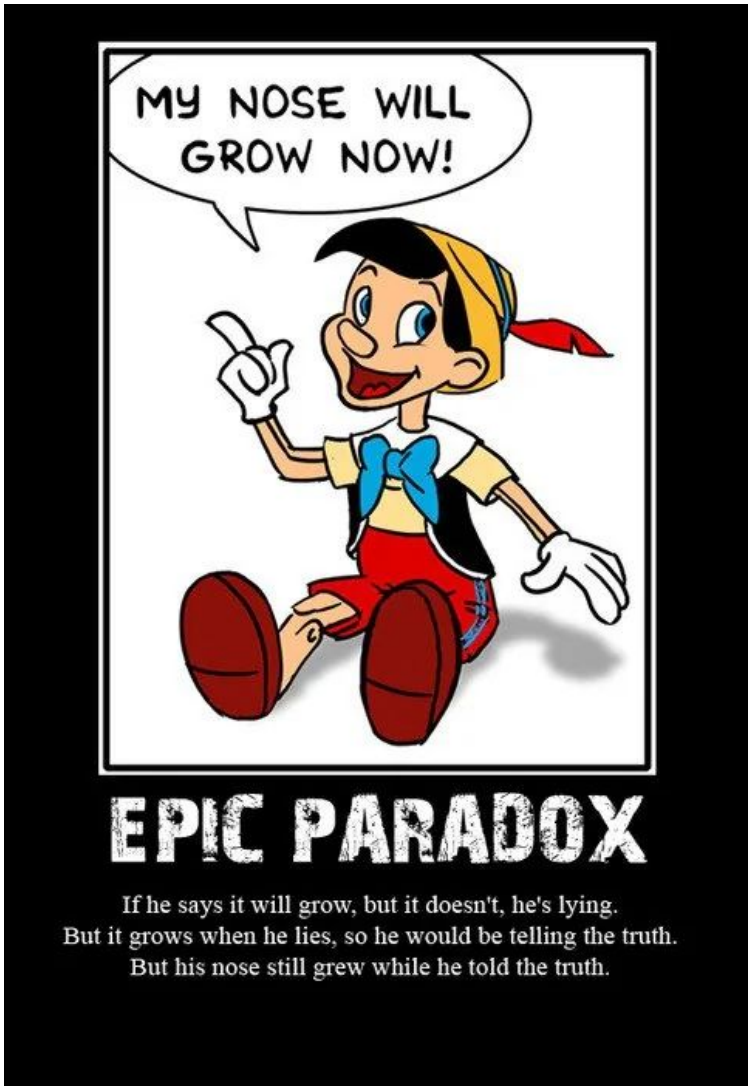
Undecidable Problems



Announcement

This week's HW is due 24 hours later than usual on Thursday at 8pm

(Because some of the problems rely on content from today's lecture)

Quote of The Day

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

- Groucho Marx

The Barber Paradox
(aka Russell's Paradox)

"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: *Does B cut their own hair?*

Review: Definition of Countable

* **Definition:** A set S is ***countable*** if it is "no larger than" the naturals $\mathbb{N} = \{0,1,2, \dots\}$, i.e. $|S| \leq |\mathbb{N}|$.
* **Equivalently:** S is countable if there exists a 1-to-1 (injective) function $f: S \to \mathbb{N}$.
* We can also show S is countable by demonstrating how to list all the elements in S such that each element $s \in S$ appears *somewhere* on the list.

Review: There Exist Undecidable Languages

1. Set of all TMs is countable
   ⇒Set of all decidable languages is countable

2. Set of all languages is uncountable, via diagonalization argument

{0,1}* (i.e. TM inputs) →

TMs ↓

	s1	s2	s3	s4	s5	s6	...
M1	1	0	0	1	1	0	...
M2	0	0	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	...
...	...	...	...	...	...	...	...

Barber Paradox is actually Diagonalization in Disguise

"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."

