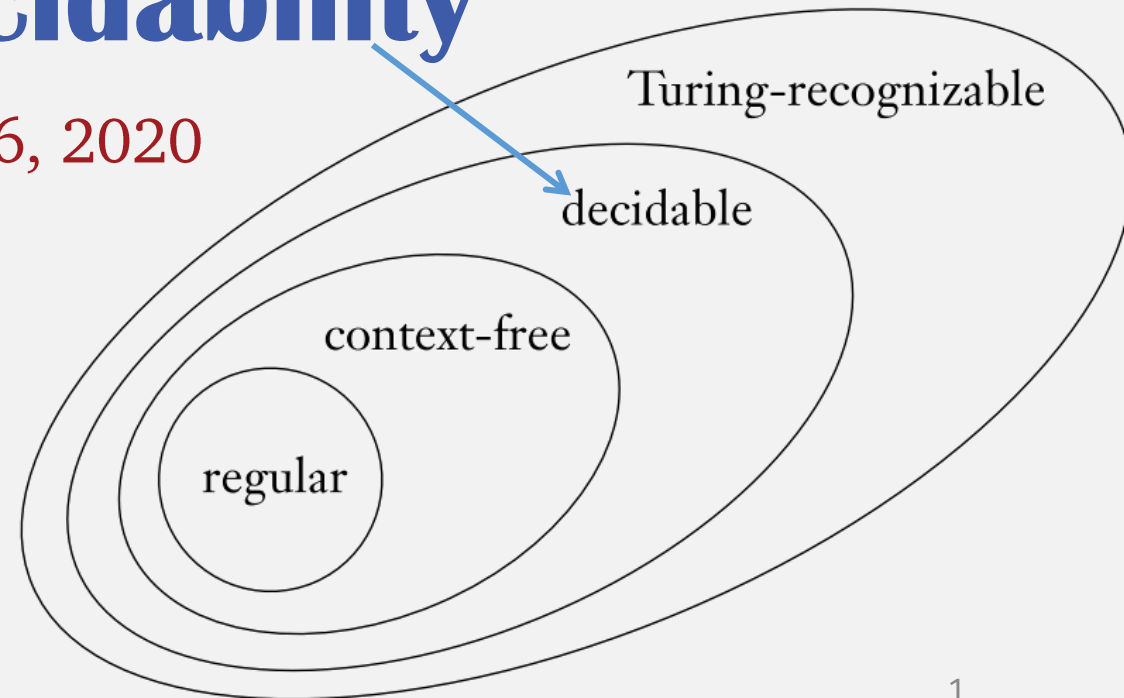


CS420

Chapter 4: Decidability

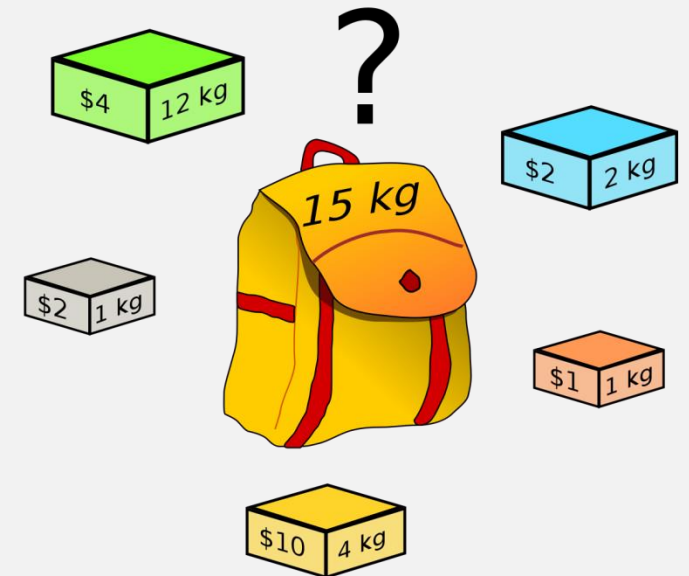
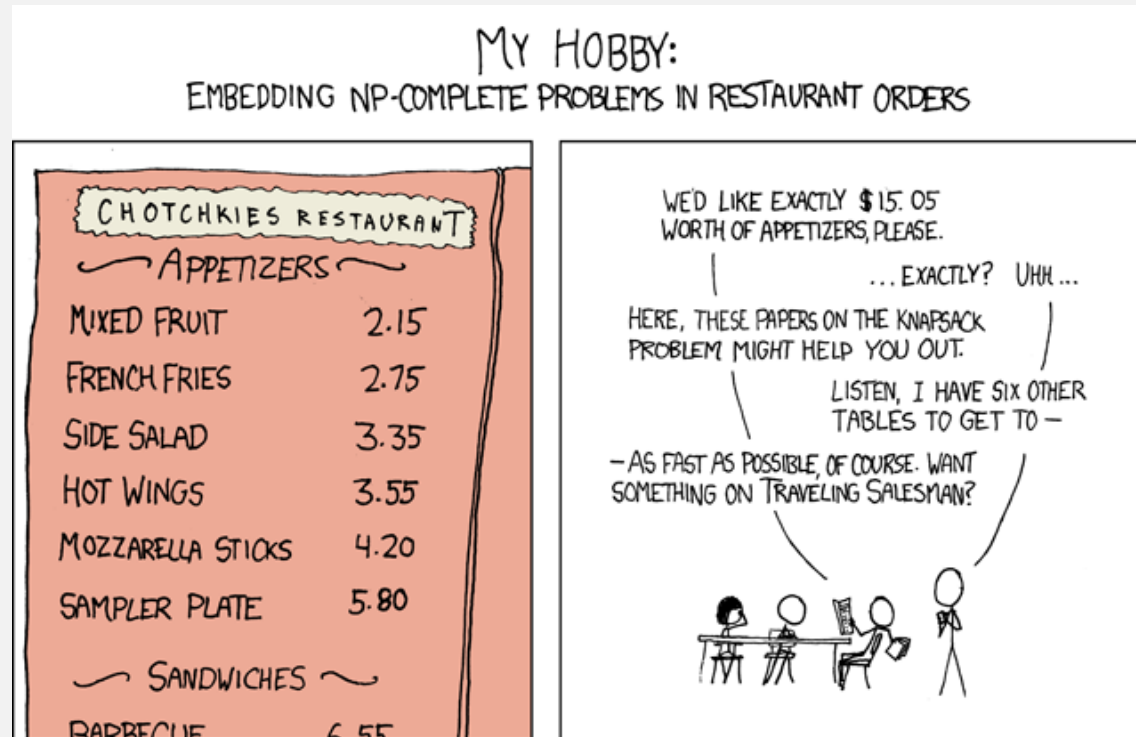
Mon, October 26, 2020



