

CS103  
WINTER 2026



Lecture 21:

# Turing Machines

Part 2 of 3

# Outline for Today

- ***The Church-Turing Thesis***
  - Just how powerful are TMs?
- ***What Does it Mean to Solve a Problem?***
  - It's more subtle than it looks.
- ***Recognizers and Deciders***
  - Two modes of problem-solving.

Recap from Last Time

# Turing Machines

- A ***Turing machine*** is a program that controls a tape head as it moves around an infinite tape.
- There are six commands:
  - **Move** *direction*
  - **Write** *symbol*
  - **Goto** *label*
  - **Return** *boolean*
  - **If** *symbol command*
  - **If Not** *symbol command*
- Despite their limited vocabulary, TMs are surprisingly powerful.

# A Sample Turing Machine

- Here's a sample TM.
- It receives inputs over the alphabet  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ .
- What strings does this TM accept?
- Can you write a regex that matches precisely the strings this TM accepts?

```
Start:  
    If Not 'a' Return False  
  
Loop:  
    Move Right  
    If Not Blank Goto Loop  
    Move Left  
    Move Left  
    If Not 'b' Return False  
    Return True
```

Answer at

<https://cs103.stanford.edu/polle>

# What Can We Do With a TM?

- Last time, we saw TMs that
  - check if a string has the form  $a^n b^n$ ,
  - check if a string has the same number of **a**'s and **b**'s, and
  - sort a string of **a**'s and **b**'s.
- Here's a list of some other things TMs can do; these TMs are included with the starter files for PS8.
  - Check if a number is a Fibonacci number.
  - Convert the number  $n$  into a string of  $n$  **a**'s.
  - Check if a string is a *tautonym* (the same string repeated twice).
  - So much more!
- **Fun exercise:** Build a faster sorting algorithm for TMs!
- This hints at the idea that TMs might be more powerful than they look.

New Stuff!

## ***Main Questions for Today:***

Just how powerful are Turing machines?

What problems can you solve with a computer?

# Real and “Ideal” Computers

- A real computer has finite memory: finite disk space, finite RAM, etc.
- But as computers get more powerful, the amount of memory available keeps increasing.
  - Compare our first PCs to your laptops!
- An ***idealized computer*** is a computer with unlimited RAM and disk space.
- It functions just like a regular computer, but never runs out of memory.

# Just How Powerful Are TMs?

- **Theorem:** Turing machines are equal in power to idealized computers.
- More specifically: any computation that can be done on a TM can be done on an idealized computer.
  - An idealized computer can solve problems that in *principle* we could solve today, but for which we simply don't have enough memory.
  - Side note: Your laptop could be faithfully simulated with a DFA!
- **Key idea:** Two models of computation are equally powerful if they can simulate each other.
  - Example: DFAs and NFAs!

# Simulating a TM

- ***Tape and Tape Head:*** The memory for a TM can be thought of as a string with some number keeping track of the current index.
- ***Commands:*** The individual commands in a TM are simple and perform only basic operations:
  - Move    Write    Goto    Return    If
- To simulate a TM, we need to
  - see which line of the program we're on,
  - determine what command it is, and
  - simulate that single command.
- ***Claim:*** This is reasonably straightforward to do on an idealized computer.
  - The “core” logic for our TM simulator is under fifty lines of code, including comments.

# Simulating a TM

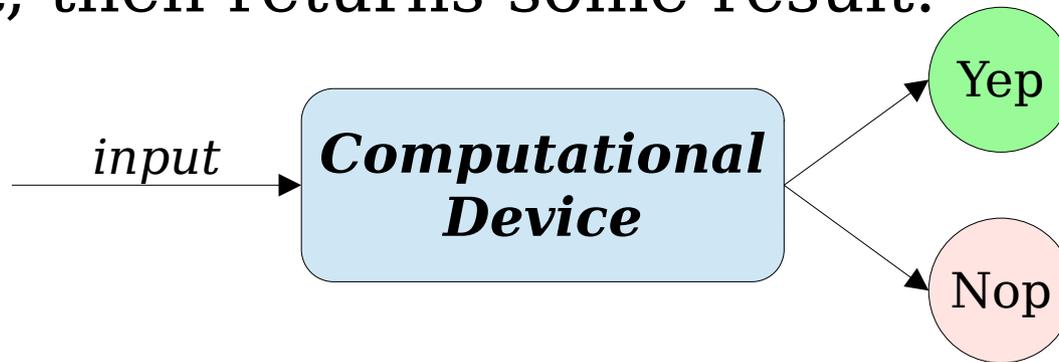
- Because a computer can simulate each individual TM instruction, an idealized computer can do anything a TM can do.
- ***Key Ideas:***
  - Even the most complicated TM is made out of individual instructions.
  - If we can simulate those instructions, we can simulate an arbitrarily complicated TM.

