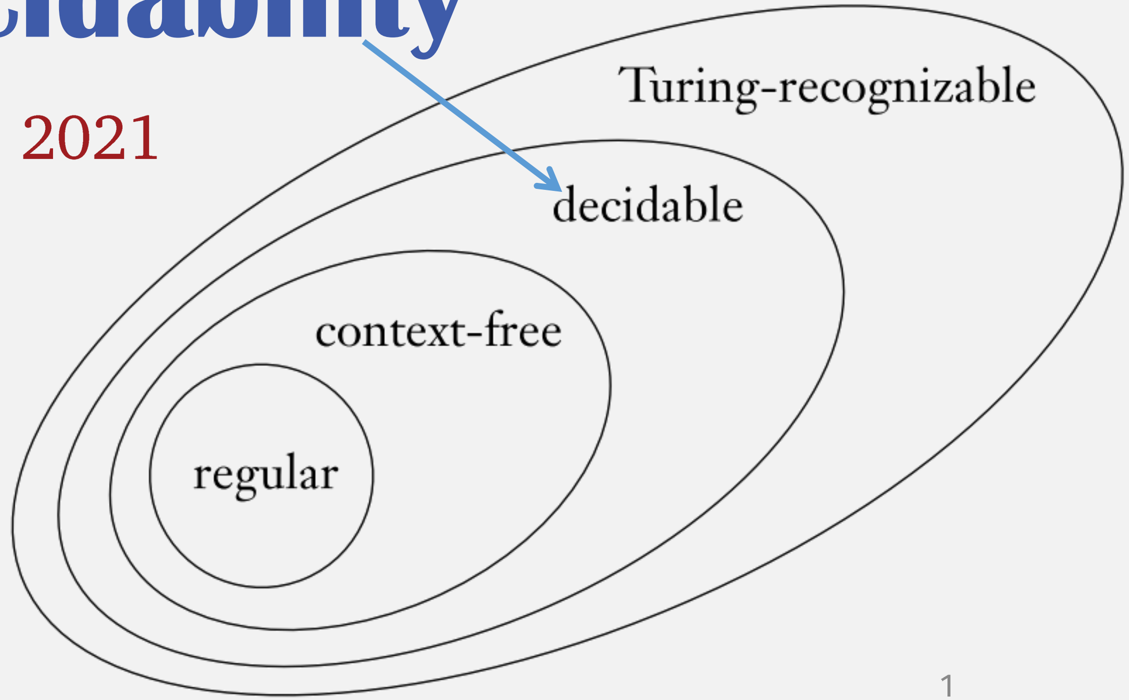
CS420
Chapter 4: Decidability

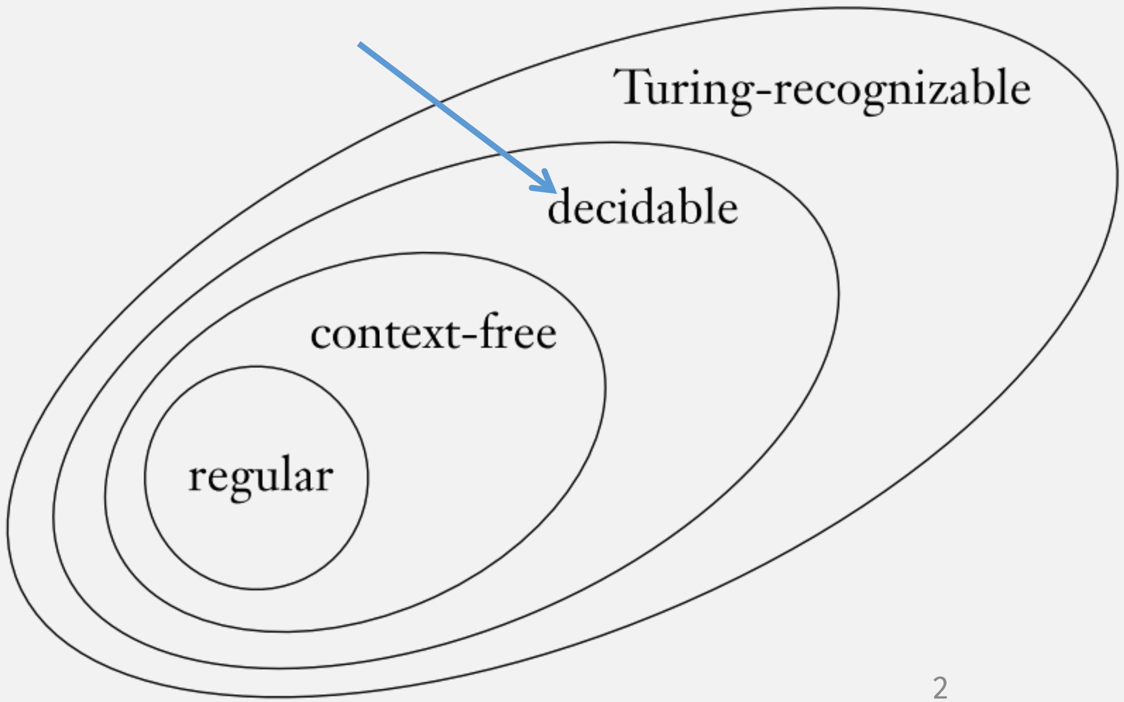
Wed March 24, 2021



Announcements

- HW 6 due Sun 3/28 11:59pm EST

- HW 7 due Sun 4/4 11:59pm EST
  - Covers Ch 4 material (starting today)



Turing Machines and Algorithms

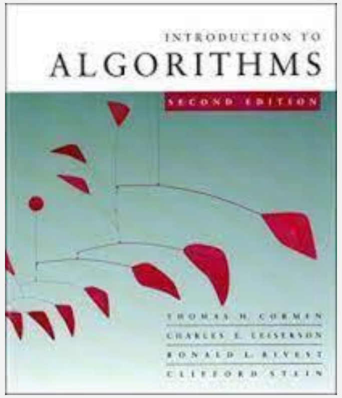
- Turing Machines can express any "computation"
  - I.e., **a Turing Machine is just a** (Python, Java, Racket, …) **program**!

- 2 classes of Turing Machines
  - Recognizers may loop forever
  - Deciders always halt

- Algorithms are an important class of programs
  - In this class, an algorithm is any program that always halts

- So deciders model algorithms!

Remember:
TMs = programs



Algorithms (i.e., Decidable Problems) about Regular Languages

Flashback: HW2, Problem 1: The "run" fn

cs.umb.edu/~stchang/cs420/s21/hw2.html

1 Simulating Computation for DFAs

Recall the formal definition of computation from page 40 of the textbook:

A finite automata $M = (Q, \Sigma, \delta, q_0, F)$ *accepts* a string $w = w_1, \ldots, w_n$, where each character $w_i \in \Sigma$, if there exists a sequence of states $r_0, \ldots, r_n$, where $r_i \in Q$, and:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$
3. $r_n \in F$

This problem asks you to demonstrate, with code, that you understand this concept.

Your Tasks

1. Write a "run" predicate (a function or method that returns true or false) that takes two arguments, an instance of your DFA representation (as defined in A Data Representation for DFAs) and a string, and "runs" the string on the DFA.

