Unsolvable Problems

Part Four

Outline for Today

- **Recap from Last Time**
  - Wow, that was a lot of stuff. Let's sift through where we are now.
- **Non-RE Languages, Part Two**
  - That was tricky. What exactly are we trying to do here?
- **Verifiers and NTMs**
  - A new framework for solvability.

Recap from Last Time

The Recursion Theorem

- There is a deep result in computability theory called ***Kleene's second recursion theorem*** that, informally, states the following:

  It is possible to construct TMs that perform arbitrary computations on their own descriptions.

- Intuitively, this generalizes our Quine constructions to work with arbitrary TMs.
- Want the formal statement of the theorem? Take CS154!

Revisiting Last Lecture

Self-Reference and **R**

- Here is a general template for using self-reference to prove a language is not in **R**:
  - Assume the language is in **R**.
  - This means that TMs can decide answers to questions of some form. Describe that that question is.
  - Build a TM that gets its own description, then *decides* whether it has a property.
  - Based on the answer it gets, force the TM to have the opposite of the reported property.

Two Intuitions

- The ***avoid your fate*** intuition:
  - Construct a machine so that it learns its fate (*decides* what to do next), then actively chooses to sidestep that fate.
- The ***impossible bind*** intuition:
  - Imagine the TM in conversation with the decider that can allegedly predict what happens next.
  - Have the TM tell the decider that it's going to do the opposite of whatever the decider says.
  - The decider's in an impossible bind – anything it says must be wrong!
- You may find one of these intuitions easier than the other. Hopefully at least one works!

$A_{TM} \notin \mathbf{R}$

- Let's suppose that A_{TM} is decidable.
- This means that TMs can *decide* whether an arbitrary TM will accept an arbitrary string.
- The following TM causes a contradiction:

M = "On input w:
- Have M get its own description, $\langle M \rangle$.
- *Decide* whether M will accept w.
- If M will accept w, choose to reject w.
- If M will not accept w, choose to accept w."

Proving $A_{TM} \notin \mathbf{R}$

- The self-referential nature of this Turing machine can make it hard to write a proof about it.
- ***Recommendation:*** Split the proof into two cases, one for each possible outcome, and trace through the machine in each case.
- In other words, work *forwards* through the Turing machine, not *backwards*.

From experience, this makes these proofs at lot easier to write!

$A_{TM} \notin \mathbf{R}$

- ***Case 1:*** M accepts w.
- If M accepts w, then in step (2), we will continue on to line (3).
- In line (3), we end up rejecting w.
- This contradicts that M accepts w.

M = "On input w:
(1) · Have M get its own description, $\langle M \rangle$.
(2) · *Decide* whether M will accept w.
(3) · If M will accept w, choose to reject w.
(4) · If M will not accept w, choose to accept w."

$A_{TM} \notin \mathbf{R}$

- ***Case 2:*** M does not accept w.
- If M accepts w, then in step (2), we will continue on to line (4).
- In line (4), we end up accepting w.
- This contradicts that M does not accept w.

M = "On input w:
(1) · Have M get its own description, $\langle M \rangle$.
(2) · *Decide* whether M will accept w.
(3) · If M will accept w, choose to reject w.
(4) · If M will not accept w, choose to accept w."

Self-Reference and **RE**

Some Intuitions

- You may find the ***impossible bind*** intuition more useful here.
- Construct a machine that works as follows:
  - The machine says, up front, that if you tell it what to do, it's going to do the opposite.
  - The machine then *also* says that if you don't say anything at all, it's going to treat that as a “no” and that you're going to be wrong.
  - No matter what you tell the machine (and even if you say nothing at all), you're going to be wrong.

Revisiting a Proof

- Assume that $L_D \in$ **RE**.
- What does this machine do?

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept $\langle M \rangle$.
  (This step might loop forever if M does accept $\langle M \rangle$).
- If M will not accept $\langle M \rangle$, choose to accept w."
- If M will accept $\langle M \rangle$, choose to reject w.

