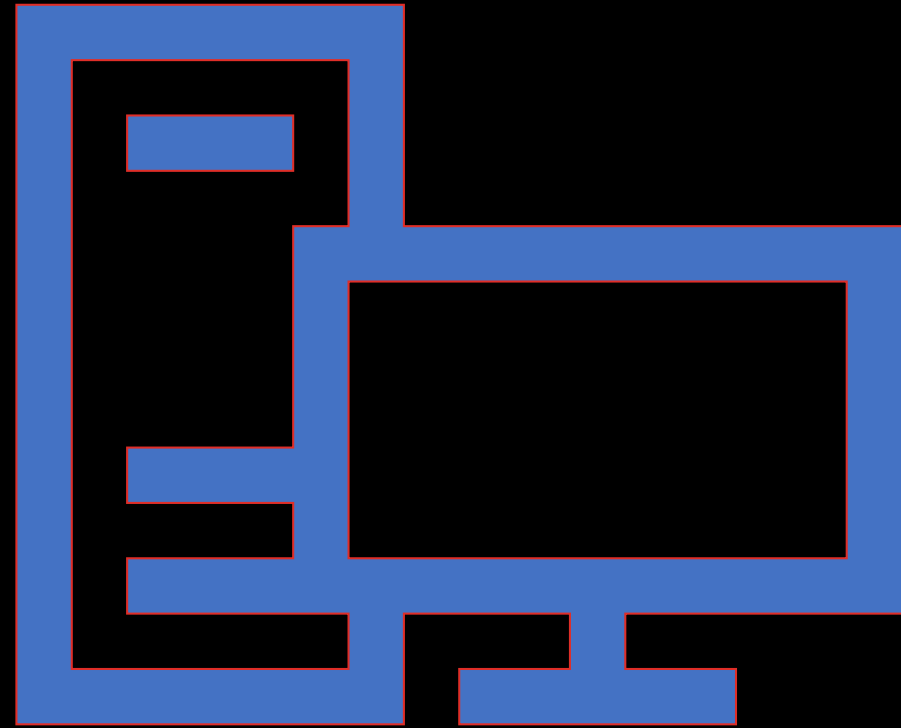


THEORY OF COMPUTATION

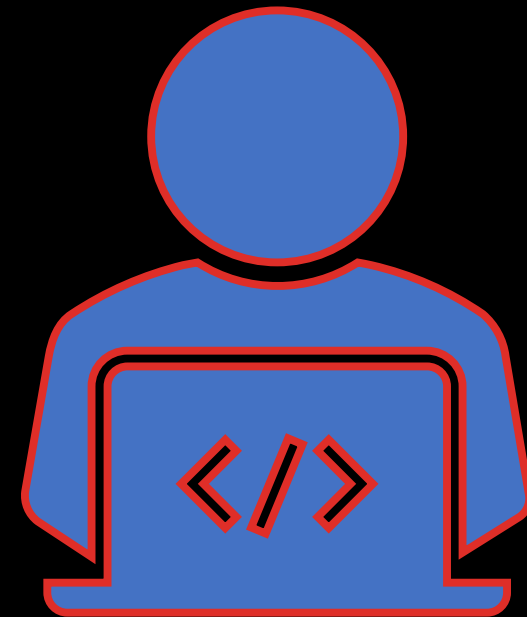


THE LIMITS OF COMPUTATION

Suppose you are given some computer code, and the input that code will run on. You want to know whether the code will halt, or enter an infinite loop.

- Compilers can catch very simple examples (such as “while(true)”), but they can’t catch more complicated examples.
- This would be a fantastically useful tool to have, if it existed. It doesn’t.
- Not only does no such tool exist, we have **proven** that no such tool **can** exist.
- This problem is known as the **Halting Problem**.

Before we can launch into the proof, we’re going to talk about a few side topics that will end up being related.



There is a barber who lives in a small village.
The barber cuts the hair of exactly the villagers
who do not cut their own hair.

What is the paradox?

- The barber neither cuts nor doesn't cut their own hair.


What is the solution to the paradox?

- I lied to you. There is no such barber.

This is a classic proof by contradiction: assume such a barber exists, and derive a contradiction.

- We will do something **very** similar when we prove the Halting Problem has no algorithmic solution.

