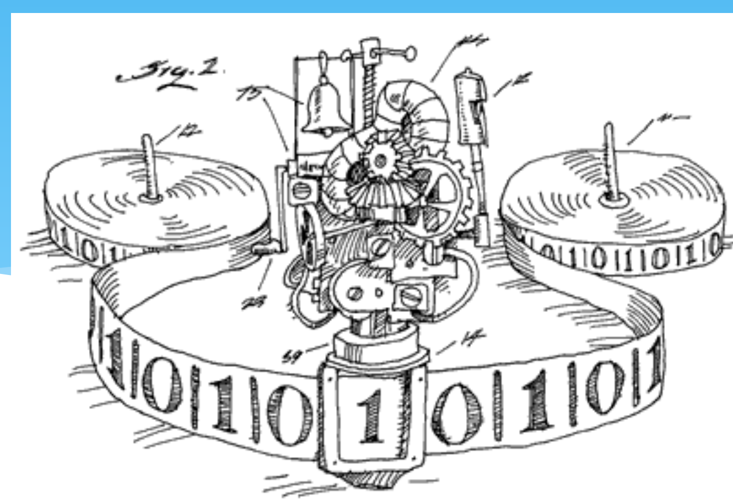
EECS 376: Foundations of Computer Science

Chris Peikert

6 February 2023



Last Weekend: OH**

EECS ECE CSE

Today's Agenda

* Recap: (un)countability
* "Almost all" problems are undecidable
* The Barber Paradox & an explicit undecidable language
* Two more undecidable problems:
  * Accepting problem L_{ACC}
  * Halting problem L_{HALT}
* Turing reductions/reducibility

Review: Countability

* A set S is ***countable*** if there is an ordered list of its elements (each element $s \in S$ gets its own unique natural number ID)

* The set of all *finite* binary strings is countable.
  * List the elements in lexicographic order: $\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$

* The set of all *infinite* binary sequences is uncountable.
  * Idea: For any list of such sequences, use *diagonalization* to construct a sequence that is *not* in the list.

Language $\equiv$ Infinite Binary Sequence

- **Claim:** A language L is *equivalent* to an *infinite* binary sequence.
- **Idea:** First, list $\Sigma^* = \{s_1, s_2, s_3, s_4, \ldots\}$ lexicographically.

  Then $L \equiv b_1 b_2 b_3 b_4 \cdots$, where $b_i \in \{0,1\}$ indicates if $s_i \in L$.

- **Example:**

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$$

$$\text{L} = \{\varepsilon, \quad 00, 01, 10, 11, \quad \ldots\}$$

$$\equiv 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \cdots$$

There is an Undecidable Language

* Set of all TMs (programs) is countable:
  * Can list them by their "source code" (descriptions as strings)
  * $\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \ldots \in \{0,1\}^*$
* Set of all decidable languages is countable:
  * Each TM decides at most one language
* Set of all languages (infinite binary strings) is uncountable
* So, there is an undecidable language!
  Indeed, "almost all" languages are undecidable.
  (Similarly, "almost all" real numbers are irrational…)

Diagonalization Yields An Undecidable Language

- List all deciders M_i w/ their decided langs $L(M_i)$ (seq's).
- There is a language L^* that is <u>not in the list</u>: 'flip the diagonal'.
- **Claim:** No TM decides L^*. Every decider M_i appears in the list, but 'behaves incorrectly' on string s_i !

X

	s1	s2	s3	s4	s5	s6	…
M1	1	0	0	1	1	0	…
M2	0	1	1	0	0	0	…
M3	1	1	1	1	1	1	…
M4	0	0	0	0	0	0	…
M5	1	0	1	0	0	0	…
…	…	…	…	…	…	…	…

L*	0	0	0	1	1	…

Summary

The ‘good’: We showed that there is an undecidable language

The ‘bad’: This language is “non-constructive” and “unnatural”

Q: Can we say anything about **“natural” problems** that we care about, or are “useful”?

The Barber Paradox

- **Sign:** *"Barber B is the best barber in town! B cuts the hair of <u>all</u> those—and <u>only</u> those—who do not cut their own hair."*

- **Question:** Who cuts B's hair?
- **Answer:** Consider some person P.

