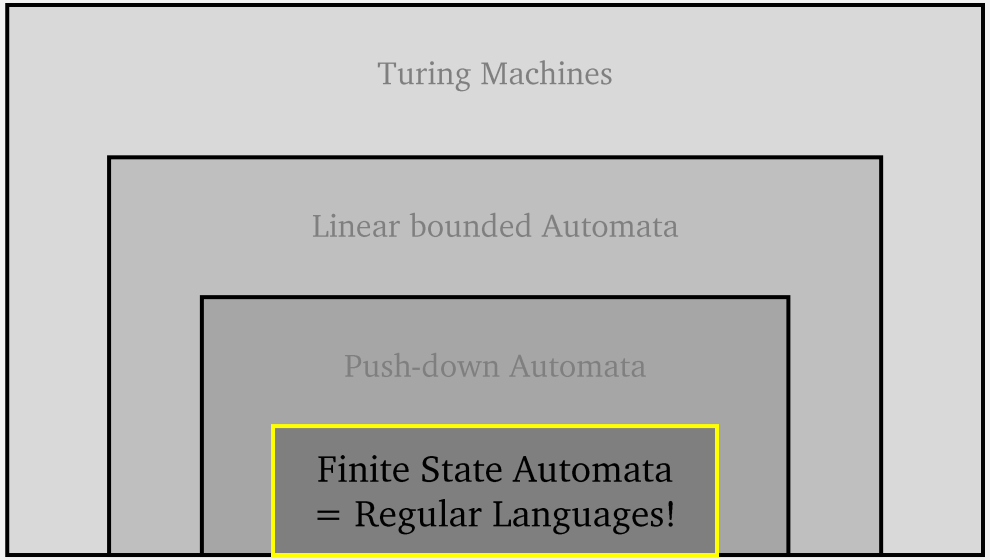
CS 420 / CS 620

Regular Languages

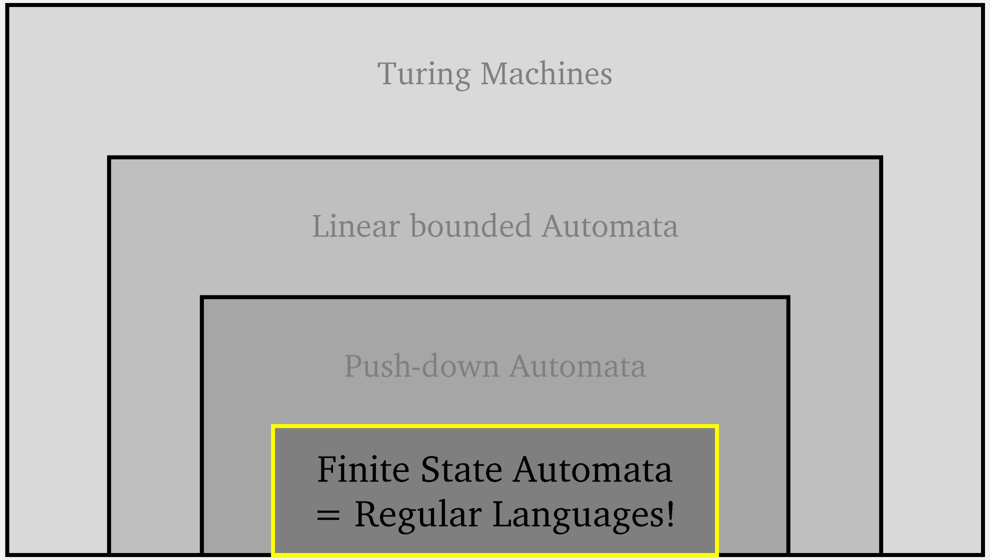
Wednesday, September 17, 2025

UMass Boston Computer Science



Announcements

- HW 2 out
  - Due: Mon 9/22 12pm (noon)



Last Time

Machine and Language Terminology

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

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

Machine and Language Terminology

- The **language** of a **machine** = set of strings that it **accepts**

A DFA M ***accepts*** w ← string

M ***recognizes language*** A ← Set of strings

if $A = \{w \mid M \text{ accepts } w\}$

"the set of all …" "such that …"

Machine and Language Terminology

- The **language** of a **machine** = set of strings that it **accepts**

DFA M ***accepts*** w

M ***recognizes language*** $L(M)$

$L(M) = \{w \mid M \text{ accepts } w\}$

L commonly used as **function mapping Machine → Language**

Machine and Language Terminology

- The **language** of a **machine** = <u>set of strings</u> that it **accepts**

DFA M ***accepts*** w

M ***recognizes language*** $L(M)$

- **Language** of $M = L(M) = \{w \mid M \text{ accepts } w\}$

Languages Are Computation Models

- The **language** of a **machine** = set of strings that it **accepts**
  - E.g., a **DFA recognizes** a **language**
- A **computation model** = set of machines it **defines**
  - E.g., **all possible DFAs** 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. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

= set of set of strings

Thus: a **computation model** equivalently = a set of languages

This class is really about studying **sets of languages!**

Regular Languages

- first **set of languages** we will study: **regular languages**

This class is <u>really</u> about studying **sets of languages!**

Regular Languages: Definition

Definition of Regular Language

If a **deterministic finite automata** (**DFA**) recognizes a **language**, then **that language** is called a **regular language**.