# Simulating a Computer

- Programming languages provide a set of simple constructs.
- You, the programmer, then combine these basic constructs together to assemble larger programs.
  - If statements
  - Loops
  - While loops
  - Functions
  - Variables
  - Linked structures and pointers
  - Object-oriented programming
  - Functional call stacks
  - Another approach: assembly!  
add, compare, jump
- **Key Idea:** A TM is powerful enough to simulate each of these individual pieces. It's therefore powerful enough to simulate anything a real computer can do.

# A Leap of Faith

- **Claim:** A TM is powerful enough to simulate any computer program that gets an input, processes that input, then returns some result.



- The resulting TM might be colossal, or really slow, or both, but it would still faithfully simulate the computer.
- We're going to take this as an article of faith in CS103. If you curious for more details, come talk to me after class.

# Can a TM Work With...

“pictures of adorable creatures?”

Sure! A picture is just a 2D array of colors, and a color can be represented as a series of numbers.



# Can a TM Work With...

~~“pictures?”~~  
“videos?”

If you think about it, a video is just a series of pictures!



# Can a TM Work With...

“music?”

Sure! Music is encoded as a compressed waveform. That's just a list of numbers.

“Generative AI?”

Sure! That's just applying a bunch of matrices and nonlinear functions to some input.

# Just how powerful *are* Turing machines?

- Doesn't it feel a bit too coincidental that they're a model of idealized computers?
  - Turing's *On Computable Numbers* was written in 1936. ENIAC was engineered between 1943 and 1945.
- A comparison: DFAs, NFAs, and regular expressions are equivalent in power but look *very different* from one another.

The ***Church-Turing Thesis*** claims that  
***every feasible method of computation  
is either equivalent to or weaker than  
a Turing machine.***

“This is not a theorem – it is a  
falsifiable scientific hypothesis.  
And it has been thoroughly  
tested!”