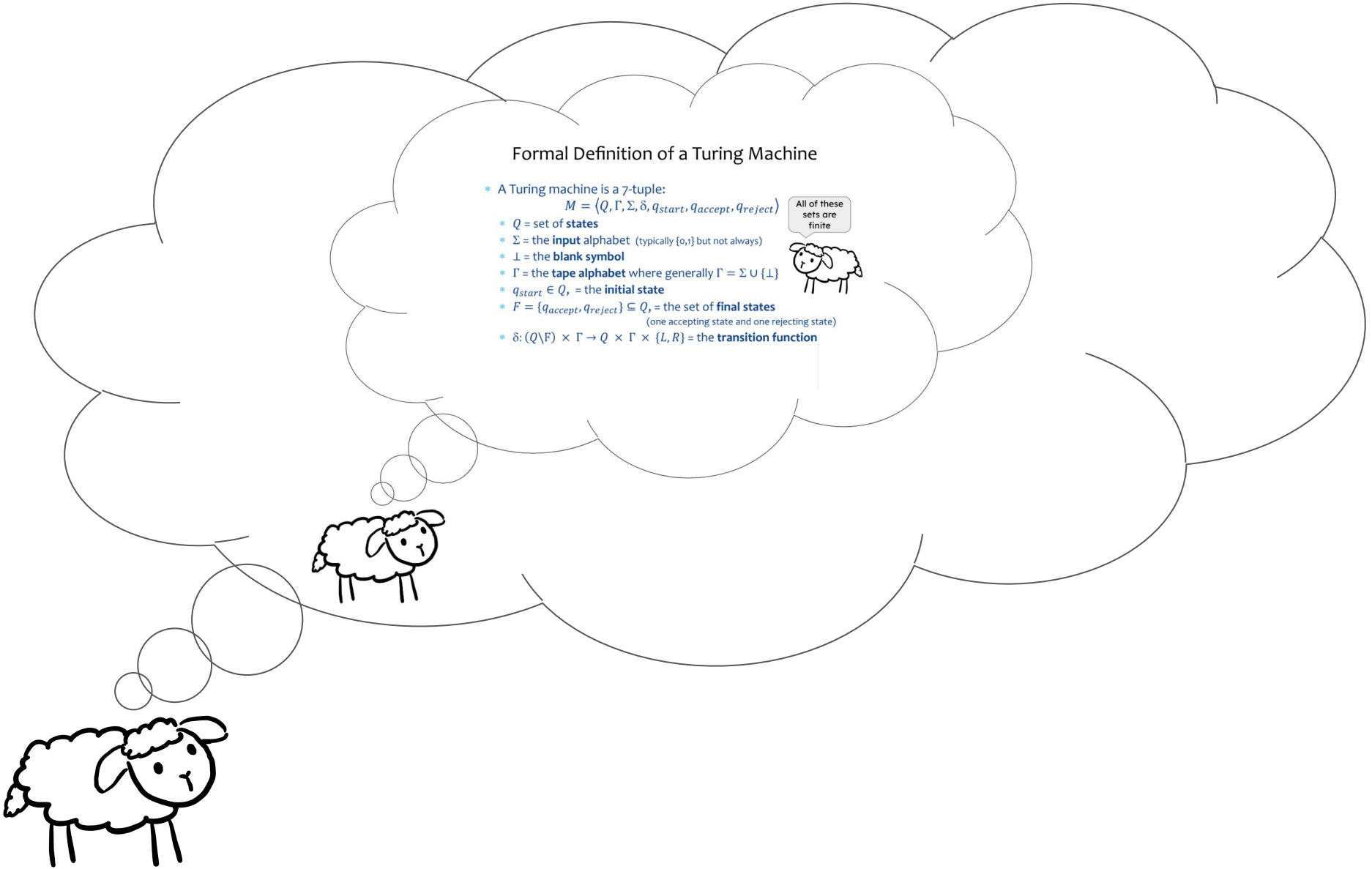
Make a table of everyone in town and whose hair they cut

	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	...	

Barber B is the flipped diagonal!

Thus, B is not on the list — the barber does not exist!

A TM can be represented as a finite string. Therefore, you can feed the description of a TM as input to a (different or same) TM.



Using the Barber Paradox to Construct an Undecidable Problem

"**Barber 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** = **TM(s)**
- hair = description of TM
- *cut* = *accept*

"***TM M*BARBER** *accepts* as input the description of all ***TMs***—and only those ***TMs***— that do not *accept* as input their own description."

Does **TM MBARBER** accept its own description?

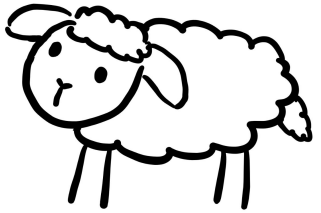
Again it's Diagonalization in Disguise

"***TM M_{BARBER}*** *accepts* as input the description of all ***TMs***—and only those ***TMs***— that do not *accept* as input their own description."

Make a table of which ***TMs*** *accept* as input the description of which ***TMs***

	M1	M2	M3	M4	M5	...
M1	0	1	1	0	1	...
M2	1	1	0	0	0	...
M3	0	0	0	0	0	...
M4	0	1	1	1	0	...
...	...	...	...	...	...	...
M_{BARBER}	1	0	1	0	...	