A Language, Regular or Not?

- If given: **a DFA M** "P"
  - We know: $L(M)$, the **language recognized by M,** is a **regular language** "Q"

Definition of Regular Language

Proof : If a **DFA** recognizes a **language**, "P"
then **that language** is called a **regular language.** "Q"

Justification?

(modus ponens)

- If given: a **Language** A
  - Is A a **regular language?**
    - Not necessarily!

Proof : ??????

Proving That a Language is Regular

Prove: A **language** L = { … } is a **regular language**

Proof:

Statements	**Justifications**
1. DFA $M = (Q, \Sigma, \delta, q_0, F)$ (TODO: actually define M) (no unbound variables!)	1. Definition of a DFA
2. **DFA** M **recognizes** L	2. TODO: ???
3. **If a DFA recognizes** L, **then** L **is a regular language**	3. Definition of a regular language
4. **Language** L **is a regular language**	4. Stmts 2 and 3 (and **modus ponens**)

"P"?

Modus Ponens

If we can prove these:

- If P then Q
- P

Then we've proved:

- Q

A Language: strings with odd # of 1s (2 min)

Come up with string examples (in a table), both
- in the language
- and **not** in the language

- **In-class exercise** (submit to gradescope):

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

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

If a DFA <u>recognizes</u> a language,
then **that language is called a regular language**.

How to prove the language is regular?

Prove there's a DFA recognizing it!

Proving That a Language is Regular

Prove: A **language** L = { … } is a **regular language**

Proof:

Statements	**Justifications**
1. DFA $M = (Q, \Sigma, \delta, q_0, F)$ (TODO: actually define M) (no unbound variables!)	1. Definition of a DFA
2. DFA M recognizes L	2. TODO: ???
3. If a DFA recognizes L, then L is a regular language	3. Definition of a regular language
4. Language L is a regular language	4. Stmts 2 and 3 (and **modus ponens**)

Designing Finite Automata: Tips

- **Input** is **read only once, one char at a time** (can't go back!)

- Must decide accept/reject after that

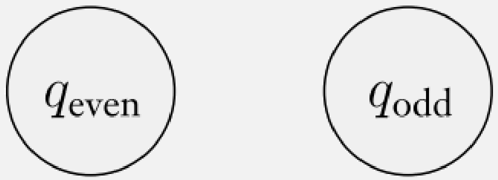
- States = the machine's "memory"!
  - # states must be decided in advance
  - Think about what information must be "remembered".

- Every state/symbol pair must have a defined transition (for DFAs)

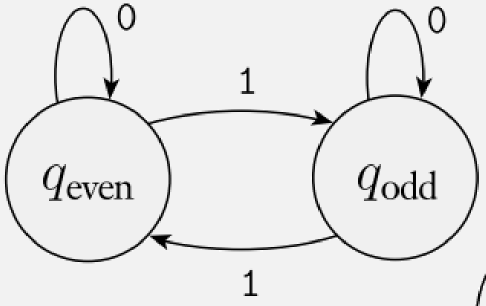
- Come up with examples to help you!

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

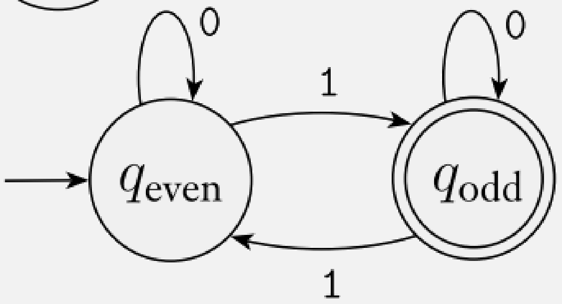
- <u>States</u>:
  - 2 states:
    - seen even **1**s so far
    - seen odds **1**s so far



- <u>Alphabet</u>: `0` and `1`
- <u>Transitions</u>:



- <u>Start</u> / <u>Accept</u> states:



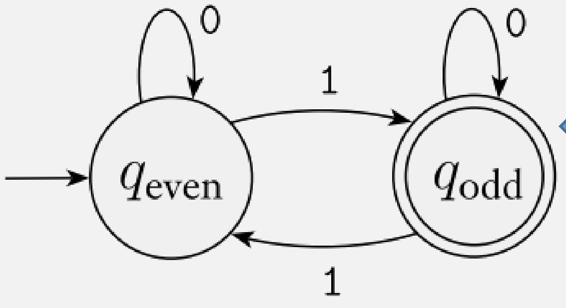
Proving That a Language is Regular

Prove: A **language** L = { … } is a **regular language**

Proof:

Statements	**Justifications**
☑ 1. DFA $M =$ See state diagram (only if problem allows!)	1. Definition of a DFA
2. DFA M recognizes L	2. TODO: ???
3. If a DFA recognizes L, then L is a regular language	3. Definition of a regular language
4. Language L is a regular language	4. Stmts 2 and 3 (and **modus ponens**)

"Prove" that DFA recognizes a language



???

Language = { w | w has odd # of **1**s}

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

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

Q1: How can we **"prove"** that a **computation** does what it's **"supposed to do"?**

These kinds of proofs (e.g., HMU 2.3.4) are usually **hard,** sometimes impossible!