Obtaining a Contradiction

- ***Case 1:*** M does not accept $\langle M \rangle$.
- Then in line (2), we will recognize this and continue to line (4).
- In line (4) we accept w, regardless of what w is.
- In particular, this means M accepts $\langle M \rangle$, contradicting that M does not accept $\langle M \rangle$.

M = "On input w:

(1) Have M get its own description, $\langle M \rangle$.

(2) Try to ***recognize*** if M does not accept $\langle M \rangle$.
(This step might loop forever if M does accept $\langle M \rangle$).

(3) If M will not accept $\langle M \rangle$, choose to accept w."

(4) If M will accept $\langle M \rangle$, choose to reject w.

Obtaining a Contradiction

- ***Case 2:*** M accepts $\langle M \rangle$.
- Then in line (2), we have two options – either step (2) loops forever, or step (2) terminates and we go to step (3).
- In either case, M does not accept the string w, regardless of what w is.
- In particular, if $w = \langle M \rangle$, we see that M does not accept $\langle M \rangle$, causing a contradiction.

M = "On input w:

(1) Have M get its own description, $\langle M \rangle$.

(2) Try to ***recognize*** if M does not accept $\langle M \rangle$.
(This step might loop forever if M does accept $\langle M \rangle$).

(3) If M will not accept $\langle M \rangle$, choose to accept w."

(4) If M will accept $\langle M \rangle$, choose to reject w.

Theorem: $L_D \notin$ **RE**.

Proof: By contradiction; assume that $L_D \in$ **RE**. This means that TMs can recognize when arbitrary TMs do not accept their own encodings. Given this, we could construct the following TM:

M = "On input w:
Have M obtain its own description, $\langle M \rangle$.
Try to *recognize* if M will not accept $\langle M \rangle$.
(This step might loop forever if M accepts $\langle M \rangle$.)
If M will not accept $\langle M \rangle$, then M chooses to accept w.
If M will accept $\langle M \rangle$, then M chooses to reject w."

Now, either M does not accept $\langle M \rangle$ or M does accept $\langle M \rangle$. If M does not accept $\langle M \rangle$, then in the recognition step M will determine this, then proceed to accept w regardless of what w is. In particular, this means that M will accept $\langle M \rangle$, contradicting that it doesn't accept $\langle M \rangle$. On the other hand, if M accepts $\langle M \rangle$, then one of two things happen. First, M might loop in the recognition step, in which case M loops on all inputs, including $\langle M \rangle$, a contradiction. Second, M might determine that it accepts $\langle M \rangle$ in the recognition step, in which case it rejects w regardless of what w is. In particular, this means that M rejects $\langle M \rangle$, a contradiction.

In all cases we reach a contradiction, so our assumption must have been wrong. Therefore, $L_D \notin$ **RE**. ■

Another Non-**RE** Language

The Complement of A_{TM}

- Recall: the language A_{TM} is the language of the universal Turing machine U_{TM}:

$$\mathbf{A_{TM} = \mathscr{L}(U_{TM}) = \{ \langle \mathit{M}, \mathit{w} \rangle \mid \mathit{M} \text{ is a TM and } \mathit{M} \text{ accepts } \mathit{w} \}}$$

- The complement of A_{TM} (denoted $\overline{A}_{TM}$) is the language of all strings not contained in A_{TM}.
- Questions:
  - What language is this?
  - Is this language in **RE**?

A_{TM} and $\overline{A}_{TM}$

- The language A_{TM} is defined as

  $\{\langle M, w\rangle \mid M$ is a TM that accepts $w\}$

- Equivalently:

  $\{x \mid x = \langle M, w\rangle$ for some TM M and string w, and M accepts $w\}$

- Thus $\overline{A}_{TM}$ is

  $\{ x \mid x = \langle M, w\rangle$ for some TM M and string w where M does not accept w, or x is not the encoding of any TM/string pair. $\}$

Comparing L_D and $\overline{A}_{TM}$

- The languages L_D and $\overline{A}_{TM}$ are closely related:
  - L_D: Does M not accept $\langle M \rangle$?
  - $\overline{A}_{TM}$: Does M not accept string w?
- Intuitively, $\overline{A}_{TM}$ seems like a more general version of L_D.
- Since $L_D \notin \mathbf{RE}$, we can reasonably expect that $\overline{A}_{TM} \notin \mathbf{RE}$.