HW5 Questions?

Recap: Turing Machines “in Real Life”

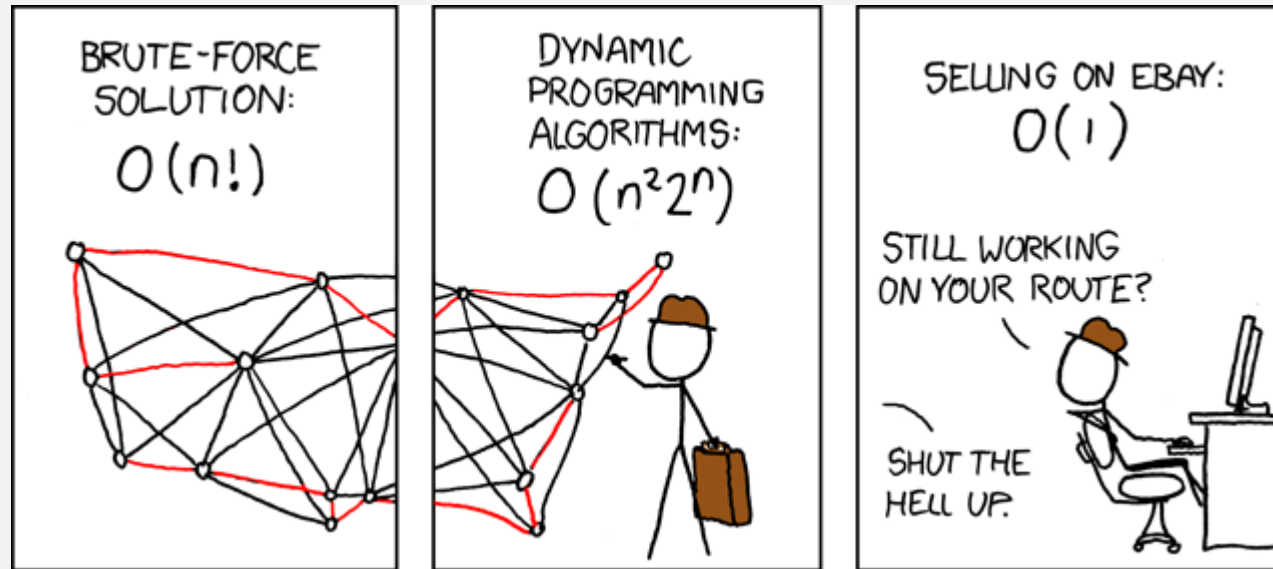
- Optimal ordering in restaurants, i.e., the Knapsack Problem