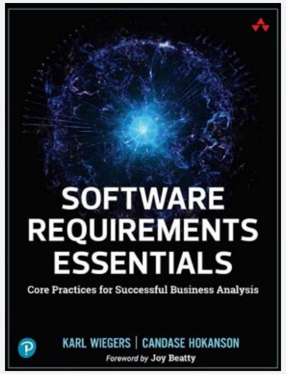
… so we will not do them in this course

But we still need to do something …

Q2: How do **programmers "prove"** that a **computation** does what it's **"supposed to do"?**

i.e., that the program is **"correct"**?

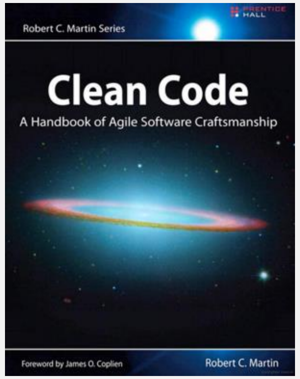
Interlude: Software Dev 101



Specification (i.e., **requirements**):

- What the **computation**, i.e., **program**, **should do**

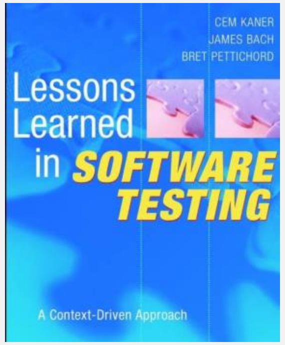
Implementation (i.e., the **code**):



- What the **program does**

Verification (i.e., **testing**):

- Does the **program do what it's supposed to?**
- (i.e., is the **program "correct"**?)



"Prove" that DFA recognizes a language

Verification
(does the program do what it's supposed to do?)

These **columns must match** for the **DFA** to **be "correct"!**

Confirm **the DFA:**
- **Accepts strings** in **the language**
- **Rejects strings** **not** in **the language**

- **In-class exercise** (part 2):

(2 min)

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

Not a **real** proof, but …

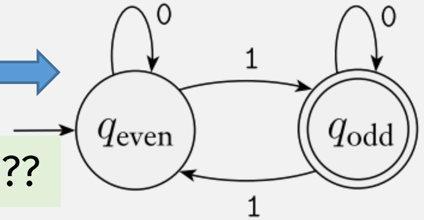
In this class, **a table like this is sufficient to "prove" that a DFA recognizes a language**

Analogous to **what programmers do** (write tests) to **"prove" their computation** (code) **"works"**

Specification
(what the **program should do**)

Implementation
(what the **program does**)

Language = { w | w has odd # of **1**s}



???

Proving That a Language is Regular

Prove: A **language** L = { ... } is a **regular language**

Proof:

Statements	**Justifications**
1. DFA $M =$	1. Definition of a DFA
See state diagram (only if problem allows!)	
2. DFA M recognizes L	☑ 2. See Examples Table
3. If a DFA recognizes L, then L is a regular language	3. Definition of a regular language
4. Language L is a regular language	4. Stmts 2 and 3 (and **modus ponens**)

Not a real proof, but ...

In this class, **an "Examples Table" is sufficient to "prove" that a DFA recognizes a language**

Another In-class exercise

Remember:
To understand the language, always come up with string examples first (in a table)! Both:
- in the language
- and not in the language

- Prove: the following language is a regular language:
  - $A = \{ w \mid w$ has exactly three 1's $\}$

You will need this later in the proof anyways!

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

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***.

Proving That a Language is Regular

Prove: A **language** L = { … } is a **regular language**

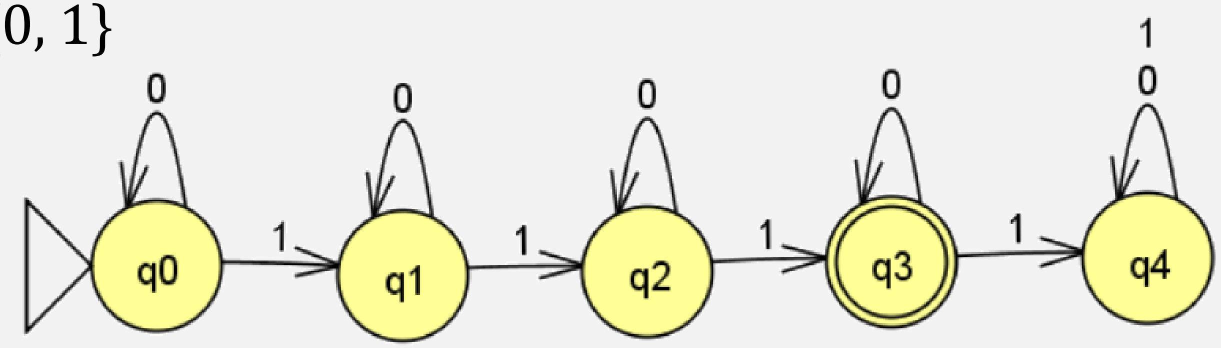
Proof:

Statements	**Justifications**
1. DFA $M = (Q, \Sigma, \delta, q_0, F)$ (TODO: actually define M) (no unbound variables!)	1. Definition of a DFA
2. DFA M recognizes L	2. TODO: ???
3. If a DFA recognizes L, then L is a regular language	3. Definition of a regular language
4. Language L is a regular language	4. Stmts 2 and 3 (and **modus ponens**)

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: a **DFA's computation** <u>**recognizes simple string patterns**</u>?