- Ryan Williams

Regular Languages

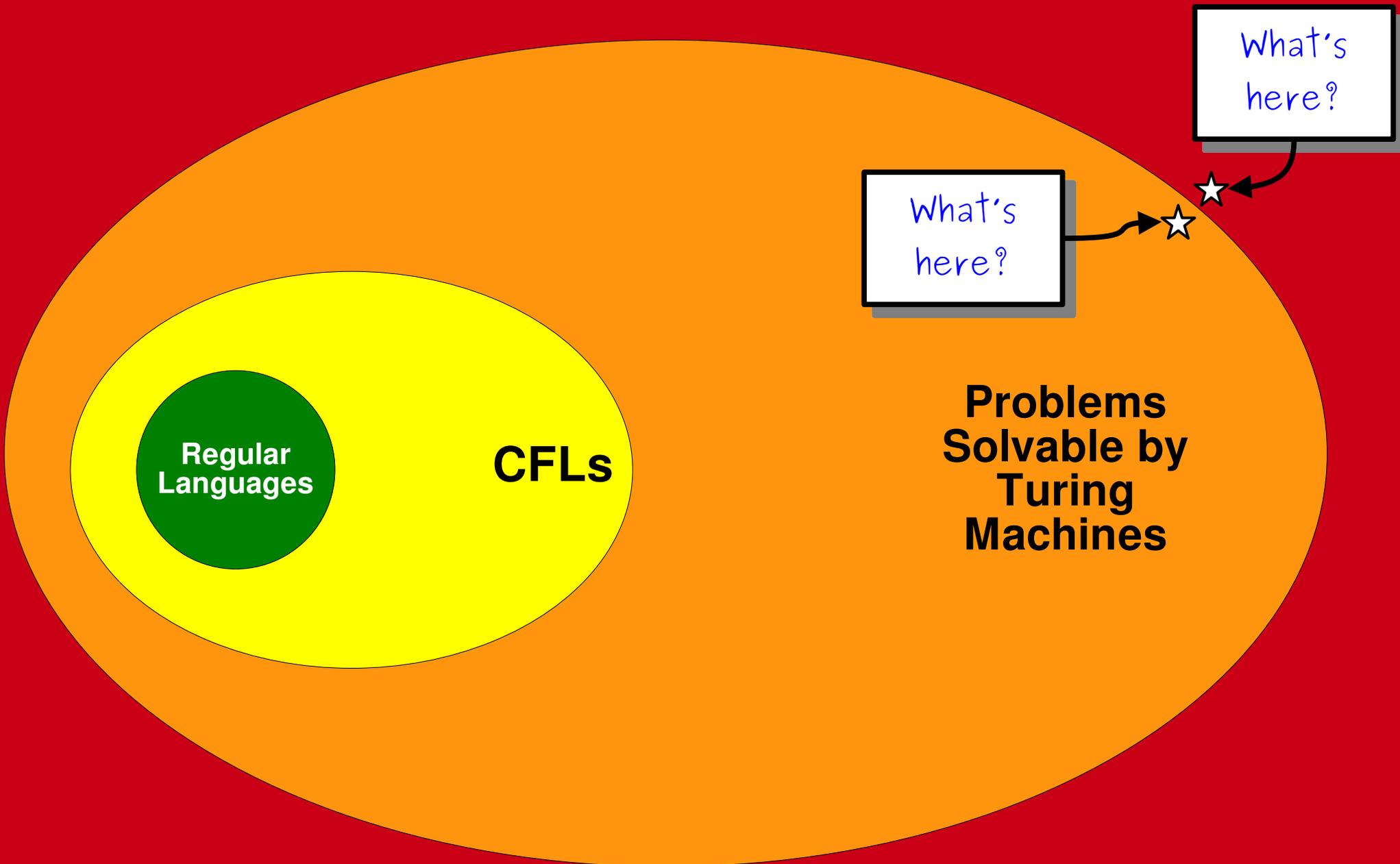
CFLs

Problems Solvable by Turing Machines

All Languages

What's here?

What's here?



**Time-Out for Announcements!**

# Problem Set 8

- Problem Set Seven was due today at 1:00 PM.
  - You can use a late day to extend the deadline to 1:00 PM on Saturday.
- Problem Set Eight is posted. It's due next Friday at 1:00 PM.
  - Construct context-free grammars and explore their expressive power.
  - Dive deeper into the structure of languages and functions between languages.
  - Tinker with TMs and what it's like to build all computation from smaller pieces.
- You know the drill: come talk to us if you have any questions, and let us know what we can do to help out.

# Reminder

- The withdrawal deadline is today at 5:00 PM.
- This is the change of grading basis deadline, as well.
  - Be sure to check out the grade cruncher before making any final decisions.

Back to CS103!

# Decidability and Recognizability

What problems can we solve with a computer?

What kind of  
computer?



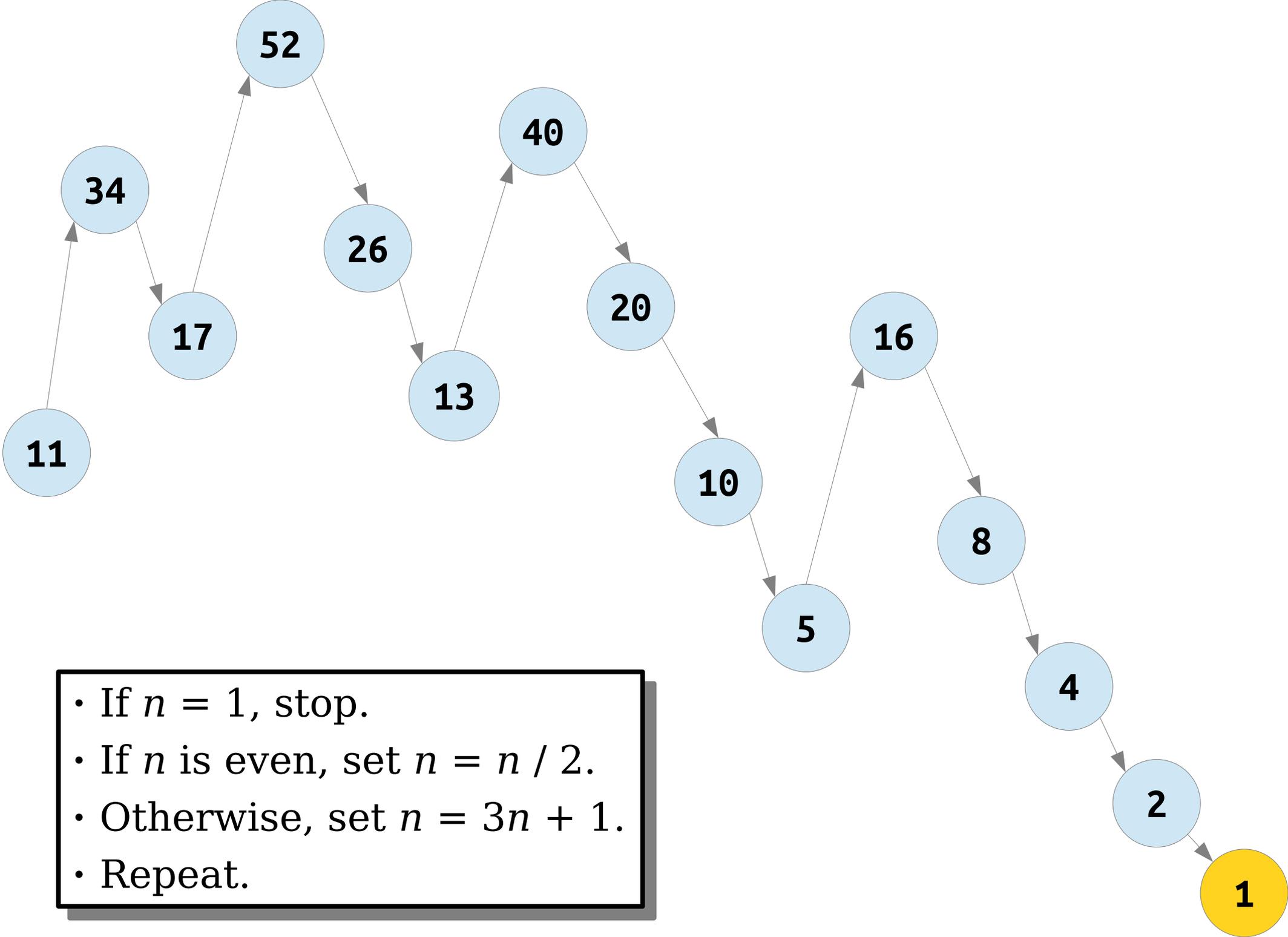
What problems can we solve with a computer?