THE BARBER PARADOX



INFINITELY-SIZED SETS

Consider the following two sets:

- The positive integers $\{1, 2, 3, \dots\}$
- The positive even integers $\{2, 4, 6, \dots\}$

Which set is bigger?

- In one sense, they're both infinite in size, so maybe they're the same?
- In another sense, the positive even integers are a strict subset of the positive integers, so maybe the positive integers are larger?
- In yet another sense, you could transform the positive integers into the positive even integers by multiplying each value by two, so maybe they're the same?

INFINITELY- SIZED SETS

Any of these intuitions could be the correct one, but mathematicians need to be consistent.

- They have chosen the last intuition to be the correct one.

Given two (possibly infinite) sets A and B, they are the same size exactly if there is a bijective function from A to B.

- That is, you can map the values in A to the values to B, so that everything in A is paired exactly once, and everything in B is paired exactly once.

$f(x) = 2x$ is the function that shows the positive integers are the same size as the positive even integers.

COUNTABLE SETS

A set is **countable**, or **countably infinite**, if it is the same size as the natural numbers (all integers 0 or larger).

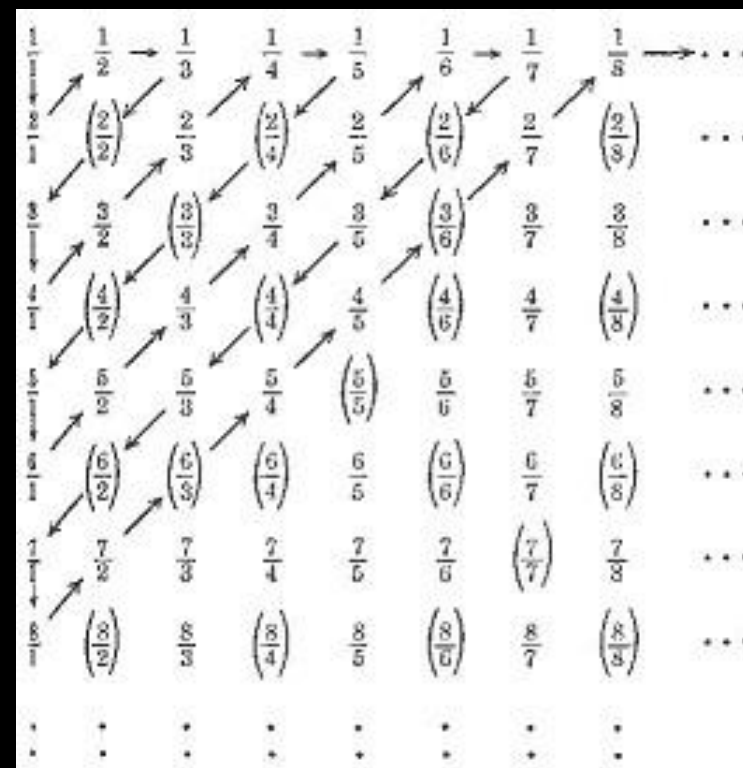
Are the set of even integers are countable (including the negatives)?

- Yes, $f(x) = x$ if x is even, $f(x) = -(x+1)$ if x is odd.

Are the set of positive rational numbers countable?

- Yes (surprisingly!)

Are the set of real numbers countable?



Proof by contradiction: suppose we have a bijective function f which maps the set of natural numbers to the set of real numbers. It might start something like this:

- $f(1) = \pi$
- $f(2) = e$
- $f(3) = 0.123456789$
- $f(4) = 32.33$ (repeating, of course), etc.

We will construct a real number $b < 1$, that definitively does not show up as an output from this function, using a process called **diagonalization**.

AN UNCOUNTABLE SET

THE CONTRADICTION

If the first decimal place of $f(1)$ is 1, then the first decimal place of b will be 2. Otherwise, it will be 1.

- If the i^{th} decimal of $f(i)$ is 1, then the i^{th} decimal of b will be 2. Otherwise, it will be 1.

In the prior example function,
 $b = 0.2211\dots$

- Does any input produce b ?
- No, because if it is $f(j)$, then they disagree at the j^{th} decimal place.



| | | |
|----|---|---------------|
| 0. | 1 | 7638725408... |
| 0. | 2 | 3481003527... |
| 0. | 9 | 481035829... |
| 0. | 0 | 1894723901... |
| 0. | 3 | 3717950224... |
| | ⋮ | |

SO WHAT, WHO CARES?

