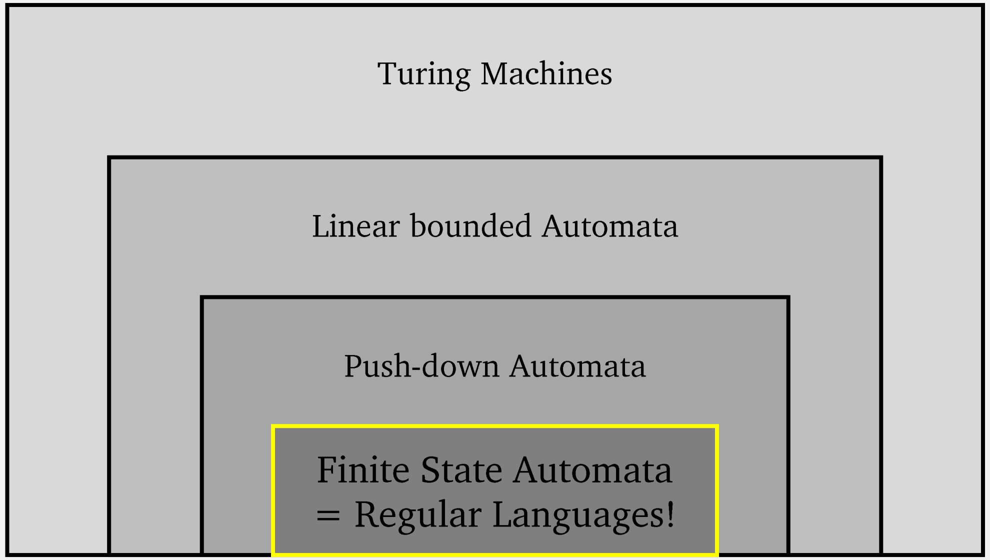
CS420

Regular Languages

Wednesday, February 1, 2023

UMass Boston Computer Science



Announcements

- HW 0 in
  - ~~Due Tues 1/31 11:59pm EST~~
- HW 1 out
  - Due Tues 2/7 11:59pm EST
- Quiz preview:
  Why do we know that a language is a regular language if it has an FSM recognizing it?

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

Last Time: Computation and Languages

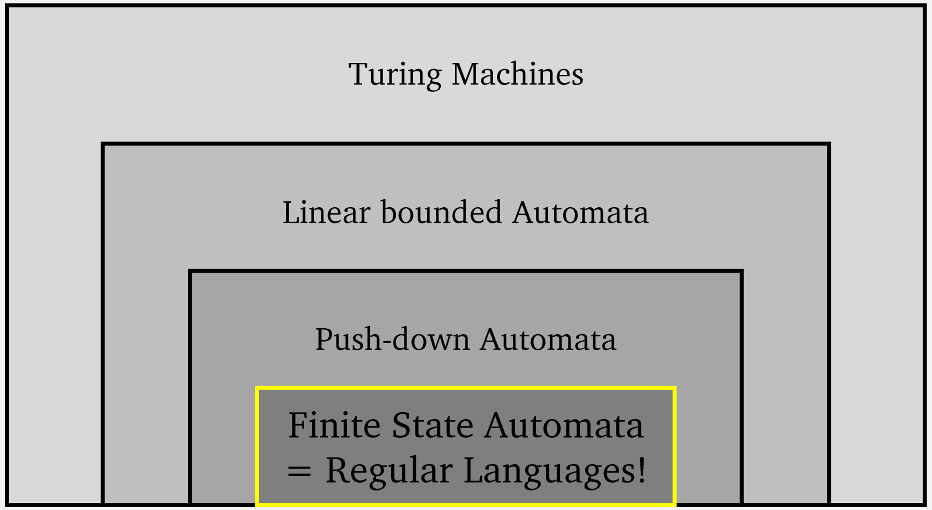
- The **language of a machine** is the <u>set of all strings that it accepts</u>

- A **computation model** is **equivalent to** the <u>set of machines</u> **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. $\delta : Q \times \Sigma \longrightarrow Q$ 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 <u>**set of languages**</u>

Last Time: Regular Languages: Definition

If a finite automaton (FSM) **recognizes** 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.**

(and modus ponens)

- If given: **a Language A**
  - **Is A 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

Example **Premises**

We know this
(definition of regular language)

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

… then we need to show

Conclusion

- A **is a regular language!**

If we want to prove this …

Proving a Language is Regular: Example

Prove that the following language is regular:

$$L = \{\, w \mid w \text{ is a string with an odd \# of } \mathtt{1}\text{s} \,\}$$

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

Proving a Language is Regular: Example

Statements	**Justifications**
1. If an FSM recognizes L, then L is a regular language	1. Def. of a Regular Language
2. $M = (Q, \Sigma, \delta, q_0, F)$ is an FSM (todo)	2. Definition of an FSM
3. M recognizes L	3. This is hard problem! In this class, we use tests.
4. $L = \{ w \mid w$ is string with odd # of `1`s$\}$ is a regular language	4. Stmt #1 & #3 (modus ponens)

When programming, how do you "prove" your program does what it is supposed to do?