The “run” algorithm as a Turing Machine

- HW2’s “run” function is a Turing Machine.
  - Remember: (Python) **programs = Turing Machines**

- What is the language recognized by this Turing Machine?
  - I.e., what are the inputs?

Flashback: HW2, Problem 1: The “run” fn

cs.umb.edu/~stchang/cs420/s21/hw2.html

1 Simulating Computation for DFAs

Recall the formal definition of computation from page 40 of the textbook:

A finite automata $M = (Q, \Sigma, \delta, q_0, F)$ *accepts* a string $w = w_1, \ldots, w_n$, where each character $w_i \in \Sigma$, if there exists a sequence of states $r_0, \ldots, r_n$, where $r_i \in Q$, and:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$
3. $r_n \in F$

This problem asks you to demonstrate, with code, that you understand this concept.

Your Tasks

1. Write a "run" predicate (a function or method that returns true or false) that takes two arguments, an instance of your DFA representation (as defined in A Data Representation for DFAs) and a string, and "runs" the string on the DFA.

The language of the “run” function

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

<u>Interlude</u>: 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
- <u>Notation</u>: <Something> = encoding for Something, as a string
  - E.g., Something might be a DFA
  - Can you think of a string “encoding” for DFAs????
    - Used in HW1, HW2, …
- Use a tuple to combine multiple encodings, e.g., <B,w> (from prev slide)