Yes!

Have you ever used **a programming language feature** for <u>**recognizing simple string patterns**</u>?

Regular Expressions!
(stay tuned!)

Programming Advice: Break Down Complex Problems

2. Break down the problem

Complexity is really just a bunch of simple problems chained together. Try to break down your task into smaller chunks that are more manageable.

https://dev.to/nuxt-wimadev/7-powerful-principles-to-tackle-complex-coding-problems-40a5

- Breaking big scary unknown problems into small manageable ones is a **core skill** for developers. And unlike syntax, it can't be easily learned from google.

This last point should be obvious to anyone who's been coding for a while.

https://medium.com/@dannysmith/breaking-down-problems-its-hard-when-you-re-learning-to-code-f10269f4ccd5

2. Break it down

After understanding the problem, the next process is to break down the problem into smaller sub-problems.

https://javascript.plainenglish.io/5-step-process-to-solve-complex-programming-problems-8e4f74cfd88e

… and then **combine the** (small) **solutions**

Combining Computation? (Programmers do this all the time)

Password Requirements

DFA

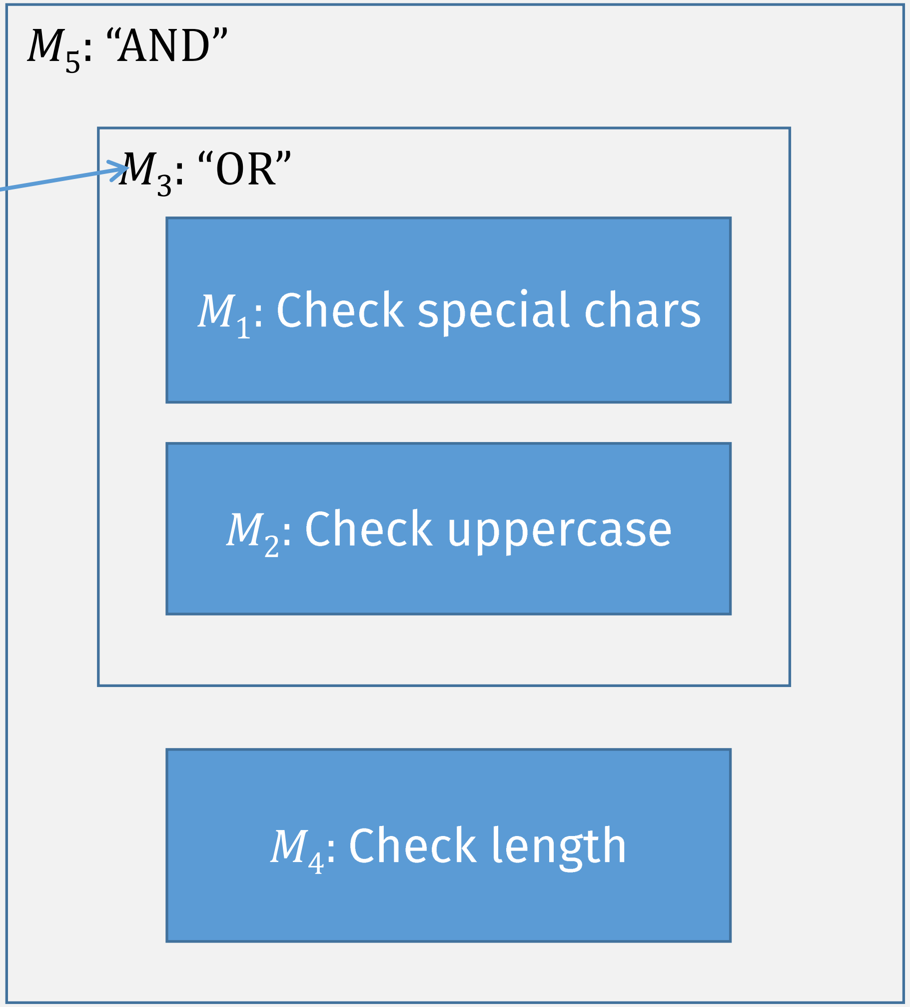
- Passwords must have a minimum length of ten (10) characters - but more is better!
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z) DFA
  - DFA lower-case letters (a-z)
  - symbols or special characters (%, &, *, $, etc.) DFA
  - numbers (0-9) DFA
- Passwords cannot contain all or part of your email address DFA
- Passwords cannot be re-used DFA

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

umb.edu/it/software-systems/password/

Password Checker DFAs

To <u>combine more than once</u>, this must be a DFA



Want 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

- 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

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

i.e., input set(s) = output set

Want: “Closed” Ops For Regular Langs!

- Set of Regular Languages = $\{L_1, L_2, \ldots\}$
  - Closed under …?
    - OR (union)
    - AND (intersection)
    - …

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

i.e., input set(s) = output set

Why Care About Closed Ops on Reg Langs?