What does it  
mean to "solve"  
a problem?



# The Hailstone Sequence

- Consider the following procedure, starting with some  $n \in \mathbb{N}$ , where  $n > 0$ :
  - If  $n = 1$ , you are done.
  - If  $n$  is even, set  $n = n / 2$ .
  - Otherwise, set  $n = 3n + 1$ .
  - Repeat.
- **Question:** Given a natural number  $n > 0$ , does this process terminate?



- If  $n = 1$ , stop.
- If  $n$  is even, set  $n = n / 2$ .
- Otherwise, set  $n = 3n + 1$ .
- Repeat.

# The Hailstone Sequence

- Consider the following procedure, starting with some  $n \in \mathbb{N}$ , where  $n > 0$ :
  - If  $n = 1$ , you are done.
  - If  $n$  is even, set  $n = n / 2$ .
  - Otherwise, set  $n = 3n + 1$ .
  - Repeat.
- Does the Hailstone Sequence terminate for...
  - $n = 5$ ?
  - $n = 20$ ?
  - $n = 7$ ?
  - $n = 27$ ?

Answer at  
<https://cs103.stanford.edu/pollev>

# The Hailstone Turing Machine

- Let  $\Sigma = \{\mathbf{a}\}$  and consider the language  $L = \{ \mathbf{a}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}$ .

- We can build a TM for  $L$  as follows:

If the input is  $\varepsilon$ , reject.

While the string is not  $\mathbf{a}$ :

If the input has even length, halve the length of the string.

If the input has odd length, triple the length of the string and append a  $\mathbf{a}$ .

Accept.

Does this Turing machine always eventually stop running?

# The Collatz Conjecture

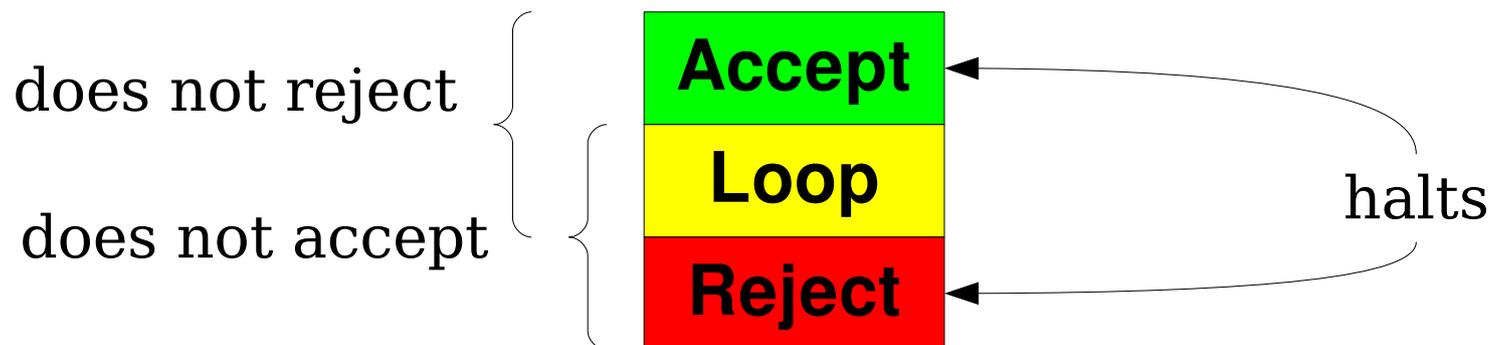
- It is *unknown* whether this process will terminate for all natural numbers.
  - In other words, no one knows whether this TM always terminates!
- The conjecture (unproven claim) that the hailstone sequence always terminates is called the ***Collatz Conjecture***.
- This problem has eluded a solution for a long time. The influential mathematician Paul Erdős is reported to have said “mathematics may not be ready for such problems.”