- (or really any optimization problem)
- NP-complete =
 - set of problems that a nondeterministic TM can decide in polynomial time

Recap: Turing Machines “in Real Life”

- Route planning

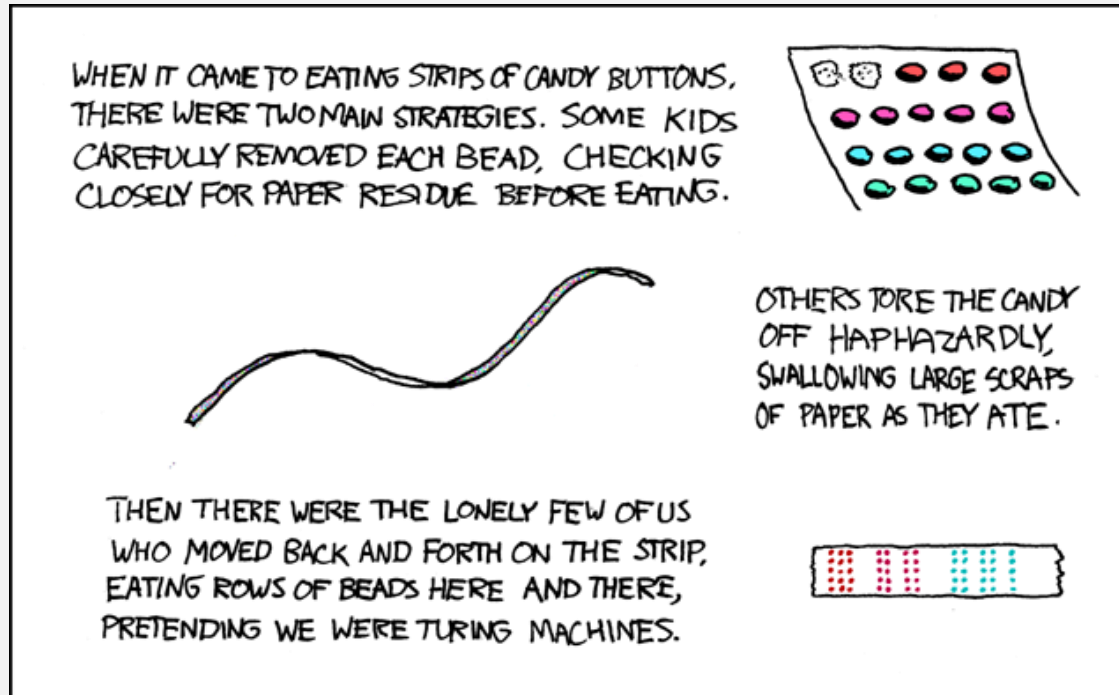


- And scheduling too
- Actually any kind of problem solving:
 - E.g., doing hw, playing board games, cooking
- TMs = computation = programs!

Remember this!

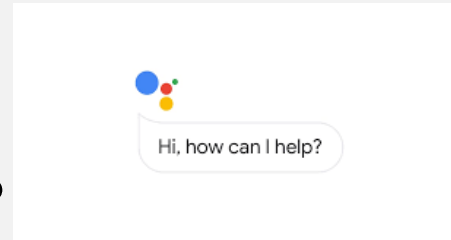
A Turing Machine in Real Life, Literally

- Eating candy???