Notation: **M** is a TM and ⟨**M**⟩ is its description



TM M_{BARBER} is the flipped diagonal, so **M_{BARBER}** isn't on the list, so doesn't exist!

⇒ The language **L_{BARBER}** = {⟨**M**⟩: **M** does not accept ⟨**M**⟩} is undecidable.

A More "Natural" Undecidable Language

Input: Turing Machine **M** and a string **x**

Output: Does **M** accept **x**?

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

Why not just run M(x)?

Attempt 1: Interpreter

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

- Specifically, an interpreter **U** takes two inputs: (1) source code **⟨M⟩**, and (2) string **x**.

- **U** simulates the execution of **M** on input **x**:
  - **M** accepts **x** ⇒ **U** accepts (**⟨M⟩**, **x**)
  - **M** rejects **x** ⇒ **U** rejects (**⟨M⟩**, **x**)
  - **M** loops on **x** ⇒ **U** loops on (**⟨M⟩**, **x**)

- This is called the ***Universal Turing Machine*** (and it does exist)

L_{ACC} is Undecidable

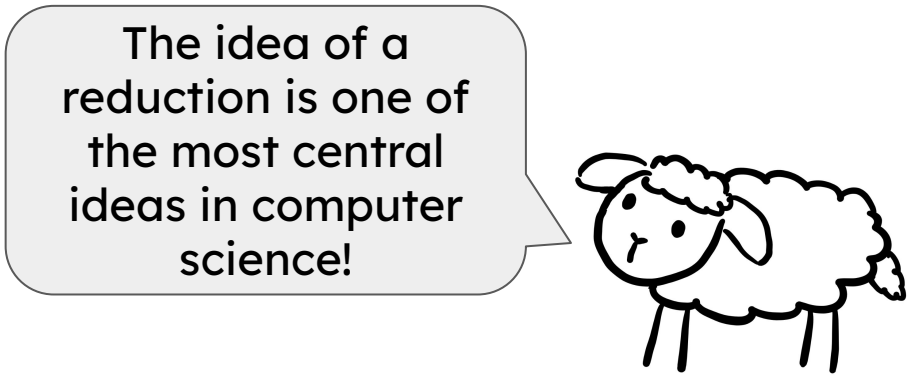
Could try to come up with a diagonalization proof …
…but let's not reinvent the wheel!

KEY IDEA: Once we have one undecidable language, we can use it to show that other languages are also undecidable!

HOW? Show that if L_{ACC} were decidable, this would let us decide some known undecidable language!

This is called a **reduction.**

(Specifically a Turing-reduction)

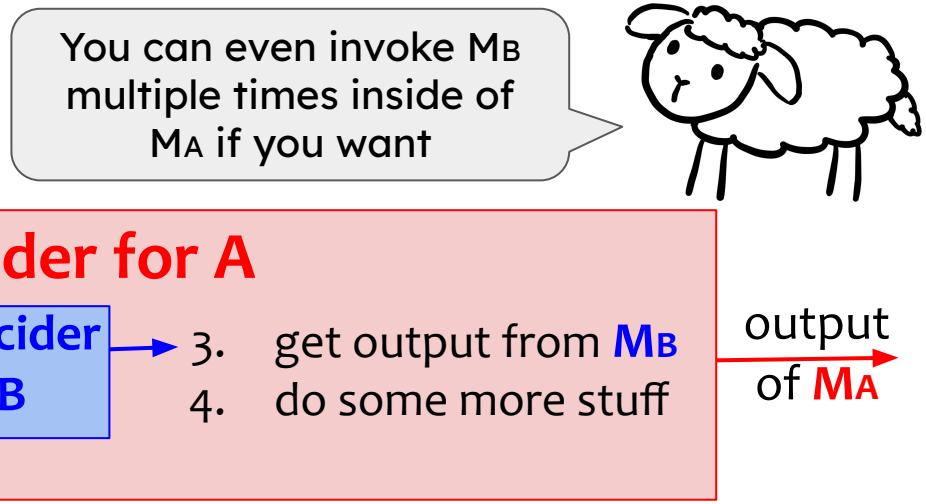


Turing Reduction from **A** to **B** (denoted **A** $\leq_T$ **B**):

"We can use a black-box decider for **B**
as a subroutine to decide **A**."

