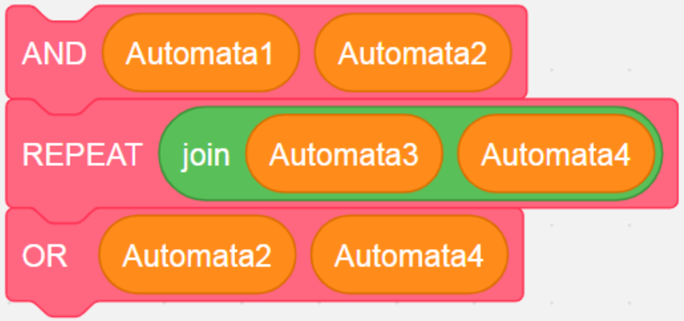
CS420

Combining Regular Languages

Wed Feb 3, 2021



What’s the Best Programming Language?

- Trick question! *Answer*: It depends on the application, obvi
- E.g., writing a …
  - … **Web App**? Use HTML + CSS + JS?
    - Or maybe TypeScript? And React? Or Angular?
  - … **Machine Learning Model**? Use R? or Python?
    - And NumPy? And Pandas? And PyTorch?
  - … **Video Game**? Use C++?
    - And Unity? Or Unreal engine?
- So a second best language should help programmers …
  - … Create new languages
  - … Tailor existing ones to fit a specific domain
  - … Use multiple languages together

HW Questions?

Last Time: In-class exercise

- Prove that this language is a regular language:
  - {w | w has exactly three 1's}
  - i.e., design a finite automata that recognizes it!

- Where Σ= {0, 1},

- Remember:

DEFINITION 1.5

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***.

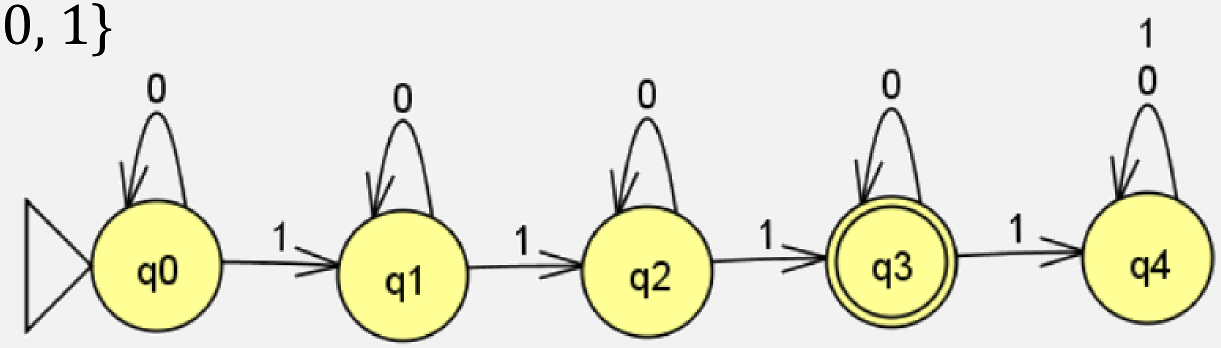
Last Time: In-class exercise

- Design finite automata recognizing:
  - {w | w has exactly three 1's}

So finite automata are used to recognize dumb patterns in strings???

Yes!

- *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*:
  - $\{q_3\}$

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

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:
  - upper-case letters (A-Z)
  - lower-case letters (a-z)
  - symbols or special characters (%, &, *, $, etc.)
  - numbers (0-9)
- Passwords cannot contain all or part of your email address
- Passwords cannot be re-used