Interlude: Informal TMs and Encodings

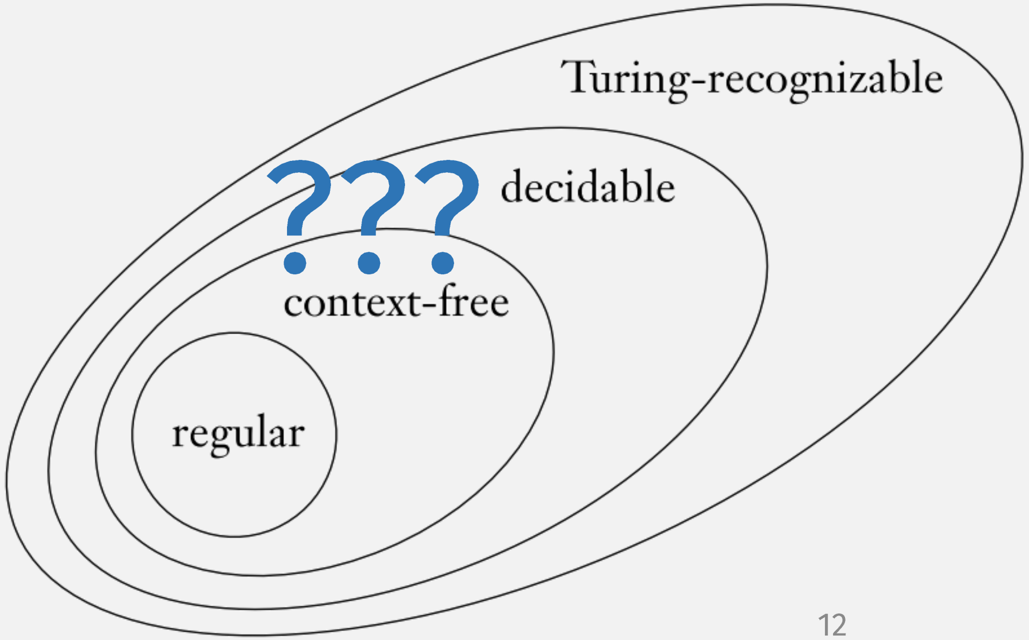
- An informal TM description:
  - Doesn't need to describe exactly how input string is encoded
  - Assumes input is a "valid" encoding
    - Invalid encodings are automatically rejected

The language of the "run" function

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

- "run" program is a Turing machine
- But is it a decider or recognizer?
  - I.e., is it an algorithm?
- To show it's an algo, need to prove:

$$A_{\mathsf{DFA}} \text{ is a decidable language}$$



How to prove that a language is decidable?

- Create a Turing machine that decides that language!

Remember:

- A decider is Turing Machine that always halts, and, for any input, either accepts or rejects it.

How to Design Deciders

- If TMs = Programs …
- … then **Creating** a TM = Programm**ing**

- 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

Thm: A_{DFA} is a decidable language

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

Decider for A_{DFA} :

$M =$ "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*."

Where "Simulate" =

- Start in the starting state "q0" …
- For each input char *x* …
  - Call delta fn with current state and *x* to compute "next state"

Remember:
TMs = programs
Creating TM = programming

- This is a decider (i.e., it always halts) because the input is always finite
- **This is just the answer to HW2's "run" function!**
  - I.e., you already "proved" this!

Thm: A_{NFA} is a decidable language

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

Decider for 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 $\langle C, w\rangle$. (from prev slide)
3. If M accepts, *accept*; otherwise, *reject*."

Remember:
TMs = programs
Creating TM = programming
Previous theorems = library

This is a decider (i.e., it always halts) because:

- Step 1 always halts bc there's a finite number of states in an NFA
- Step 2 always halts because M is a decider

How to Design Deciders, Part 2

- If TMs = Programs …
- … then **Creating** a TM = Programm**ing**

- 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:

- Previous (constructive) theorems are a "library" of reusable TMs
- When creating a TM, try to use these theorems to help you
  - Just like you use libraries when programming!
- E.g., "Library" for DFAs:
  - NFA->DFA, Regexp->NFA,
  - union, intersect, star, homomorphism, FLIP,
  - A_{DFA}, A_{NFA}, A_{REX}, …

Thm: A_{REX} is a decidable language

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

Decider:

$P =$ "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*."

This is a decider because:

- Step 1 always halts because converting reg expr to NFA is done recursively, where the reg expr gets smaller at each step, eventually reaching the base case
- Step 2 always halts because N is a decider

DFA TMs Recap (So Far)

Remember:
TMs = programs
Creating TM = programming
Previous theorems = library

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
  - Deciding TM = program = HW2 “run” function
- $A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}$
  - Deciding TM = program = HW3 NFA->DFA + DFA “run”
- $A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$
  - Deciding TM = program = HW4 Regexp->NFA + NFA->DFA + DFA “run”

Thm: E_{DFA} is a decidable language

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

Decider:

$T =$ "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.