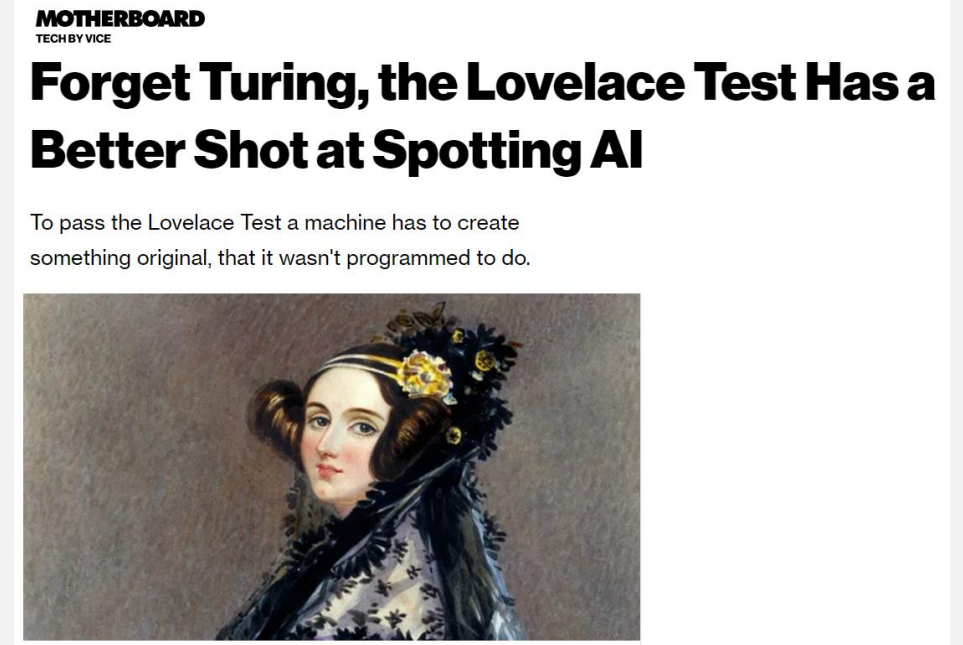


Turing Machines, Philosophically

- TMs = computation
- Your brain = a computer, so ...
- TMs = thought???
- Turing Test: chatbot or human?