$\overline{A}_{TM} \notin$ **RE**

- Suppose that $\overline{A}_{TM} \in$ **RE**.
- Then a TM can *recognize* when an arbitrary TM will not accept an arbitrary string.
- What does this machine do?

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept w.
  (This step may loop forever if M accepts w).
- If M will not accept w, choose to accept w.
- If M will accept w, choose to reject w."

$\overline{A}_{TM} \notin$ RE

- ***Case 1:*** *M* does not accept *w*.
- Then in the recognition step, *M* will recognize this and proceed to the third line.
- *M* then accepts *w*, contradicting that it doesn't accept *w*.

M = "On input *w*:

- Have *M* get its own description, ⟨*M*⟩.
- Try to ***recognize*** if *M* does not accept *w*.
  (This step may loop forever if M accepts w).
- If *M* will not accept *w*, choose to accept *w*.
- If *M* will accept *w*, choose to reject *w*."

$\overline{A}_{TM} \notin$ RE

- ***Case 2:*** M does accept w.
- Then in the recognition step, M will either loop infinitely and not accept w (contradicting that it accepts w) or continue to line 4.
- In line 4, M rejects w, contradicting that it accepts w.

M = "On input w:

- Have M get its own description, $\langle M \rangle$.
- Try to ***recognize*** if M does not accept w.
  (This step may loop forever if M accepts w).
- If M will not accept w, choose to accept w.
- If M will accept w, choose to reject w."

Theorem: $\overline{A}_{TM} \notin$ **RE**.

Proof: By contradiction; assume that $\overline{A}_{TM} \in$ **RE**. This means that TMs can recognize TM/string pairs that belong to $\overline{A}_{TM}$. Equivalently, this means that Turing machines can recognize when an arbitrary TM does not accept an arbitrary string. Given this, we could then construct the following TM:

M = "On input w:
Have M obtain its own description, $\langle M \rangle$.
Try to *recognize* if M will not accept w.
(This step might loop forever if M accepts w.)
If M will not accept w, then M chooses to accept w.
If M will accept w, then M chooses to reject w."

Now, either M does not accept w or it does accept w. If M does not accept w, then it will determine this in the recognition step, then proceed to accept w, contradicting that it does not accept w. If M does accept w, then it will either loop infinitely in the recognition step (contradicting that it accepts w), or it will determine this, then proceed to reject w (contradicting that it accepts w).

In all cases we reach a contradiction, so our assumption must have been wrong. Therefore, $\overline{A}_{TM} \notin$ **RE**. ■

Time-Out for Announcements!

Problem Set Eight

- Problem Set Eight goes out right now, is due on Monday, December 1 (right after the break.)
  - Explore impossible problems, NTMs, verifiers, and the full landscape of computability theory!
  - Limited / no office hours over break; just a heads-up.
- Problem Set Seven due at the start of today's lecture.

Midterms Graded

- Your amazing TAs graded all of the midterms this weekend.
- They'll be available, with solutions, outside after class.
- After that, they'll be available for pickup outside my office.
- Want to request a regrade? Please do so by the Monday after break by physically handing your exam to Keith right before or right after class.
- ***Please don't dealphabetize the exams!*** Really, just don't. That's super selfish.

A Note on the Honor Code

- As a friendly reminder, our Honor Code policy requires you to cite any assistance you receive on the problem sets beyond help from teammates, course materials, and course staff.
- As long as you cite your sources, you are not in violation of the Stanford Honor Code.
- Looked something up online? Talked with other people? Blatantly cheated by copying answers from an old solution set or a friend who previously took CS103? Be sure to include a citation!
- Our policy is to allow citations to be added to solutions up to three days after you submit. Please feel free to take advantage of this.

Your Questions!

…will be answered on Wednesday. (Sorry, we need to get through the rest of the material for today.)

Back to CS103!

Something I Was Going to Do

- I was going to take this time to show you an example of a language that was neither **RE** nor co-**RE**, but unfortunately we don't have the time.
- Instead, you'll see one on the problem set.
- I'll also release a set of extra slides on the course website detailing a monstrously hard problem that's neither **RE** nor co-**RE**. You can peruse them at your leisure.
- Instead, I want to talk about...