# An Important Observation

- Unlike finite automata, which automatically halt after all the input is read, TMs keep running until they explicitly return true or return false.
- As a result, it's possible for a TM to run forever without accepting or rejecting.
- What does “solving” a problem with a TM mean when a TM might run forever without giving an answer?

# Very Important Terminology

- Let  $M$  be a Turing machine.
- $M$  **accepts** a string  $w$  if it returns true on  $w$ .
- $M$  **rejects** a string  $w$  if it returns false on  $w$ .
- $M$  **loops infinitely** (or just **loops**) on a string  $w$  if when run on  $w$  it neither returns true nor returns false.
- $M$  **does not accept  $w$**  if it either rejects  $w$  or loops on  $w$ .
- $M$  **does not reject  $w$**  if it either accepts  $w$  or loops on  $w$ .
- $M$  **halts on  $w$**  if it accepts  $w$  or rejects  $w$ .



# Recognizers and Recognizability

- A TM  $M$  is called a **recognizer** for a language  $L$  over  $\Sigma$  if the following statement is true:

$$\forall w \in \Sigma^*. (w \in L \leftrightarrow M \text{ accepts } w)$$

- A language  $L$  is called **recognizable** if there is a recognizer for it.
- If you are absolutely certain that  $w \in L$ , then running a recognizer for  $L$  on  $w$  will (eventually) confirm this.
  - Eventually,  $M$  will accept  $w$ .
- If you don't know whether  $w \in L$ , running  $M$  on  $w$  may never tell you anything.
  - $M$  might loop on  $w$  - but you can't differentiate between "it'll accept if you wait longer" and "it will never come back with an answer."
- Does this feel like "solving a problem" to you?

# Recognizers and Recognizability

- Our hailstone TM  $M$  is a recognizer for  
 $L = \{ \mathbf{a}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}$ .
- Why?
  - If the sequence terminates starting at  $n$ , then  $M$  accepts  $\mathbf{a}^n$ .
  - If the sequence doesn't terminate, then  $M$  loops forever on  $\mathbf{a}^n$ , never giving an answer.
- What does that mean?
  - If you (somehow) know the sequence terminates for  $n$ , then  $M$  will eventually confirm this.
  - If you don't know, then  $M$  might not tell you anything.

# Recognizers and Recognizability

- Earlier this quarter you explored sums of five cubes. Now, let's talk about sums of three cubes.
- Are there integers  $x$ ,  $y$ , and  $z$  where...
  - $x^3 + y^3 + z^3 = 10$ ?
  - $x^3 + y^3 + z^3 = 11$ ?
  - $x^3 + y^3 + z^3 = 12$ ?
  - $x^3 + y^3 + z^3 = 13$ ?

Answer at  
<https://cs103.stanford.edu/pollev>

# Recognizers and Recognizability

- Surprising fact: until 2019, no one knew whether there were integers  $x$ ,  $y$ , and  $z$  where

$$x^3 + y^3 + z^3 = 33.$$

- A heavily optimized computer search found this answer:

$$x = 8,866,128,975,287,528$$

$$y = -8,778,405,442,862,239$$

$$z = -2,736,111,468,807,040$$

- As of February 2026, no one knows whether there are integers  $x$ ,  $y$ , and  $z$  where

$$x^3 + y^3 + z^3 = 114.$$

# Recognizers and Recognizability

- Consider the language

$$L = \{ a^n \mid \exists x \in \mathbb{Z}. \exists y \in \mathbb{Z}. \exists z \in \mathbb{Z}. x^3 + y^3 + z^3 = n \}$$