Tips on Designing Finite Automata

<u>Analogy</u>
Finite Automata ~ "Programs" ::
Designing Finite Automata ~ "Programming"!

In programming, to "understand" a problem, create examples!

1. <u>Confirm understanding</u> of the problem
   - Create tests: examples and expected results (**`accept`** / **`reject`**)

FSM M Examples: accept strs with odd # **1**s

- On input `1`:
  - **Accept**
- On input `0`:
  - **Reject**
- On input `01`:
  - **Accept**
- On input `11`:
  - **Reject**
- On input `1101`:
  - **Accept**
- On input ε
  - **Reject**

Tips on Designing Finite Automata

Analogy
Finite Automata ~ “Programs” ::
Designing Finite Automata ~ “Programming”!

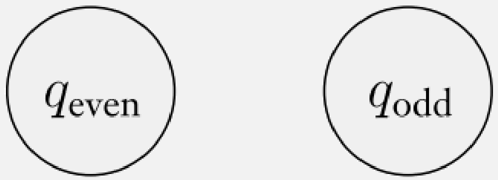
1. Confirm understanding of the problem
   - Create tests: examples and expected results (`accept` / `reject`)

2. Decide information to “remember”
   - These are the machine states: some are accept states; one is start state

3. Determine transitions between states

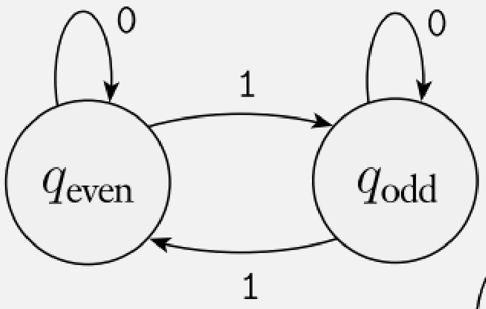
Designing FSM M: accept strs with odd # **1**s

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

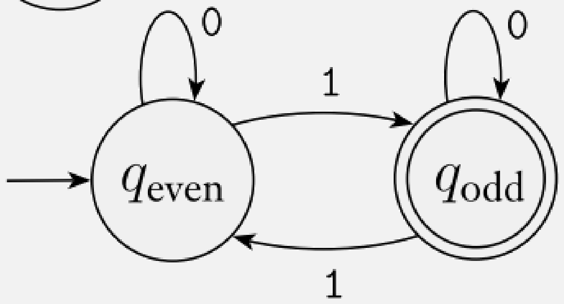


- Alphabet: `0` and `1`

- Transitions:



- Start / Accept states:

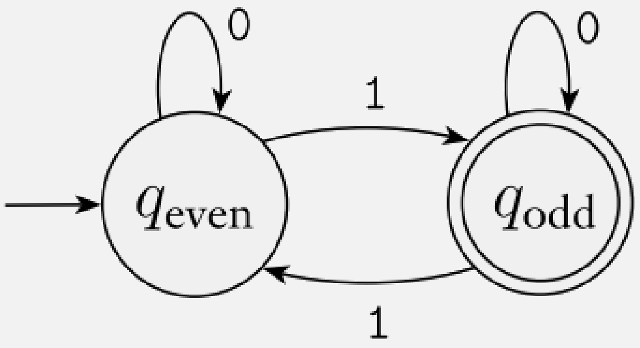


Tips on Designing Finite Automata

Analogy
Finite Automata ~ "Programs" ::
Designing Finite Automata ~ "Programming"!

1. Confirm understanding of the problem
   - Create tests: examples and expected results (`accept` / `reject`)
2. Decide information to "remember"
   - These are the machine states: some are accept states; one is start state
3. Determine transitions between states
4. Test machine behaves as expected
   - Use initial examples; and create additional tests if needed

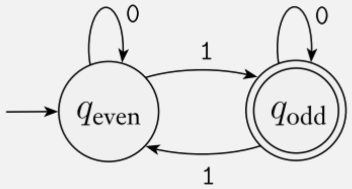
Does the Machine Accept Expected Strings?