1. P cuts own hair $\Longrightarrow$ B does not cut P's hair.
2. P does not cut own hair $\Longrightarrow$ B cuts P's hair.

- **Question:** What if $P = B$?

1. B cuts own hair $\Longrightarrow$ B does not cut B's hair.
2. B does not cut own hair $\Longrightarrow$ B cuts B's hair.

Contradiction! **Such a barber cannot exist!**

Barber Paradox and Diagonalization

- Let X be a list of everyone in town, and whose hair they cut:

$$X(i,j) = \begin{cases} 1 & \text{if } P_i \text{ cuts } P_j\text{'s hair} \\ 0 & \text{otherwise} \end{cases}$$

- Barber B is the flipped diagonal: $B(i)$ is 0 if P_i cuts P_i's hair, else 1.
- Thus, B is not on the list X – the barber does not exist!

X

	P1	P2	P3	P4	P5	…
P1	0	1	1	0	1	…
P2	1	1	0	0	0	…
P3	0	0	0	0	0	…
P4	0	1	1	1	0	…
…	…	…	…	…	…	…

B	1	0	1	0	…	

Quote of The Day

"I would not join any club that would have me as a member."

– Groucho Marx

Barber Paradox & Computability

Source Code as Input

- Since **a program's source code is just a string**, it can be passed as input to another program—or even to the program itself!
- **Example:**

```
int M1(string s) {
… return 0, else
return 1;
}
```

$\langle M_1 \rangle =$ "int M1(string s) {\n… return 0, else \n return 1;\n}\n"

Q: What does $M_1(\langle M_1 \rangle)$ return?

The Barber Paradox (Part 2)

- **Sign:** *"Barber B is the best barber in town! B cuts the hair of all those—and only those—who do not cut their own hair."*
- Let's consider a computational analogy, where:
  - barber, people $\Rightarrow$ program(s)
  - hair $\Rightarrow$ source code
  - cut $\Rightarrow$ accept

- **Result:** *"Program B accepts the source code of all programs—and only those programs—that do not accept their own source code."*

- **Reminder:** The ***language of a program*** is the set of inputs it *accepts*.

  Thus, $L(B) = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$

The Barber Paradox (Part 2)

- **Result:** *"Program B accepts the source code of all programs—and only those programs—that do not accept their own source code."*

- **Question:** Does program B accept its own source code?
- **Answer:** Suppose P is a program.

1. P accepts its own code $\Longrightarrow$ B does not accept P's code.
2. P does not accept its own code $\Longrightarrow$ B accepts P's code.

- **Question:** What if $P = B$?

1. B accepts its own code $\Longrightarrow$ B does not accept B's code.
2. B does not accept its own code $\Longrightarrow$ B accepts B's code.

Contradiction! **Program B cannot exist**

Barber and Diagonalization (Part 2)

* Let X be a list of all programs, and whose code they accept:

$$X(i,j) = \begin{cases} 1 & \text{if } P_i \text{ accepts } P_j\text{'s code} \\ 0 & \text{otherwise} \end{cases}$$

* Program B is the flipped diagonal: $B(i)$ is 0 if P_i accepts P_i's code, 1 otherwise.
* Thus, B is not on the list X - program B does not exist!

X

	P1	P2	P3	P4	P5	…
P1	0	1	0	1	1	…
P2	1	1	0	1	0	…
P3	0	0	0	0	0	…
P4	1	1	1	0	1	…
…	…	…	…	…	…	…

B	1	0	1	1	…	

An Explicit Undecidable Language

- Since no program has
  $$L_{\text{BARBER}} = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$$
  as its language, L_{BARBER} is ***undecidable***.

- This is our first example of an explicit undecidable language!

- **Q:** Why do we care about this language?
- **A:** Once we have one undecidable language, we can use it to show that other languages are also undecidable.

A More Useful Language

It would be useful to have a program that does the following:

Given as input a Turing Machine M and a string x, determine whether M accepts x.

```
bool M(string x):
n ← 100
while (n > 1):  n ← n − 1
return T
```

Example: Does M accept x=376 ?