- Here's pseudocode for a recognizer to see whether such a triple exists:

```
for max = 0, 1, 2, ...
  for x from -max to +max:
    for y from -max to +max:
      for z from -max to +max:
        if  $x^3 + y^3 + z^3 = n$ : return true
```

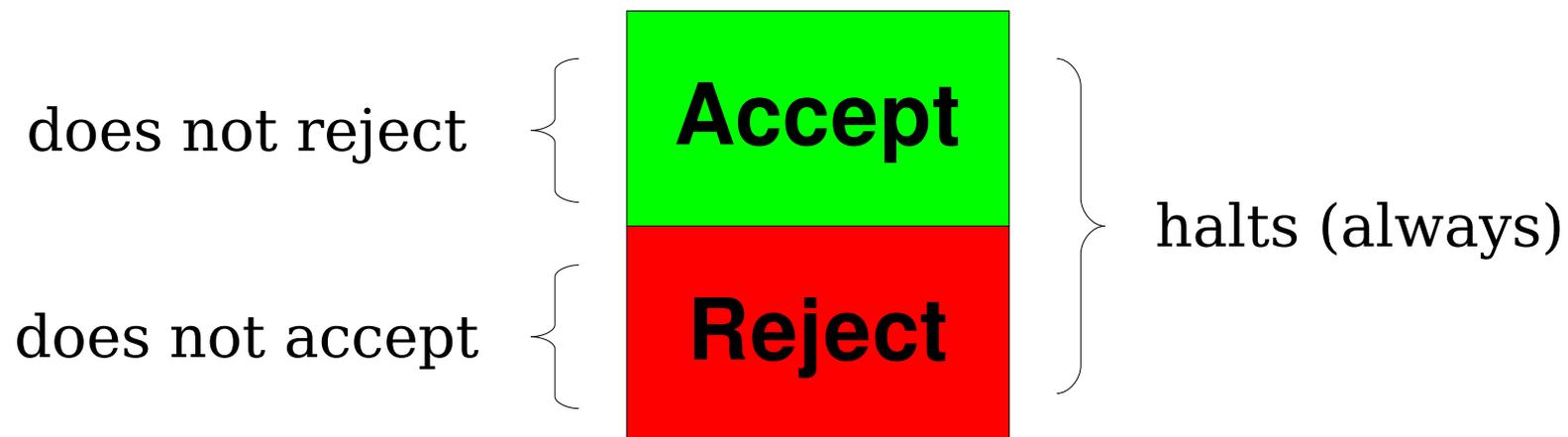
- If you somehow knew there was a triple  $x$ ,  $y$ , and  $z$  where  $x^3 + y^3 + z^3 = n$ , running this program will (eventually) convince you of this.
- If you weren't sure whether a triple exists, this recognizer might not be useful to you.

# Recognizers and Recognizability

- The class **RE** consists of all recognizable languages.
- Formally speaking:  
$$\mathbf{RE} = \{ L \mid L \text{ is a language and there's a recognizer for } L \}$$
- You can think of **RE** as “all problems with yes/no answers where “yes” answers can be confirmed by a computer.”
  - Given a recognizable language  $L$  and a string  $w \in L$ , running a recognizer for  $L$  on  $w$  will eventually confirm  $w \in L$ .
  - The recognizer will never have a “false positive” of saying that a string is in  $L$  when it isn't.
- This is a “weak” notion of solving a problem.
- Is there a “stronger” one?

# Deciders and Decidability

- Some, but not all, TMs have the following property: the TM halts on all inputs.
- If you are given a TM  $M$  that always halts, then for the TM  $M$ , the statement “ $M$  does not accept  $w$ ” means “ $M$  rejects  $w$ .”



# Deciders and Decidability

- A TM  $M$  is a **decider** for a language  $L$  over  $\Sigma$  when

$\forall w \in \Sigma^*. M \text{ halts on } w.$

$\forall w \in \Sigma^*. (w \in L \leftrightarrow M \text{ accepts } w)$

- A language  $L$  is **decidable** when there is a decider for it.
- Equivalently:
  - A decider  $M$  for a language  $L$  accepts all strings in  $L$  and rejects all strings not in  $L$ .
  - A decider  $M$  for a language  $L$  is a recognizer for  $L$  that halts on all inputs.
- Intuitively, if you don't know whether  $w \in L$ , running  $M$  on  $w$  will “create new knowledge” by telling you the answer.
- This is a “strong” notion of “solving a problem.”