State machine

State machine

State machine

State machine

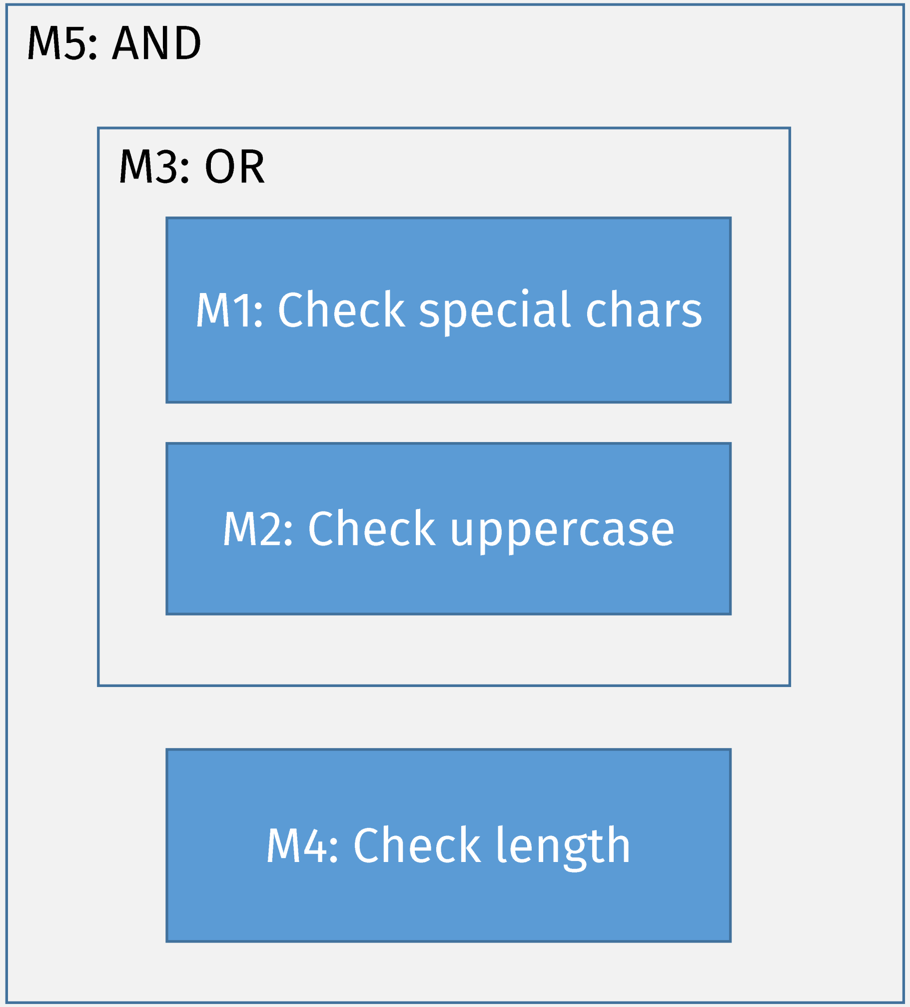
State machine

State machine

State machine

How to combine them together?

Password checker



Want to be able to easily combine finite automata machines

To keep combining, operations must be **closed**!

“Closed” Operations

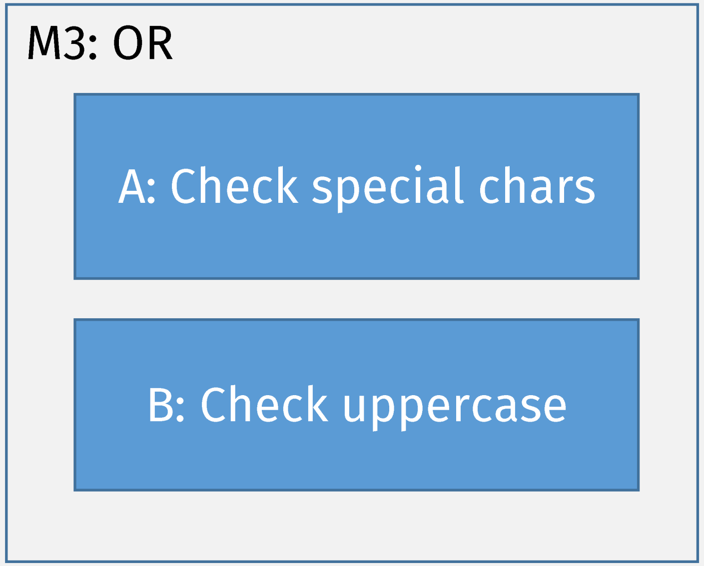
- Natural numbers = {0, 1, 2, …}
  - Closed under addition: if x and y are Natural, then z = x + y is a Nat
  - Closed under multiplication?
    - yes
  - Closed under subtraction?
    - no
- Integers = {…, -2, -1, 0, 1, 2, …}
  - Closed under addition and multiplication
  - Closed under subtraction?
    - yes
  - Closed under division?
    - no
- Rational numbers = {x | x = y/z, y and z are ints}
  - Closed under division?
    - No?
    - Yes if z !=0