In other words, is the following language decidable?:

$$L_{\mathrm{ACC}} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$$

Attempt 1: Interpreter ("Universal" Turing Machine)

- An ***interpreter*** is a program that takes another program as input and simulates its behavior.

- Specifically, an interpreter U takes two inputs: (1) some source code $\langle M \rangle$, and (2) a string x.

- $U(\langle M \rangle, x)$ just simulates the execution of M on x. So:
  - M accepts $x \implies U$ accepts $(\langle M \rangle, x)$
  - M rejects $x \implies U$ rejects $(\langle M \rangle, x)$
  - M loops on $x \implies U$ loops on $(\langle M \rangle, x)$

Does Interpreter Decide L_{ACC} ?

- $U(\langle M \rangle, x)$ simulates the execution of M on x:
  - M accepts $x \implies U$ accepts $(\langle M \rangle, x)$
  - M rejects $x \implies U$ rejects $(\langle M \rangle, x)$
  - M loops on $x \implies U$ loops on $(\langle M \rangle, x)$

In both cases $(\langle M \rangle, x) \notin L(U)$

- The language of U is:

$$L(U) \equiv L_{\text{ACC}} = \Big\{ \big(\langle M \rangle, x\big) : M \text{ accepts } x \Big\}$$

- However, U is ***not a decider*** for L_{ACC}: U loops on some inputs.
- **Q:** Can we write a decider for L_{ACC}?

L_{ACC} is Undecidable

* **Q:** Is there a decider for $L_{\text{ACC}} = \left\{ \left(\langle M \rangle, x \right) : M \text{ accepts } x \right\}$?
* A hypothetical decider C for L_{ACC} must behave as follows:
  * M accepts $x \Longrightarrow C$ accepts $(\langle M \rangle, x)$
  * M does not accept $x \Longrightarrow C$ rejects $(\langle M \rangle, x)$

M rejects or loops

* Could try to come up with a clever diagonalization.
  But let's not reinvent the wheel!
* **IDEA:** show that such C would let us decide some known-undecidable language: contradiction! Via (Turing) reduction

L_{ACC} is Undecidable

We need to implement:

B takes one input: $\langle M \rangle$

M does not accept $\langle M \rangle \Longrightarrow B$ accepts $\langle M \rangle$

M accepts $\langle M \rangle \Longrightarrow B$ rejects $\langle M \rangle$

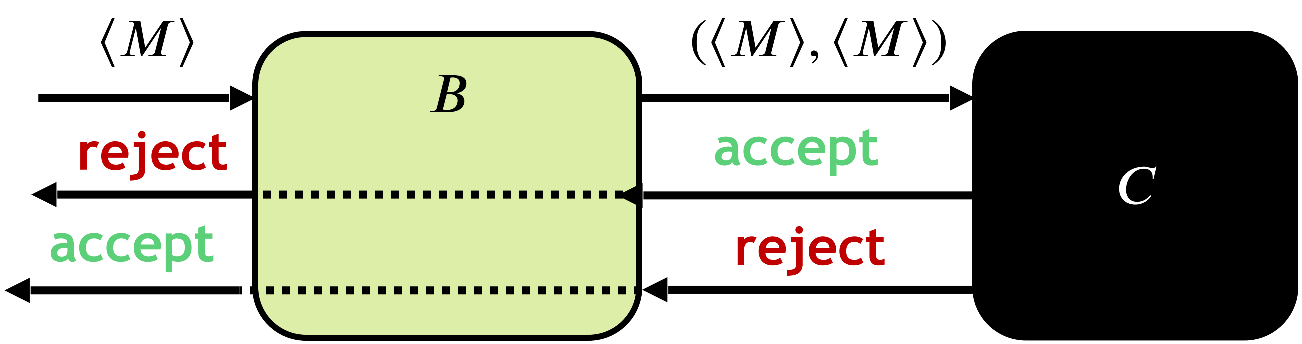
We have:

C takes two inputs: $\langle M \rangle$ and x.