Redefining Problem-Solving

Redefining Problem Solving

- Initially, we saw the **R** and **RE** languages as our notion of “solving” a problem.
- After seeing the co-**RE** languages, we realized that many “impossible” problems were actually possible.
- Let's now look at two new versions of problem-solving and see how they relate back to **R**, **RE**, and co-**RE**:
  - Verifiers.
  - Nondeterministic TMs.

An Intuition for **RE**

- Intuitively, a language L is in **RE** if a TM can search for positive proof that a string w belongs to L.
- Such a machine could work as follows:
  - Find a possible proof.
  - Check the proof.
  - If correct, accept!
  - If not, try the next proof.
- **Question:** The hard part is usually the checking, not the searching. Can we skip the searching part?

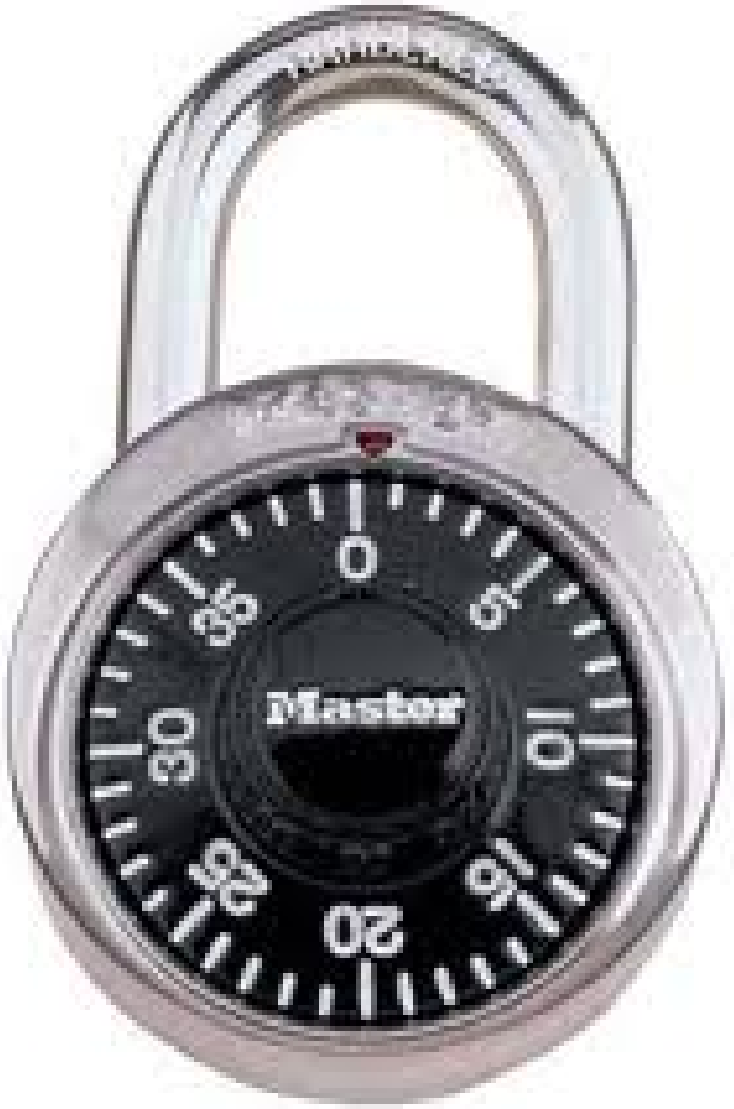
Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V is a decider (that is, V halts on all inputs.)
  - For any string $w \in \Sigma^*$, the following is true:

    $$\boldsymbol{w \in L \quad \text{iff} \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Intuitively, what does this mean?

Intuiting Verifiers



Question: Can this lock be opened?

