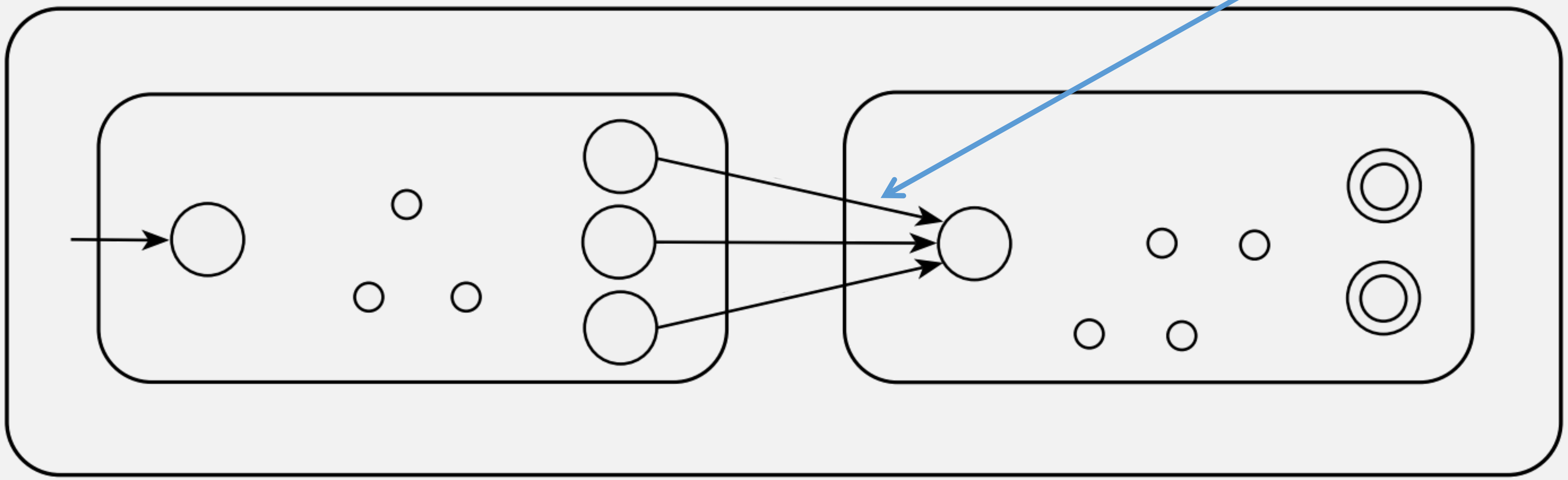


PROBLEM: Can only read input once, can't backtrack

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 $A \circ B$ = {$\boxed{\textbf{ab}}$**cde**, $\boxed{\textbf{ab}}$**ccde**}

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

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

Overlapping Concatenation Example

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

- **But if M sees ab as first part of input …**
- **M must decide to <u>either</u>:**
    - **stay in M_1** (correct, if full input is $\boxed{\textbf{abc}}\boxed{\textbf{cde}}$)

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 $A \circ B$** = {**abcde**, **abccde**}

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

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

Check-in Quiz 9/15

On gradescope