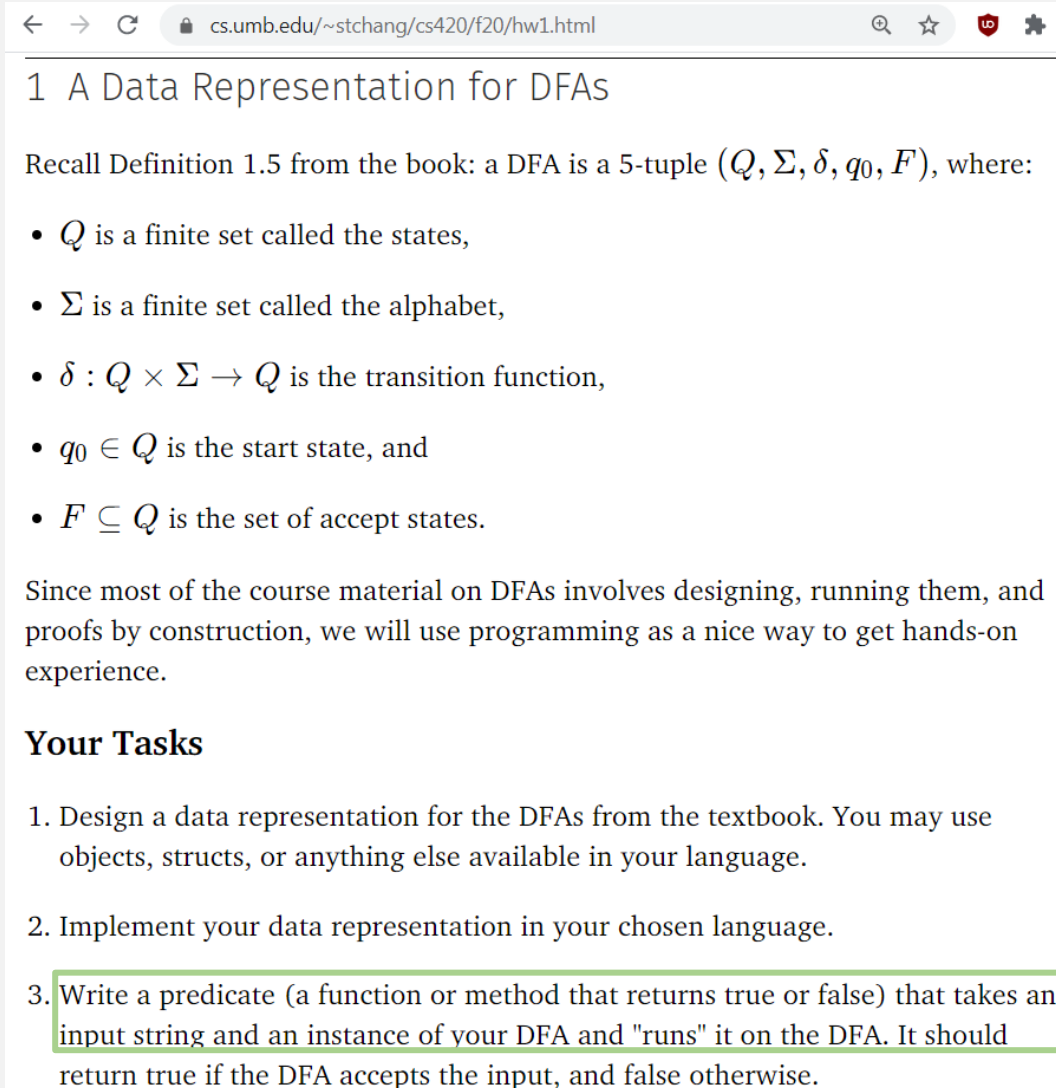


- Lovelace Test: Can TMs create Art???



Decidable Problems about Regular Languages

Flashback: HW1, Problem 1: The “run” fn



← → ↻ cs.umb.edu/~stchang/cs420/f20/hw1.html 🔍 ☆ 🛡️ ⚙️

1 A Data Representation for DFAs

Recall Definition 1.5 from the book: a DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

Since most of the course material on DFAs involves designing, running them, and proofs by construction, we will use programming as a nice way to get hands-on experience.

Your Tasks

1. Design a data representation for the DFAs from the textbook. You may use objects, structs, or anything else available in your language.
2. Implement your data representation in your chosen language.
3. Write a predicate (a function or method that returns true or false) that takes an input string and an instance of your DFA and "runs" it on the DFA. It should return true if the DFA accepts the input, and false otherwise.

What “machine” implements the “run” fn?

- Is the “run” function a regular language?
- Is the “run” function context-free language?
- What does it mean to say the “run” function is a language???

The language of the “run” function

$$A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \}$$

Interlude: Encoding Things into Strings

- A Turing machine's input is always a string
- So anything we want to give to TM must be **encoded** as string
- Notation: $\langle \text{Obj} \rangle$ = encoding for object Obj as string
 - E.g., Obj = a DFA
 - Can you think of a string “encoding” for DFAs???? (hint: hw1, again)
- Combine multiple Objs via tuple, e.g., $\langle B, w \rangle$ from prev slide