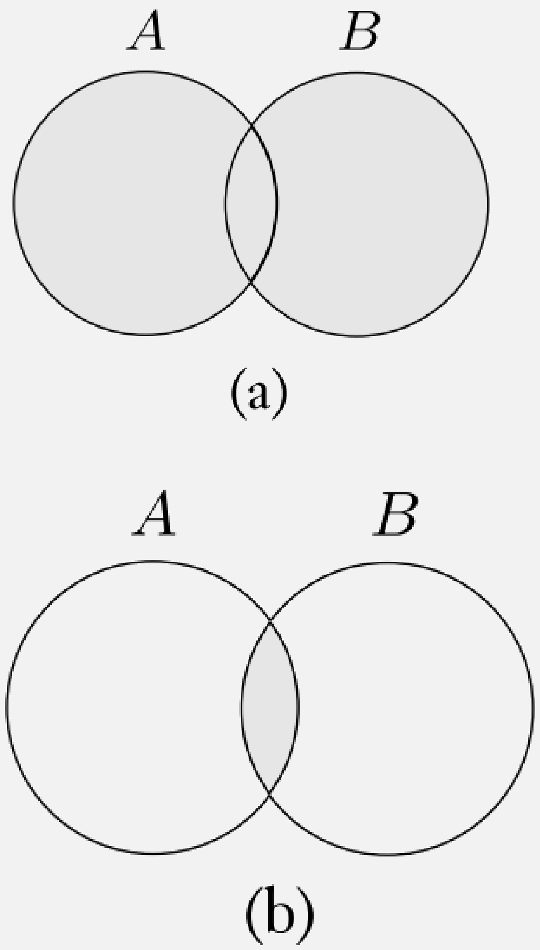
Any set is **closed** under some operation if applying that operation to members of the set returns an object still in the set.

Why Care About Closed Ops on Reg Langs?

- Closed operations preserves “regularness”
- I.e., it preserves the same computation model
- So result of combining machines can be combined again

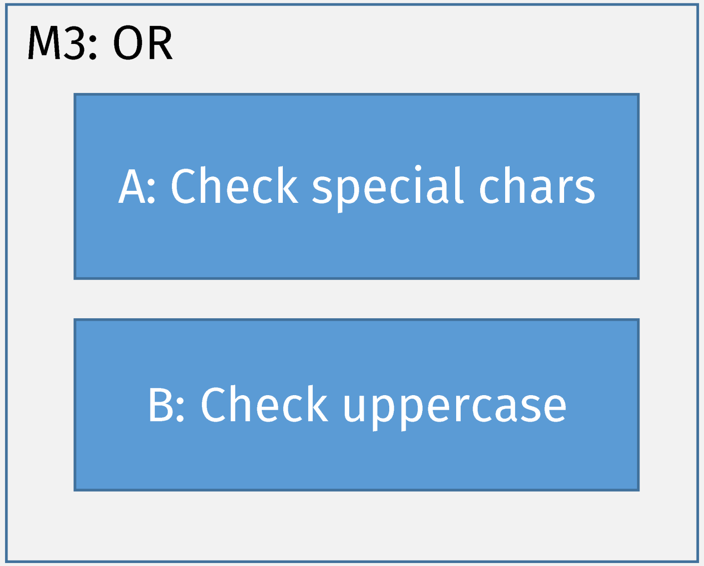
Password checker: “Or” = “Union”

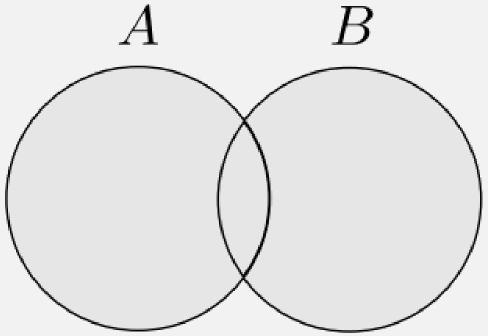




???

Password checker: “Or” = “Union”





(a)

A Closed Operation: Union

THEOREM 1.25

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 do we prove that a language is regular?
  - Create a FSM recognizing it!
- Create machine combining machines recognizing A_1 and A_2.