- On input `1`: **??**
  - **Accept**
- On input `0`:
  - **Reject**
- On input `01`:
  - **Accept**
- On input `11`:
  - **Reject**
- On input `1101`:
  - **Accept**
- On input ε
  - **Reject**

Proving a Language is Regular: Example

Statements

1. If an FSM recognizes L,
   then L is a regular language
2. M =  is an FSM

3. M recognizes L
4. $L = \{ w \mid w$ is string with odd # of `1`s$\}$
   is a regular language

Justifications

1. Def. of a Regular Language
2. Definition of an FSM
3. See examples. This isn't a proof, but good enough for programmers(?), and CS 420
4. Stmt #1 & #3 (modus ponens)

■

In-class exercise

- <u>Prove:</u> the following language is a regular language:
  - $A = \{w \mid w$ has exactly three $\mathbf{1}$'s$\}$
  - Key step: **design a finite automata that recognizes it!**

Come up with examples first!

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

- 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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

In-class exercise Solution

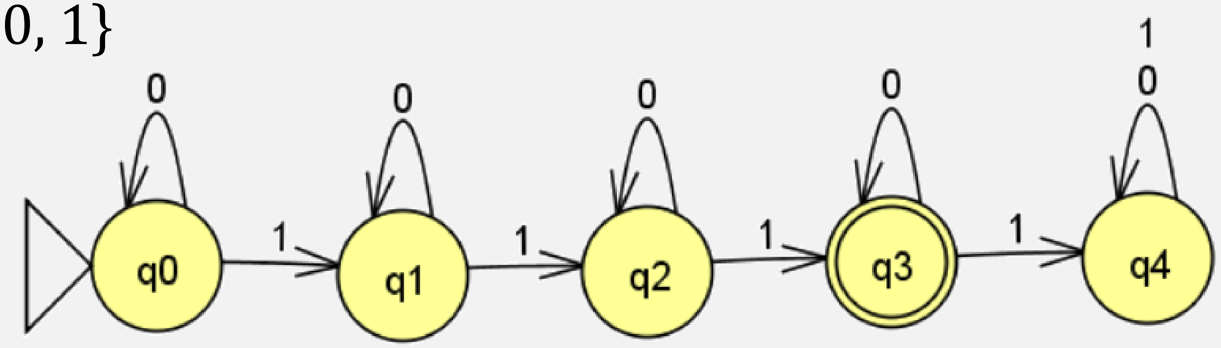
- Design finite automata recognizing:
  - $\{w \mid w \text{ has exactly three 1's}\}$

So finite automata are used to **recognize** <u>simple string patterns</u>?

Yes!

Do you know a "programming language" to <u>recognize simple string patterns</u>?

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

Make sure to test this with your examples!

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

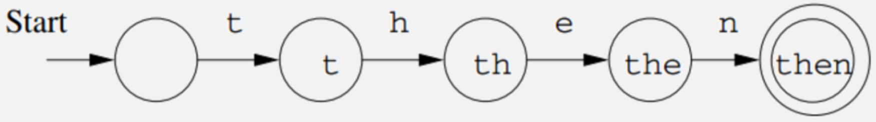
deterministic

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

1. Q is a 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 "program" with access to only a single cell of memory,
    - Where: states = the possible values that can be written to memory

- Often used for **text matching**



Combining DFAs?