M accepts $x \Longrightarrow C$ accepts $\big(\langle M \rangle, x\big)$

M doesn't accept $x \Longrightarrow C$ rejects $\big(\langle M \rangle, x\big)$

* **Proof:** Assume (for contradiction) that a decider C exists for $L_{\mathrm{ACC}} = \big\{\big(\langle M \rangle, x\big) : M \text{ accepts } x\big\}$.
  We will use C to construct a decider B for $L_{\mathrm{BARBER}} = \big\{\langle M \rangle : M \text{ does not accept } \langle M \rangle\big\}$:



L_{ACC} is Undecidable

We need to implement:

B takes one input: $\langle M \rangle$

M does not accept $\langle M \rangle \implies B$ accepts $\langle M \rangle$

M accepts $\langle M \rangle \implies B$ rejects $\langle M \rangle$

We have:

C takes two inputs: $\langle M \rangle$ and x

M accepts $x \implies C$ accepts $\big(\langle M \rangle, x\big)$

M doesn't accept $x \implies C$ rejects $\big(\langle M \rangle, x\big)$

- **In code:**
- B on input $\langle M \rangle$: *(B takes one input: source code of some program M)*
  - Run C on $(\langle M \rangle, \langle M \rangle)$ *(Ask C: "does M accept its source code"?)*
  - If C accepts, ***reject***; if C rejects, ***accept*** *(flip C's answer)*

- **Analysis:**
  - B halts on any input, because C does (C is a decider).
  - $\langle M \rangle \in L_{\mathrm{BARBER}} \iff M$ does not accept $\langle M \rangle$ $\iff C$ rejects $(\langle M \rangle, \langle M \rangle) \iff B$ accepts $\langle M \rangle$.

EECS M ECE CSE

The Halting Problem and (More) Turing Reductions

Halting Problem

Halting Problem: Given a TM M and string x as input, decide if M halts on x.
Fundamental problem in software and hardware design!

- **Q:** Why can't we just run M on x (using an interpreter) to see if it halts?
  - **A:** M might loop on x, so we might never produce an answer!

A hypothetical decider H for L_{HALT} takes two inputs, $\langle M \rangle$ and x:

- M halts (accepts or rejects) on $x \Longrightarrow H$ accepts $\big(\langle M \rangle, x\big)$
- M doesn't halt (loops) on $x \Longrightarrow H$ rejects $\big(\langle M \rangle, x\big)$

Show that H can't exist, using undecidability of L_{ACC}. How?
Use hypothetical H to <u>construct a decider for</u> L_{ACC}.
Since L_{ACC} is undecidable, we get a contradiction.

L_{HALT} is Undecidable

We need to implement:

C is given two inputs: $\langle M \rangle$ and x

M accepts $x \implies C$ accepts $(\langle M \rangle, x)$

M does not accept x $\implies C$ rejects $(\langle M \rangle, x)$

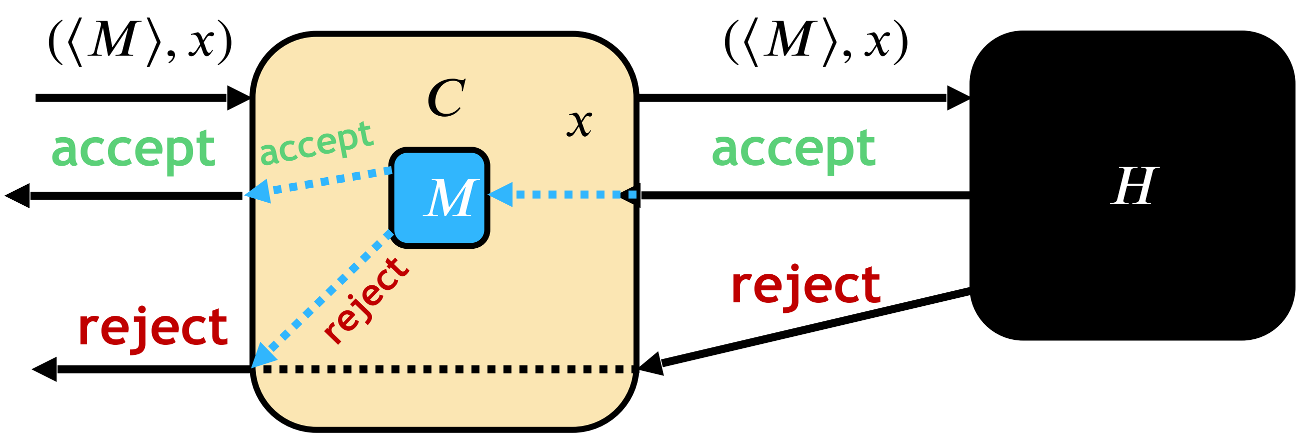
We have:

H takes two inputs: $\langle M \rangle$ and x

M accepts/rejects $x \implies H$ accepts $(\langle M \rangle, x)$

M loops on $x \implies H$ rejects $(\langle M \rangle, x)$

- **Claim:** $L_{\text{HALT}} = \{(\langle M \rangle, x) : M \text{ halts on } x\}$ is undecidable.
- **Proof:** Assume (for contradiction) that some H decides L_{HALT}. We construct a decider C for $L_{\text{ACC}} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$:



L_{HALT} is Undecidable

We need to implement:

C is given two inputs: $\langle M \rangle$ and x

M accepts $x \Longrightarrow C$ accepts$\left(\langle M \rangle, x\right)$

M does not accept $\langle M \rangle \Longrightarrow C$ rejects$\left(\langle M \rangle, x\right)$

We have:

H is given two inputs: $\langle M \rangle$ and x

M accepts or rejects $x \Longrightarrow H$ accepts $\left(\langle M \rangle, x\right)$

M loops on $x \Longrightarrow H$ rejects $\left(\langle M \rangle, x\right)$

- **In code:**
- C on input$(\langle M \rangle, x)$: *(Must answer: "does M accept x?")*
  - Run H on $(\langle M \rangle, x)$ *(Ask H: "does M halt on x?")*
  - If H rejects, ***reject*** *(M loops on x, so it doesn't accept it)*
  - If H accepts, run M on x *(M halts on x, so this simulation will terminate)*
    - If M accepts, ***accept***; If M rejects, ***reject*** *(answer as M does)*

- **Analysis:** C halts on any input (why?). Moreover:
  - M accepts $x \Longrightarrow H$ accepts $\left(\langle M \rangle, x\right) \Longrightarrow C$ accepts $\left(\langle M \rangle, x\right)$
  - M rejects $x \Longrightarrow H$ accepts $\left(\langle M \rangle, x\right) \Longrightarrow C$ rejects $\left(\langle M \rangle, x\right)$
  - M loops on $x \Longrightarrow H$ rejects $\left(\langle M \rangle, x\right) \Longrightarrow C$ rejects $\left(\langle M \rangle, x\right)$

- **Conclusion:** L_{HALT} decidable $\Longrightarrow L_{\text{ACC}}$ decidable
- **Contrapositive:** L_{ACC} undecidable $\Longrightarrow L_{\text{HALT}}$ undecidable

Turing Reducibility

- **Question:** How can we show that a language is undecidable?
- **Two Options:**
  - Directly: L_{BARBER} is undecidable.
  - Indirectly: If L_{HALT} is decidable then so is L_{ACC}.

- **Definition:** Language A is ***Turing reducible*** to language B, written $\boldsymbol{A \leq_T B}$, if there exists a TM (program) M, with access to a *membership oracle* ("black box") for B, that decides A.

- **Intuition:** solving A is "no harder than" solving B.
- **We have shown:** $L_{\text{BARBER}} \leq_T L_{\text{ACC}} \leq_T L_{\text{HALT}}$

Conclusion and Exercises

* **Theorem:** Suppose $A \leq_T B$. Then B is decidable $\implies A$ is decidable.
* **Proof:** use decider for B to implement the membership oracle.
* **Question:** How can we show *undecidability* of a language?
* **Contrapositive:** Suppose $A \leq_T B$. Then A is *undecidable* $\implies B$ is *undecidable*.
* **Strategy:** Pick an undecidable language A and show that $A \leq_T B$.

* **Question 1:** Is it true that $L_{\mathrm{HALT}} \leq_T L_{\mathrm{ACC}}$?
* **Question 2:** Suppose $A \leq_T B$. Must $B \leq_T A$?