I.e., this is a "reachability" algorithm

we check if accept states are "reachable" from start state

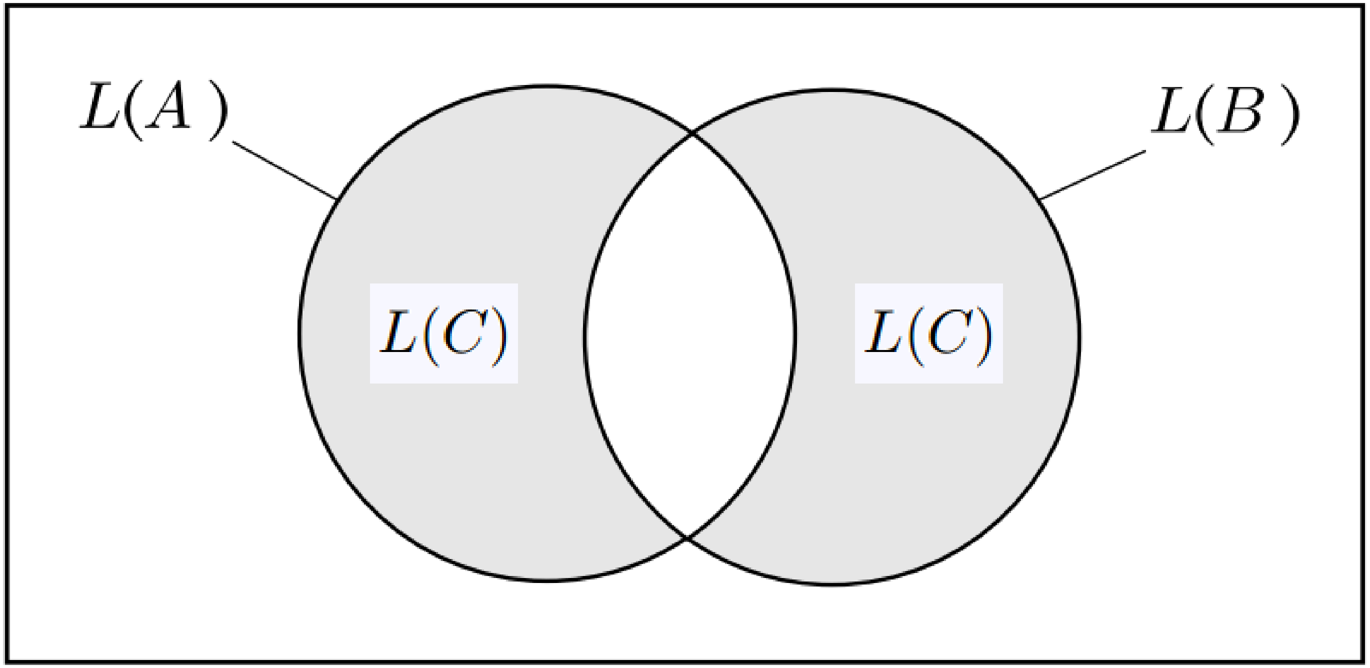
Thm: EQ_{DFA} is a decidable language

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

Trick: Use Symmetric Difference

Symmetric Difference

Bonus Pts:
prove negation,
i.e., set complement,
is closed for regular
languages



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

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

Thm: EQ_{DFA} is a decidable language

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

Construct decider using 2 ingredients:

- Symmetric Difference algo: $L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$
  - Construct C = Union, intersection, negation of machines A and B
- decider (from "library") for: $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
  - Because $L(C) = \emptyset$ iff $L(A) = L(B)$

$F =$ "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*."

Summary: Decidable DFA Langs (i.e., algorithms)

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

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

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

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

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

Remember:
TMs = programs
Creating TM = programming
Previous theorems = library

Next time:

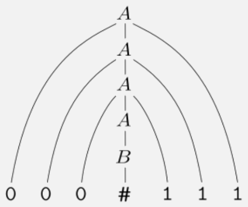
Decidable Problems (i.e., Algorithms)
about Context-Free Languages (CFLs)

Next time : A_{CFG} is a decidable language

$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

- This a is very practically important problem …
- … equivalent to:
  - Is there an **algorithm to parse programming lang** with grammar G?

- A Decider for this problem could … ?
  - Try all possible derivations of G?
  - But this might never halt
    - e.g., if there is a rule like: S -> $0S$ or S -> S
  - This TM would be a recognizer but not a decider



- Idea: can the TM stop checking after some length?
  - i.e., Is there upper bound on the number of derivation steps?

Check-in Quiz 3/24

On gradescope