Password Requirements

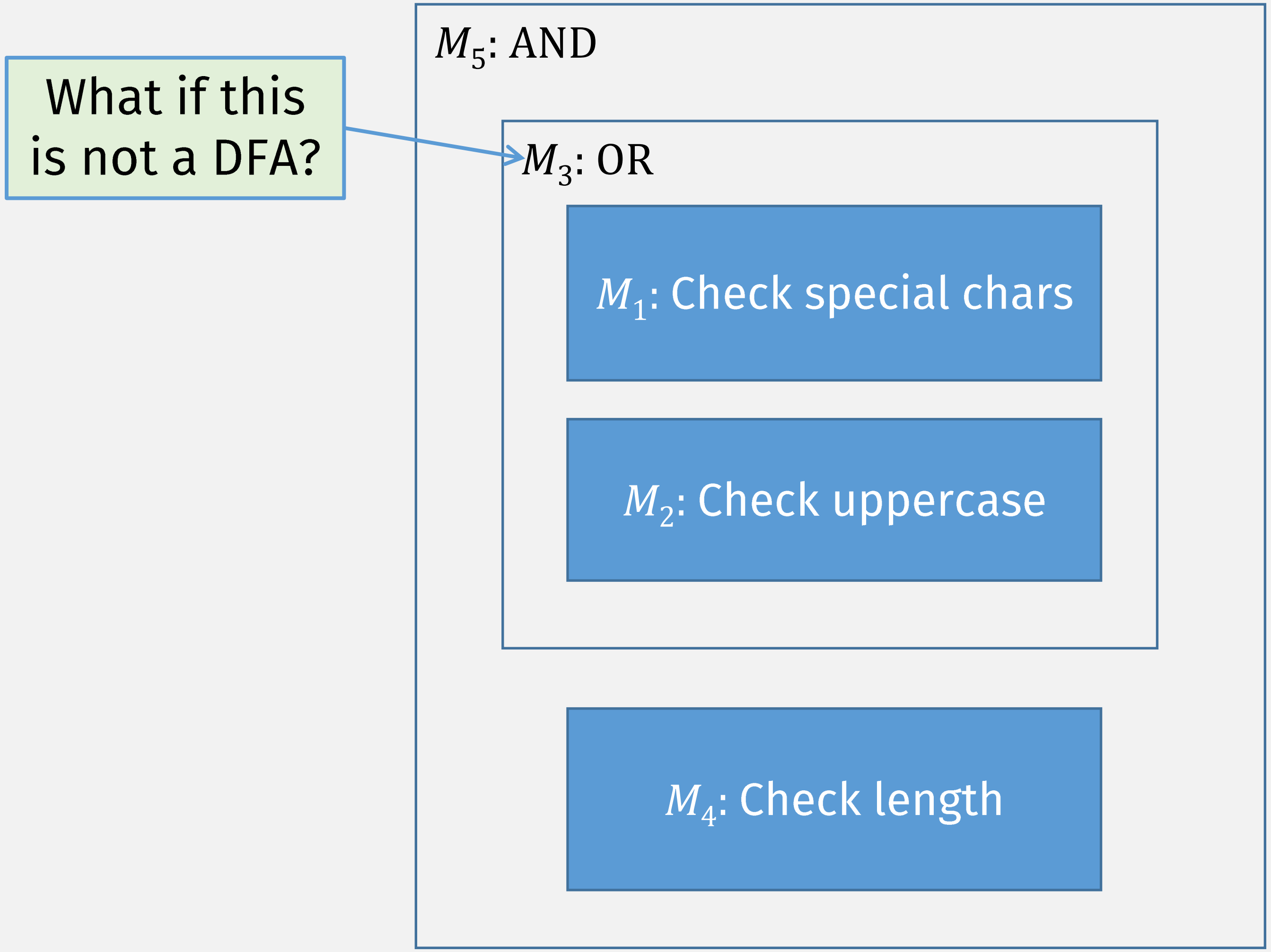
- Passwords must have a minimum length of ten (10) characters - but more is better! ← DFA
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z) ← DFA
  - lower-case letters (a-z) ← DFA
  - symbols or special characters (%, &, *, $, etc.) ← DFA
  - numbers (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 all requirements, combine smaller DFAs into one big DFA?

(We do this with programs all the time)

Password Checker DFAs



Want to be able to easily combine DFAs, i.e., composability

We want these operations:

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

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

To combine more than once, operations must be **closed**!

“Closed” Operations

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

- Set of Natural numbers = $\{0, 1, 2, ...\}$
  - <u>Closed</u> 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 = $\{..., -2, -1, 0, 1, 2, ...\}$
  - <u>Closed</u> 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?

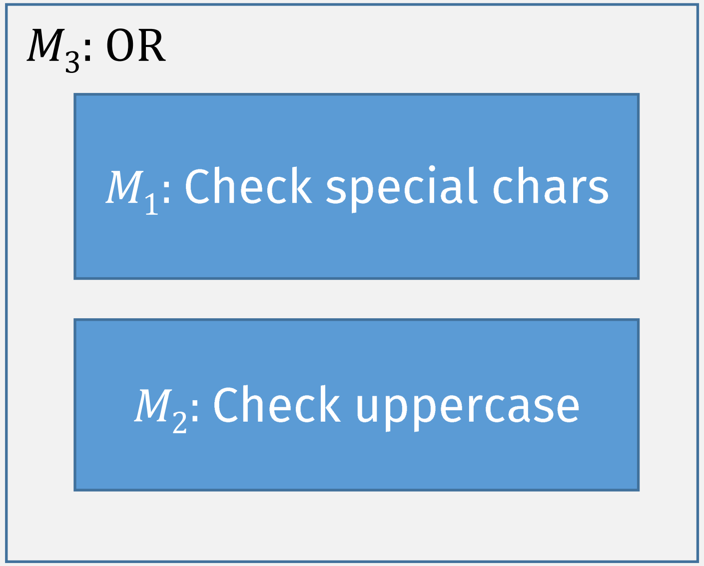
- Closed operations preserve “regularness”
- I.e., it preserves the same computation model!
- This way, a “combined” machine can be “combined” again!