# Deciders and Decidability

- The class **R** consists of all decidable languages.
- Formally speaking:  
$$\mathbf{R} = \{ L \mid L \text{ is a language and there's a decider for } L \}$$
- You can think of **R** as “all problems with yes/no answers that can be fully solved by computers.”
  - Given a decidable language, run a decider for  $L$  and see what happens.
  - Think of this as “knowledge creation” – if you don’t know whether a string is in  $L$ , running the decider will, given enough time, tell you.
- The class **R** contains all the regular languages, all the context-free languages, most of CS161, etc.
- This is a “strong” notion of solving a problem.

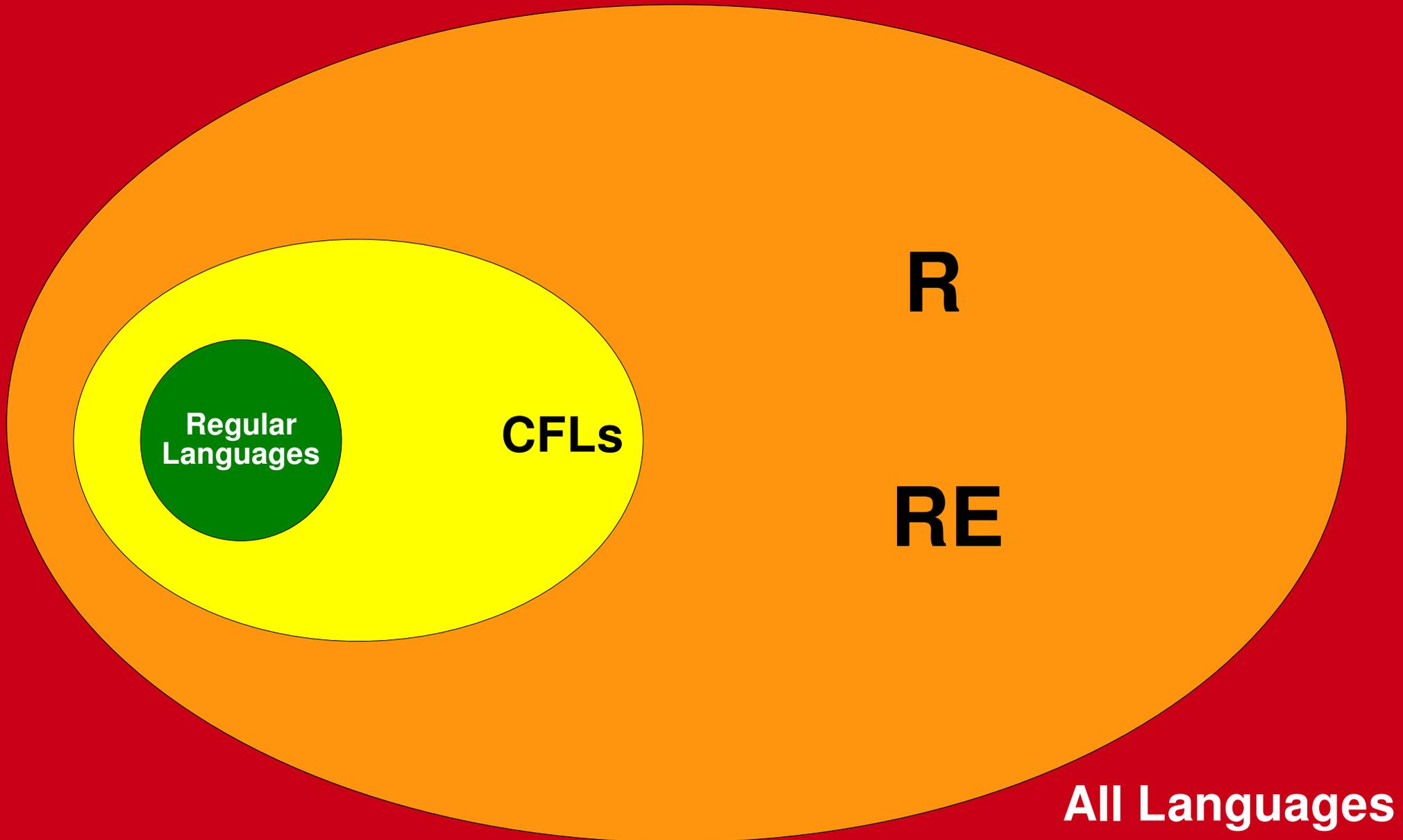
# **R** and **RE** Languages

- Every decider for  $L$  is also a recognizer for  $L$ .
- This means that  $\mathbf{R} \subseteq \mathbf{RE}$ .
- Hugely important theoretical question:

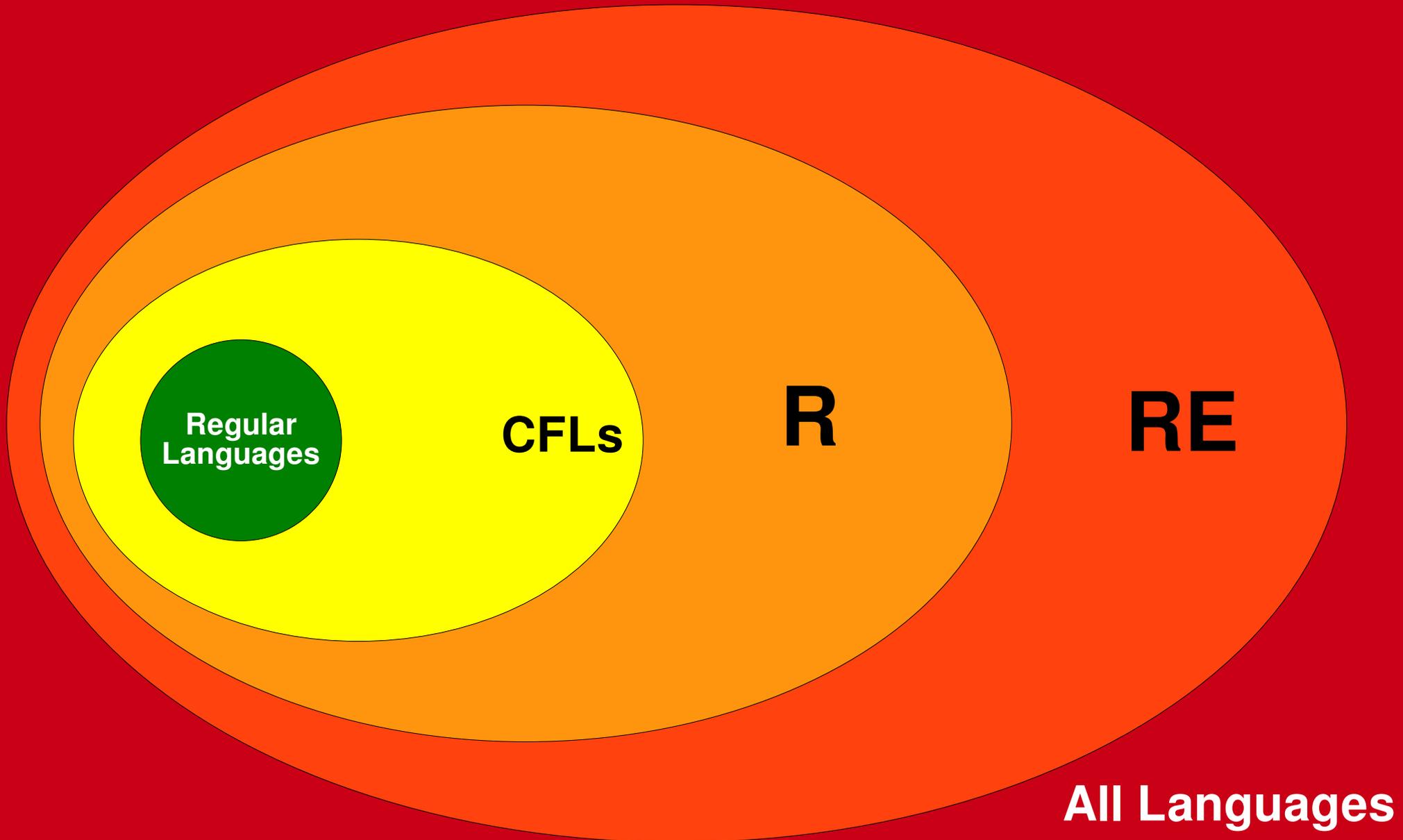
$$\mathbf{R} \stackrel{?}{=} \mathbf{RE}$$

- That is, if you can just confirm “yes” answers to a problem, can you necessarily *solve* that problem?

# Which Picture is Correct?



# Which Picture is Correct?



# Unanswered Questions

- Why exactly is **RE** an interesting class of problems?
- What does the  **$R \stackrel{?}{=} RE$**  question mean?
- Is  **$R = RE$** ?
- What lies beyond **R** and **RE**?
- Find out next week!

# Next Time

- ***Emergent Properties***
  - Larger phenomena made of smaller parts.
- ***Universal Machines***
  - A single, “most powerful” computer.
- ***Self-Reference***
  - Programs that ask questions about themselves.