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V is a decider (that is, V halts on all inputs.)
  - For any string $w \in \Sigma^*$, the following is true:

    $w \in L$ iff $\exists c \in \Sigma^*$. V accepts $\langle w, c \rangle$
- Some notes about V:
  - If V accepts $\langle w, c \rangle$, then we're guaranteed $w \in L$.
  - If V does not accept $\langle w, c \rangle$, then either
    - $w \in L$, but you gave the wrong c, or
    - $w \notin L$, so no possible c will work.

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V is a decider (that is, V halts on all inputs.)
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \text{iff} \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Some notes about V:
  - If $w \in L$, a string c for which V accepts $\langle w, c \rangle$ is called a ***certificate*** for c.
  - V is required to halt, so given any potential certificate c for w, you can check whether the certificate is correct.

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V is a decider (that is, V halts on all inputs.)
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \text{iff} \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Some notes about V:
  - Notice that $\mathscr{L}(V) \neq L$. Intead:

$$\boldsymbol{\mathscr{L}(V) = \{\ \langle w, c \rangle \mid w \in L \text{ and } c \text{ is a certificate for } w\ \}}$$

  - The job of V is just to check certificates, not to decide membership in L.

Some Verifiers

- Let L be the following language:

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

- Let's see how to build a verifier for L.
- A certificate for a grammar G string w should convince us that G accepts w. What kind of information would help us with that?
- One option: Let the certificate be a possible derivation of w from the start symbol.
- Our verifier then just needs to check whether the derivation is valid.

Some Verifiers

- Let L be the following language:

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

- Here is one possible verifier for L:

> V = "On input $\langle G, w, c \rangle$, where G is a CFG:
> Check whether c is a valid derivation of w
> from the start symbol of G.
> If so, accept. If not, reject."

- If the certificate is a correct derivation, we know for a fact that G can generate w.
- If not, we can't tell whether we got a bad certificate or whether G doesn't generate w.

Some Verifiers

- Let L be the following language:

$$L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$$

- Let's see how to build a verifier for L.
- A certificate for $\langle n \rangle$ should convince us that the hailstone sequence terminates for n. A bad certificate shouldn't leave us running forever.
- A thought: if the hailstone sequence terminates for n, then it has to terminate in some number of steps.
- Let the certificate be that number of steps.

Some Verifiers

- Let L be the following language:

$$L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$$

V = "On input $\langle n, k \rangle$, where $n, k \in \mathbb{N}$.
Check that $n \neq 0$.
Run the hailstone sequence, starting at n, for at most k steps.
If after k steps we reach 1, accept.
Otherwise, reject."

- Do you see why $\langle n \rangle \in L$ iff there is some k such that V accepts $\langle n, k \rangle$?

What languages are verifiable?

Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof idea:*** Build a recognizer that tries every possible certificate to see if $w \in L$.
- ***Proof sketch:*** Show that this TM is a recognizer for L:

M = "On input w:
    For $i = 0$ to ∞
        For each string c of length i:
            Run V on $\langle w, c \rangle$.
            If V accepts $\langle w, c \rangle$, M accepts w."

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof idea:*** If $L \in$ **RE**, there is a recognizer M for L.
- If $w \in L$, M accepts w after some number of steps. If $w \notin L$, M never accepts.
- Have the certificate for w be the number of steps before M accepts w.

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof sketch:*** Let L be an **RE** language and let M be a recognizer for it. Then show that this is a verifier for L:

V = "On input $\langle w, n \rangle$, where $n \in \mathbb{N}$:
Run M on w for n steps.
If M accepts w within n steps, accept.
If M did not accept w in n steps, reject."

Verifiers and **RE**

- The verifier definition of **RE** gives a new perspective on **RE** languages.
- Previously, we could think about *searching* for proof that $w \in L$.
- Now, we can think about *checking* a proof that $w \in L$ without asking where that proof came from.
- A language is in **RE** if there is a way to check a proof that $w \in L$.

Next Time

- **Nondeterministic TMs**
  - How powerful are NTMs? How do they compare to normal TMs?
- **Intro to Complexity Theory**
  - How do we measure the complexity of a solution to a problem?
- **The Cobham-Edmonds Thesis**
  - An analog of Church-Turing for efficiency.
- **The Complexity Classes P and NP**
  - Two fundamental complexity classes for efficient computation.