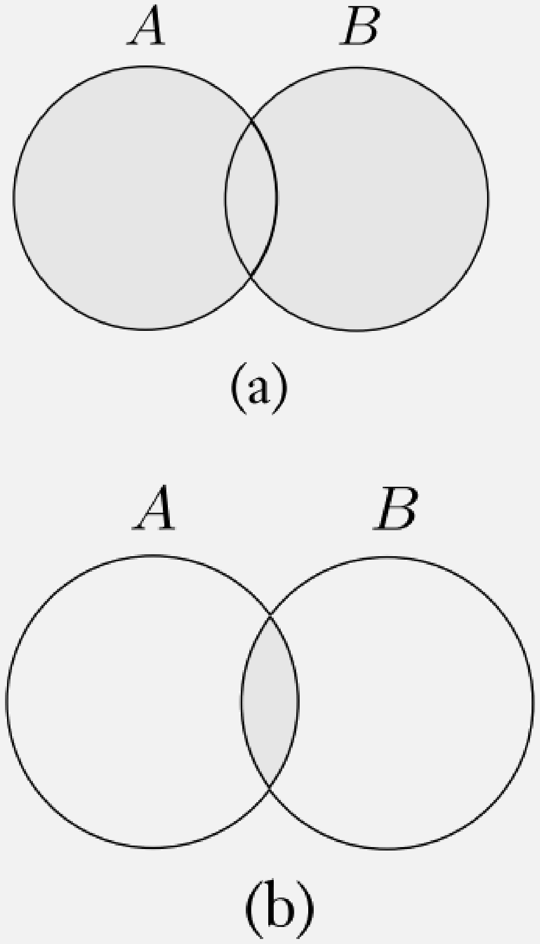
<u>We want:</u>

OR, AND : DFA × DFA → DFA

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

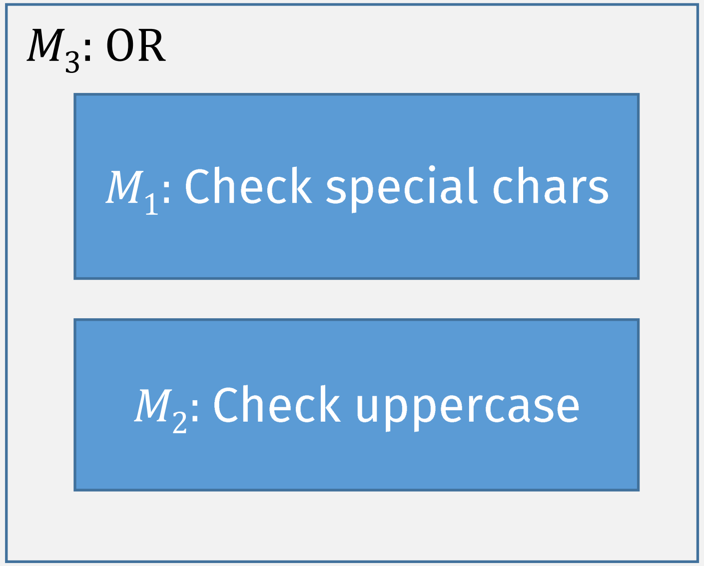
Password Checker: "**OR**" = "Union"

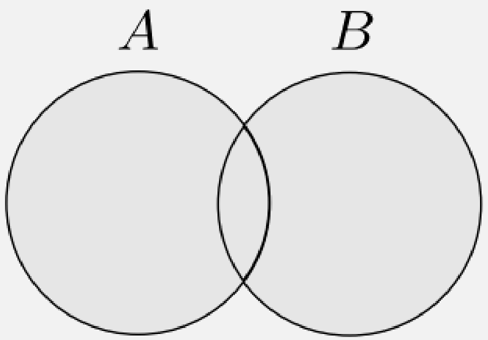




???

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

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

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

Check-in Quiz 2/1

On gradescope