Kinds of Mathematical Proof

- Proof by construction
  - Construct the mathematical object in question
- Proof by contradiction
- Proof by induction

Union Closed?

THEOREM 1.25

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 $A_1 \cup A_2$.

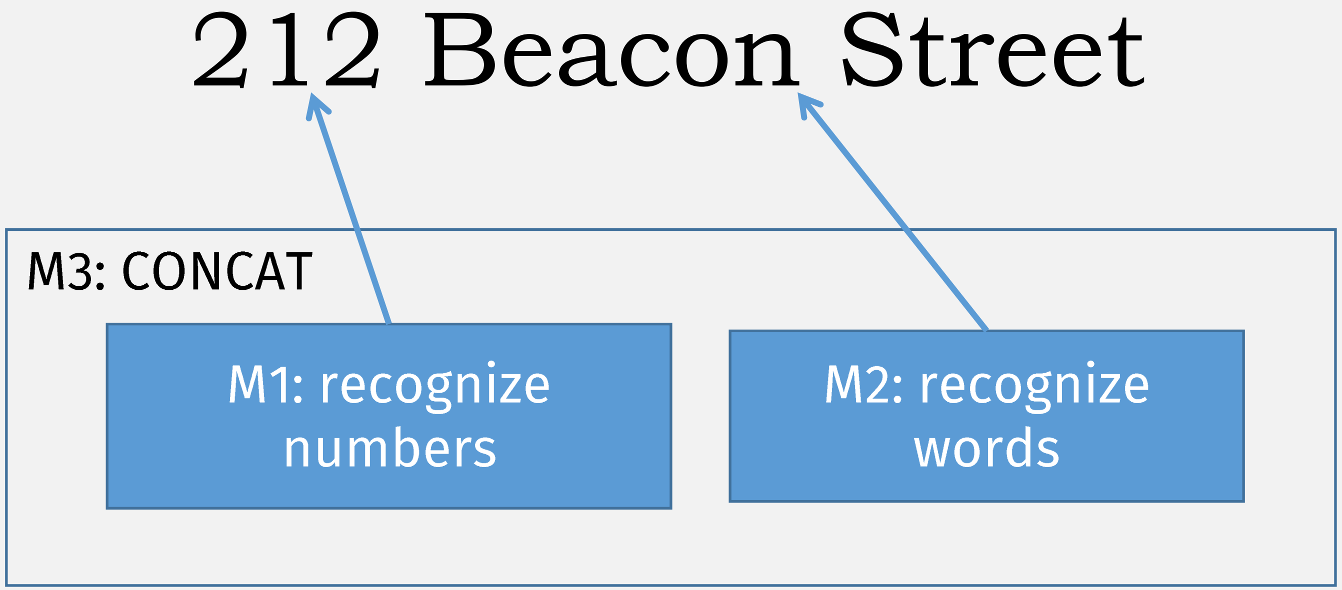
Proof (implement for hw2)

M runs its input on both M_1 and M_2 in parallel; accept if either accepts

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

Another operation: Concatenation

- Example: Matching street addresses



Is Concatenation Closed?

THEOREM 1.26

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? (like union)
  - From DFA M_1 (which recognizes A_1),
  - and DFA M_2 (which recognizes A_2)

Is Concatenation Closed?