What it implies:

1. If **B** is decidable then **A** is decidable.
2. Contrapositive: If **A** is undecidable then **B** is undecidable.



"Problem **B** is at least as hard as Problem **A**"

Reduction from L_{BARBER} to L_{ACC} (i.e. $L_{BARBER} \leq_T L_{ACC}$)

We need to implement:

M_{BARBER} takes one input: $\langle M \rangle$

M does not accept $\langle M \rangle \Rightarrow M_{BARBER}$ accepts

M accepts $\langle M \rangle \Rightarrow M_{BARBER}$ rejects

Suppose we have:

M_{ACC} takes two inputs: $\langle M \rangle$, x

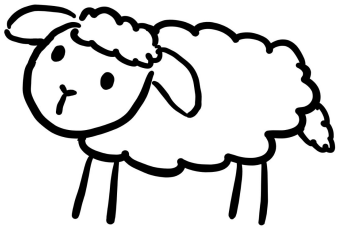
M accepts $x \Rightarrow M_{ACC}$ accepts

M does not accepts $x \Rightarrow M_{ACC}$ rejects

We need to specify the pseudocode:

$M_{BARBER}(\langle M \rangle)$:

We are allowed to use $M_{ACC}(\langle M \rangle, x)$ as a subroutine, with the inputs of our choice

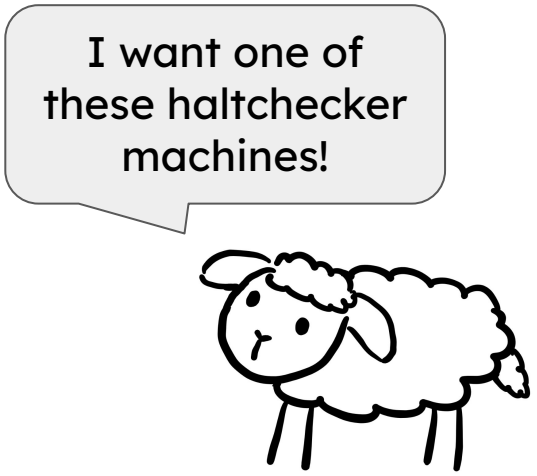


Another Undecidable Language:
Halting Problem

Input: Turing Machine **M** and a string **x**

Output: Does **M** halt when given input **x**?

Language: $L_{HALT} = \{(\langle M \rangle, x) : M \text{ halts on input } x\}$



Reduction from L_{ACC} to L_{HALT} (i.e. $L_{ACC} \leq_T L_{HALT}$)

We need to implement:

M_{ACC} takes two inputs: $\langle M \rangle$, x

M accepts $x \Rightarrow M_{ACC}$ accepts

M loops or rejects $x \Rightarrow M_{ACC}$ rejects

Suppose we have:

M_{HALT} takes two inputs: $\langle M \rangle$, x

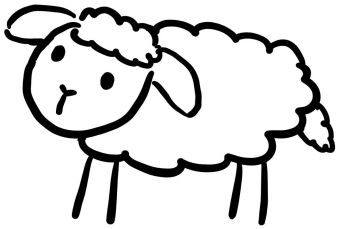
M accepts or rejects $x \Rightarrow M_{HALT}$ accepts

M loops on input $x \Rightarrow M_{HALT}$ rejects

We need to specify the pseudocode:

$M_{ACC}(\langle M \rangle, x)$:

We are allowed to use $M_{HALT}(\langle M \rangle, x)$ as a subroutine, with the inputs of our choice



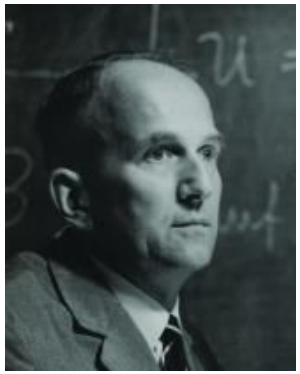
What about small programs?

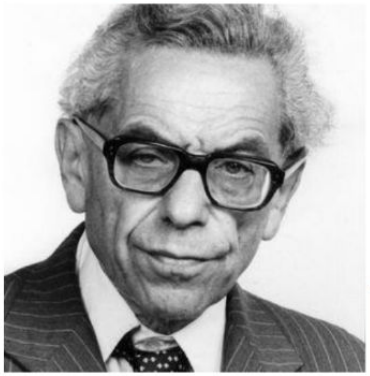
Question: What if you know the input program is <100 characters?
Then is the halting problem decidable? **Yes!**

But is an algorithm known? **No!**

Collatz Conjecture: This program halts for every n:

```
int n;
while (n > 1) {
  n = (n%2) ? 3*n+1 : n/2;
}
```





Paul Erdős offered $500 for this problem
Nobody knows the answer!

https://www.udiprod.com/halting-problem/