- **Closed operations for regular langs preserve "regularness"**

- I.e., it <u>**preserves the same computation model**</u>**!**

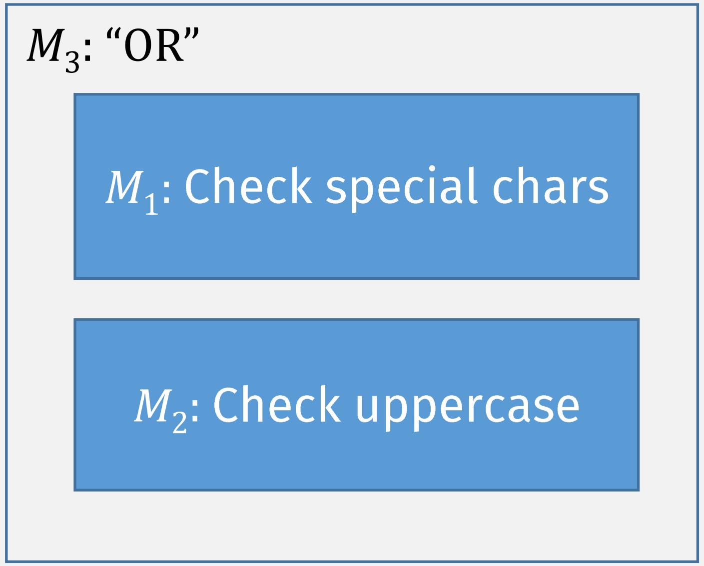
- Can **"combine" smaller "regular" computations** to get **bigger ones:**

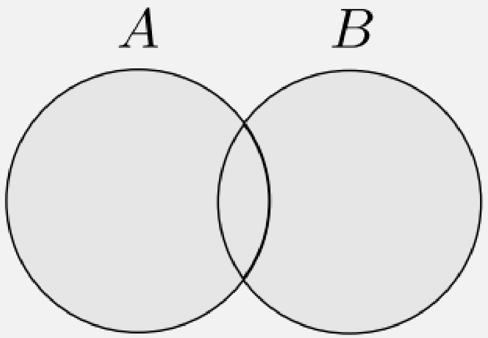
<u>For Example:</u>

OR: Regular Lang $\times$ Regular Lang $\rightarrow$ Regular Lang

- So this semester, **we will look for operations that are closed!**

Password Checker: “**OR**” = “Union”





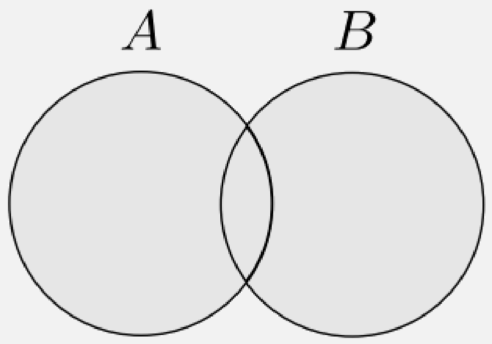
Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

Union of Languages

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

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

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



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**

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**

Is Union Closed For Regular Langs?

THEOREM

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

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

Want to prove this statement

Or this (same) statement

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

Using:

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

Proving:

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

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

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 $A_1 \cup A_2$.

t Q), or

• If P is TRUE, then Q is TRUE (**modus ponens**)

Would have to prove there are no regular languages (impossible)

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 lang **has a DFA**, then it's **regular**

Statements Do we know anything about A_1 and A_2?

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 $M = (Q, \Sigma, \delta, q_0, F)$ (todo)	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. S
6. $A_1 \cup A_2$ is a regular language	
7. The class of regular languages is closed under the union operation.	7. From stmt #1 and #6

???

???

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

Definition of Regular Language ???

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

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

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

Wait! “If A Then B” == “If B Then A” ??

(Actual) Definition of Regular Language

If a lang **has a DFA**, then it’s **regular**

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

Definition of Regular Language ???

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

Equivalence of Conditional Statements

- Yes or No? **“If X then Y” is equivalent to:**
  - “If Y then X” (**converse**)
    - No!

(Actual) Definition of Regular Language

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

If Regular, Then DFA?

- Prove: If L is a **regular language**, then a **DFA recognizes** L
- Proof (Sketch)

  Case analysis:
  - **Look at all** If-then **statements** of the form:
    - "If ... **language** L, then L is a **regular language**"
  - (At least one is true!)
  - **Figure out which one(s)** led to conclusion:
    - "L is a **regular language**"
  - (There's only 1!)

  - So it must be that: (because there was only 1 possible way to show that the language is regular)

"Corollary"

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

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

“Corollary”

???

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

Is Union Closed For Regular Langs?

Statements

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

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

???