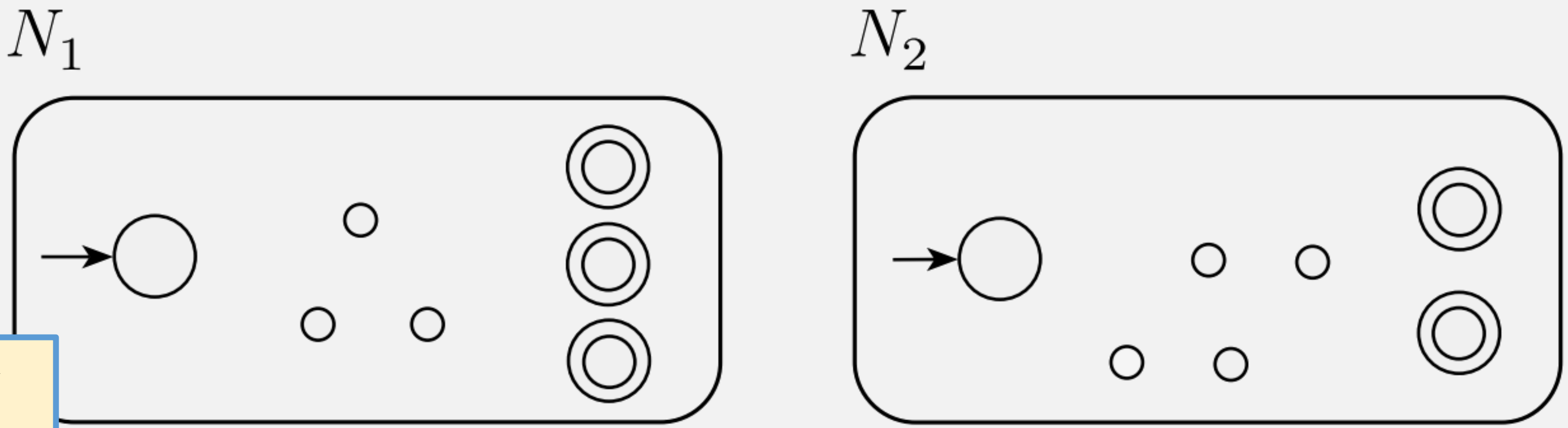
THEOREM 1.26

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 $A_1 \circ A_2$.

- Can't directly combine A_1 and A_2
  - don't know when to switch from A_1 to A_2 (can only read input once)
- Need a new kind of machine!
- So is concatenation not closed for reg langs???

Concatentation



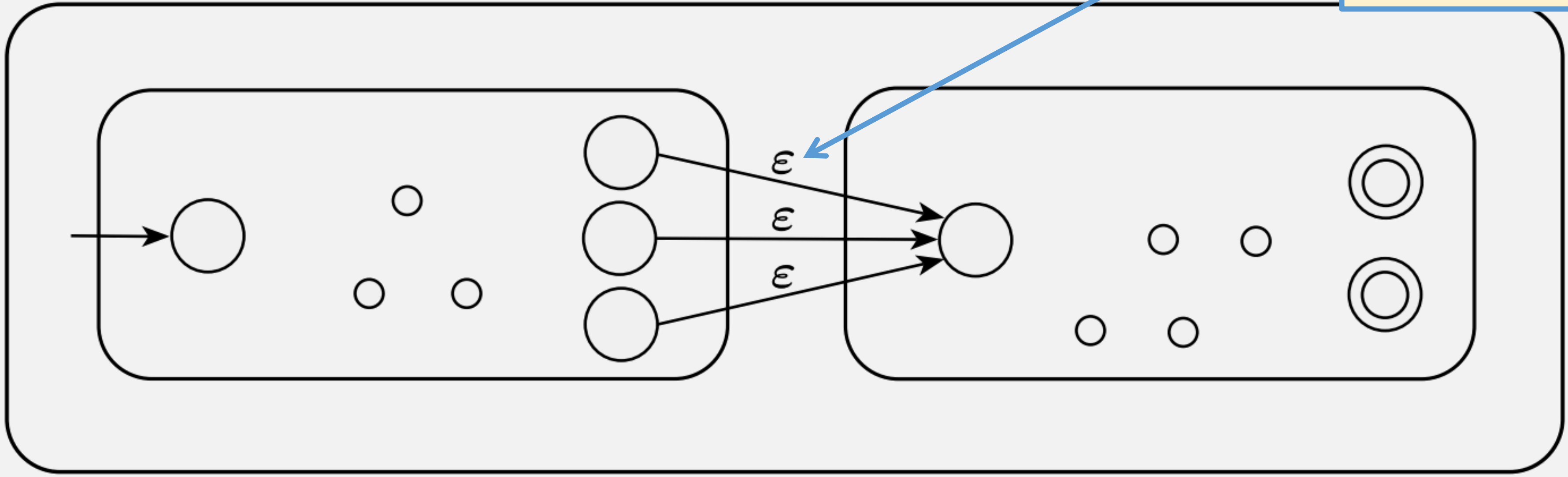
N is a new kind of machine, an NFA!
(next time)

Let N_1 recognize A_1, and N_2 recognize A_2.

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

ε = empty string = no input

So *N* can:
- stay in current state **and**
- move to next state



Check-in Quiz 2/3

On gradescope