Interlude: Informal TMs and Encodings

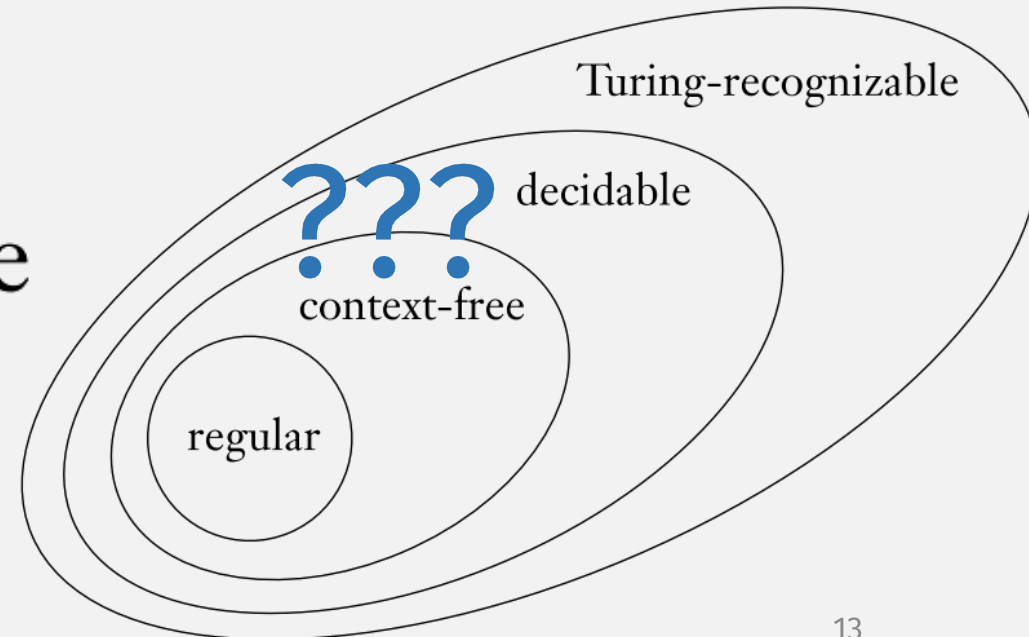
- An informal TM description:
 - Won't always describe exactly how input is encoded
 - Implicitly checks that input is a “valid” encoding

The language of the “run” function

$$A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \}$$

- Want to prove:

A_{DFA} is a decidable language



Thm: A_{DFA} is a decidable language

$$A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \}$$

- Create a Turing machine that decides A_{DFA}
 - (A decider is TM that, for all inputs, always halts and accepts/rejects)

$M =$ “On input $\langle B, w \rangle$, where B is a DFA and w is a string:

1. Simulate B on input w .
2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*.”

- “Simulate” =
 - Start in the starting state “ q_0 ”
 - Iterating over input one char at a time
 - Call delta transition fn to compute the “next state”
- You all already “proved” this in hw1!

Remember:
TMs = programs

Thm: A_{NFA} is a decidable language

$$A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}$$

- Create a Turing machine that decides A_{NFA} :

$N =$ “On input $\langle B, w \rangle$, where B is an NFA and w is a string:

1. Convert NFA B to an equivalent DFA C , using the procedure for this conversion given in Theorem 1.39.
2. Run TM M on input $\langle C, w \rangle$.
3. If M accepts, *accept*; otherwise, *reject*.”

- You all already “proved” this one too!
 - Previous (constructively) proved theorems are a “library” of TMs

Remember:
**Constructive Proofs =
libraries of TMs = programs**

Thm: A_{REX} is a decidable language

$$A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ is a regular expression that generates string } w \}$$

- Create a Turing machine:

$P =$ “On input $\langle R, w \rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A by using the procedure for this conversion given in Theorem 1.54.
2. Run TM N on input $\langle A, w \rangle$.
3. If N accepts, *accept*; if N rejects, *reject*.”

- Yet another theorem you already “proved”!

DFA TMs Recap (So Far)

- $A_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$
 - Deciding TM = program = HW1 “run” function
- $A_{\text{NFA}} = \{\langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w\}$
 - Deciding TM = program = NFA->DFA + “run”
- $A_{\text{REGEX}} = \{\langle R, w \rangle \mid R \text{ is a regular expression that generates string } w\}$
 - Deciding TM = program = Regexp->NFA + NFA->DFA + “run”

Remember:
TMs = programs

Turing machines = Programs

- **Creating** a TM = Programming
- E.g., if HW asks “Show that lang L is decidable” ...
 - .. you must create a TM that decides L; to do this ...
 - ... think of how to write a (halting) program that does what you want
- Hint: When creating a TM, use previously proved theorems
 - I.e., just like you use libraries when programming!
 - E.g., “Library” for DFAs:
 - NFA→DFA, Regexp→NFA, union, intersect, star, homomorphism, backwards,
 $A_{DFA}, A_{NFA}, A_{REG}, \dots$

Thm: E_{DFA} is a decidable language

$$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

- Create a Turing machine:

$T =$ “On input $\langle A \rangle$, where A is a DFA:

1. Mark the start state of A .
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, *accept*; otherwise, *reject*.”

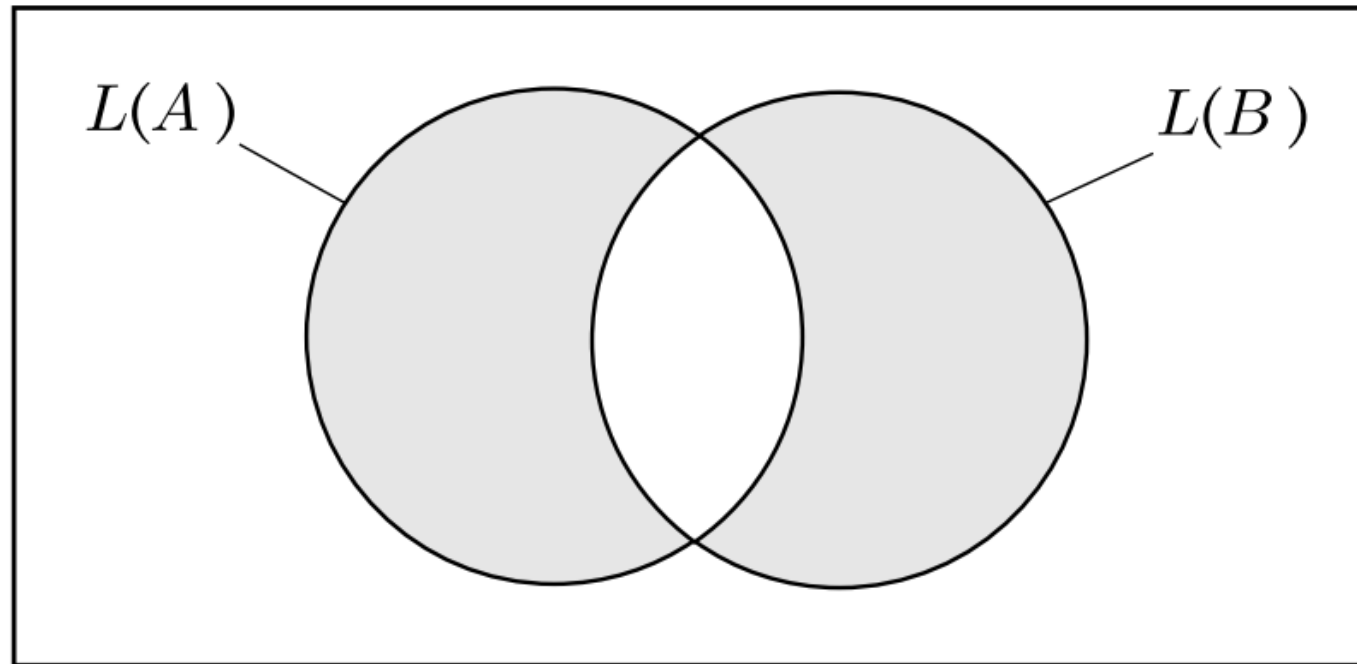
- I.e., check if accept states are “reachable” from start state

Thm: EQ_{DFA} is a decidable language

$$EQ_{\text{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

- Trick: Use Symmetric Difference

Symmetric Difference



$$L(C) = \left(L(A) \cap \overline{L(B)} \right) \cup \left(\overline{L(A)} \cap L(B) \right)$$

$$L(C) = \emptyset \text{ iff } L(A) = L(B)$$

Thm: EQ_{DFA} is a decidable language

$$EQ_{\text{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

- Use Symmetric Difference $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$
 $L(C) = \emptyset$ iff $L(A) = L(B)$

- Construct C = Union and intersection of machines A and B
- Use “library” TM for: $E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

$F =$ “On input $\langle A, B \rangle$, where A and B are DFAs:

1. Construct DFA C as described.
2. Run TM T deciding E_{DFA} on input $\langle C \rangle$.
3. If T accepts, *accept*. If T rejects, *reject*.”

Check-in Quiz 10/26

On gradescope

End of Class Survey 10/26

See course website