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

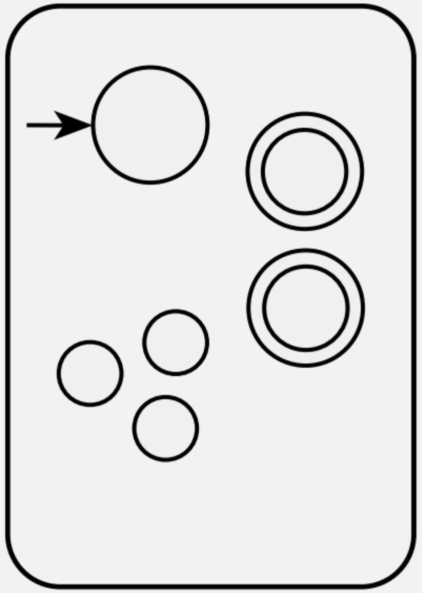
Justifications

1. Assumption
2. Def of Regular Language
3. Def of Regular Language
4. Def of DFA
5. See examples
6. Def of Regular Language
7. From stmt #1 and #6

"Corollary"

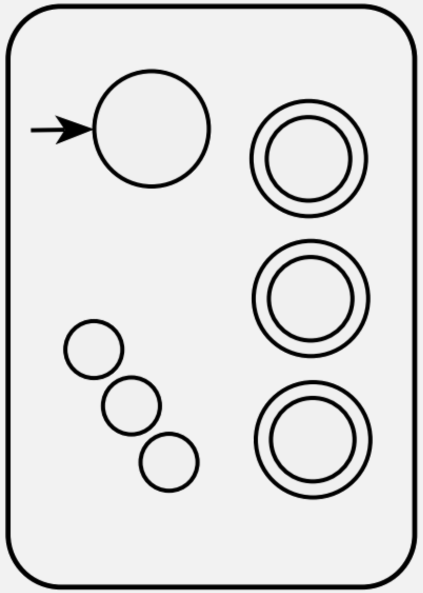
M_1

recognizes A_1



M_2

recognizes A_2



DEFINITION

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

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

Regular language A_1
Regular language A_2

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

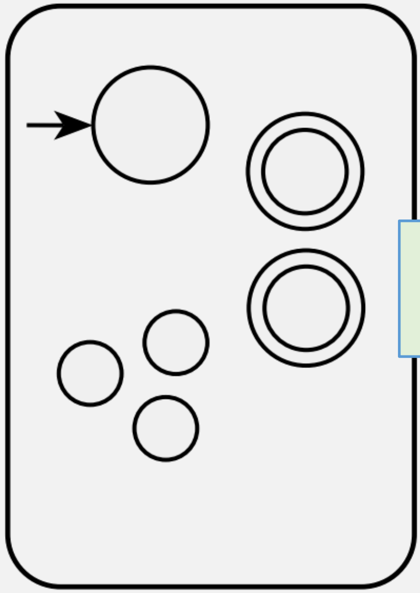
$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \text{ recognize } A_1,$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2), \text{ recognize } A_2,$$

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

Union

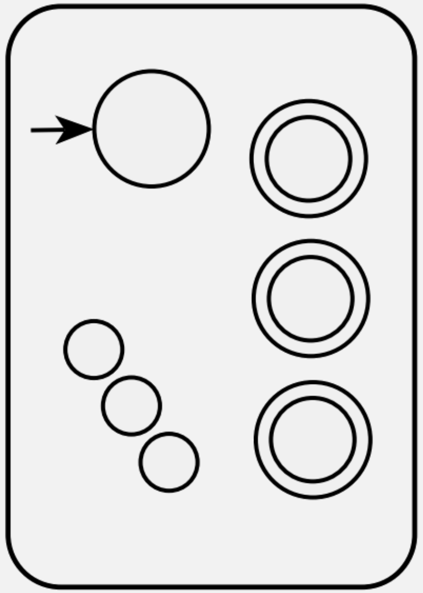
M_1

recognizes A_1



M_2

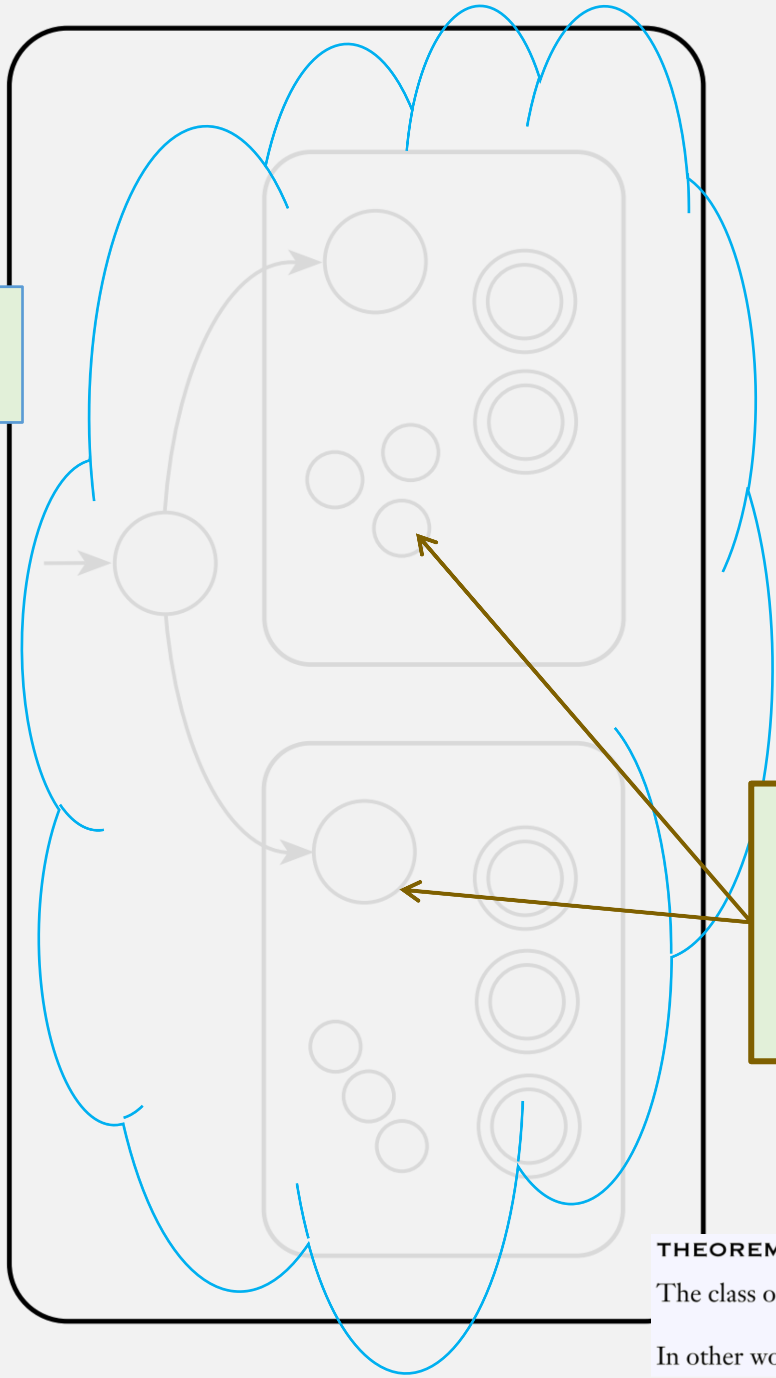
recognizes A_2



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

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

states of M: all possible pair combinations of states of M_1 and 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: $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  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. $\delta \colon Q \times \Sigma \longrightarrow Q$ 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 (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

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

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

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

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. $\boxed{\text{Construct DFA } M = (Q, \Sigma, \delta, q_0, F)}$ ☑	4. Def of DFA
5. M recognizes $A_1 \cup A_2$	5. See examples (TODO!)
6. $A_1 \cup A_2$ is a regular language	6. Def of Regular Language
7. The class of regular languages is closed under the union operation. In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.	7. From stmt #1 and #6

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

"Prove" that DFA recognizes a language

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

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

Be careful when choosing examples!

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

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

Don't know A_1 and A_2 exactly …

… but we know …

… they are **sets of strings**!

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

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

“Prove” that DFA recognizes a language

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

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

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

String	In lang $A_1 \cup A_2$?	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 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 ??	(J1)

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

constructed $M = (Q, \Sigma, \delta, q_0, F)$ to Accept if <u>either</u> M_1 or M_2 accept (J1)

Else **reject**

Previously

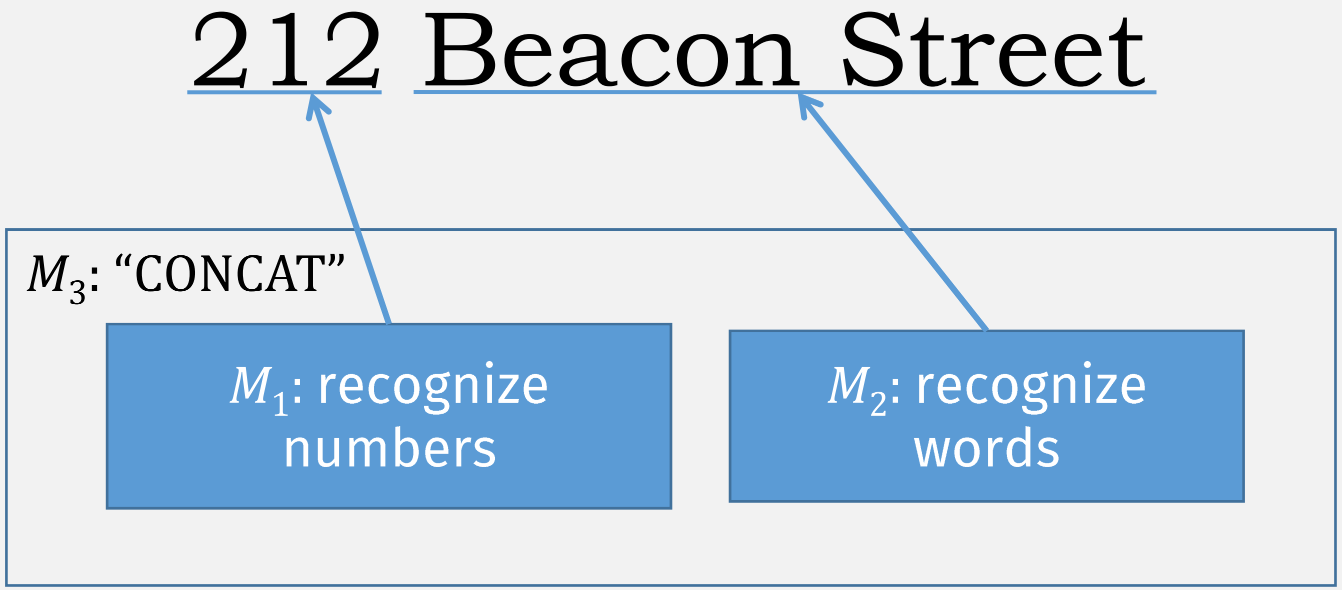
Is Union Closed For Regular Langs?

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

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

Is Concatenation Closed?

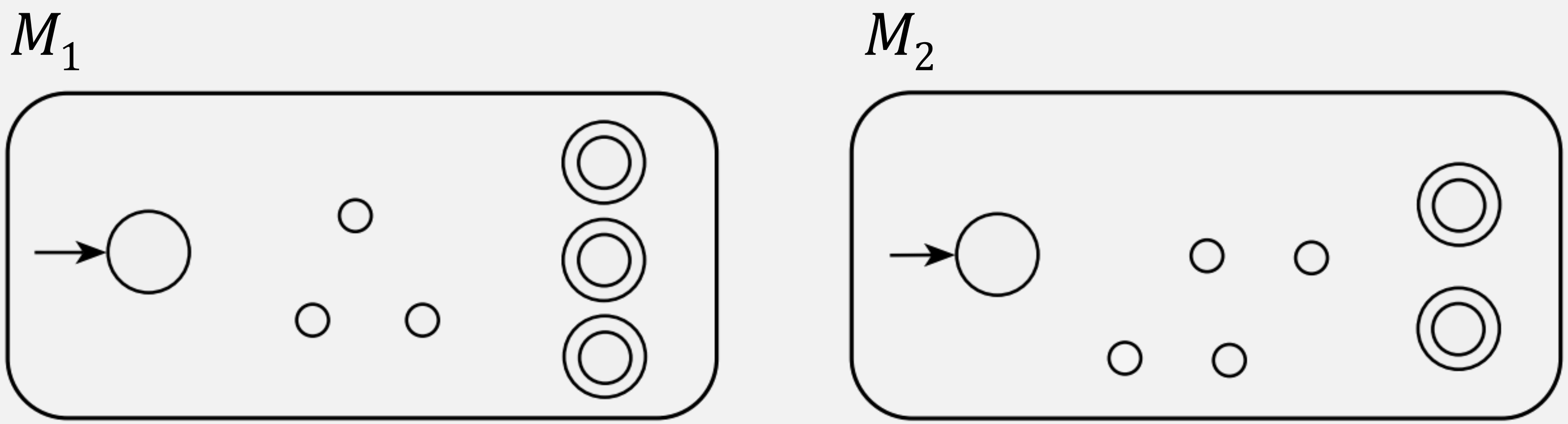
THEOREM

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

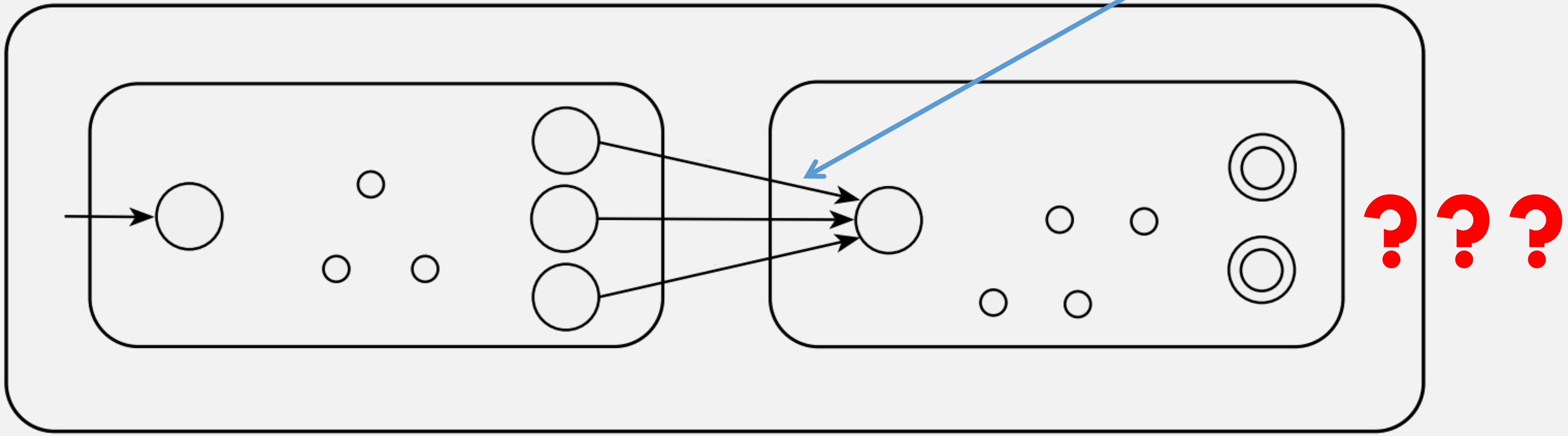


PROBLEM: Can only read input once, can't backtrack

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

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 either:
    - 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 <u>either</u>:
  - stay in M_1 (correct, if full input is `jenssmith`)
  - or switch to M_2 (correct, if full input is `jensmith`)
- <u>But</u> to recognize $A \circ B$, it needs to handle <u>both cases</u>!!
  - 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?**