What do countable and uncountable sets have to do with Computer Science?

The number of algorithms is countable

- After compilation, an algorithm is just a binary string. There are countably many binary strings.

The number of **problems** is uncountable.

- If proven, this shows that there must be a problem which cannot be solved by any algorithm (and then we will prove the Halting Problem is one such problem).

I'VE GOT 99 PROBLEMS...

First lets define a problem:

- A problem takes as input a single integer, and outputs a T/F value.
- Note that this is actually a very restrictive definition, there are actually plenty more problems that output integers or take multiple inputs, etc.
- We will show that there are uncountably many problems even under this restrictive definition.

Since there are countably many algorithms, we will try to find a bijective function from the algorithms to the problems (via a proof by contradiction).

Assume that there is such a function.

THE CONTRADICTION

Here is a problem P that is not on this table:

- If M_1 outputs T on input 1, then P outputs F, otherwise it outputs T.
- If M_i outputs T on input i , then P outputs F, otherwise it outputs T.

Does any algorithm produce P ?

- No, if you assume the j^{th} algorithm does, you can see that they disagree on the output when given j as input.

| Algorithm | f(1) | f(2) | f(3) | f(4) | ... |
|-----------|------|------|------|------|-----|
| M_1 | F | T | T | F | ... |
| M_2 | F | T | F | T | ... |
| M_3 | T | T | T | T | ... |
| M_4 | F | F | F | F | ... |
| ... | ... | ... | ... | ... | ... |



THE HALTING PROBLEM

We will now use the previous examples to show that the Halting Problem has no algorithmic solution.

- Assume that it does: $H(M, w)$, which takes algorithm M , and the input w that M is supposed to run on, as input.

Using this library function, we will write a new program B (for barber):

```
B(computer code M)
  If ( $H(M, M) == \text{False}$ ) Then Return True
  Else Loop forever...
```

THE CONTRADICTION

First note that we are asking what would happen if M takes itself as input.

- Since there are countably many algorithms, we can refer to M by its index i , where M is the i^{th} algorithm, and run M on input i .

The contradiction arises when you run B on itself.

- If $B(B)$ halts, then $H(B,B)$ will return true. However, since $H(B,B)$ returns true, $B(B)$ will loop forever.
- If $B(B)$ loops forever, then $H(B,B)$ will return false. However, since $H(B,B)$ returns false, $B(B)$ will terminate.

```
B(computer code M)
  If (H(M,M) == False)
  Then Return True
  Else Loop forever...
```

ALTERNATE EXPLANATION #1

The program B is the barber: when you ask if the barber cuts their own hair, you get a contradiction. If you ask B what it returns when run on itself, you get a contradiction.

- The solution is that I lied: there is no barber, and there is no B. This was a classic proof by contradiction.
- However, I literally gave you the code for B: it required only one library function: H!
- Therefore, H cannot possibly exist.

ALTERNATE EXPLANATION #2

In our proof to show there is an unsolvable problem, we explicitly used B as our example.

- On input M_i , B outputs the opposite thing that M_i outputs.
- Therefore, the problem B is not produced by any algorithm.

We explicitly showed that $B \leq H$.

- Therefore, if H is solvable, then B is solvable.
- B is not solvable, therefore H is not solvable.

Therefore, the Halting problem is **undecidable**.

| Algorithm | $f(M_1)$ | $f(M_2)$ | $f(M_3)$ | $f(M_4)$ | ... |
|-----------|----------|----------|----------|----------|-----|
| M_1 | F | T | T | F | ... |
| M_2 | F | T | F | T | ... |
| M_3 | T | T | T | T | ... |
| M_4 | F | F | F | F | ... |
| ... | ... | ... | ... | ... | ... |



THE POST CORRESPONDENCE PROBLEM

In the **Post Correspondence Problem** (PCP), you are given a collection of dominoes.

- Each domino has two strings, one on the top, the other on the bottom.
- Your task is to make a sequence of dominoes so that the string obtained by concatenating the top strings is the same as the one obtained by the bottom strings.
- You may use each domino as many times as you like
- This problem sounds like a fairly standard algorithms problem... but this problem is *undecidable*!

SOME EXAMPLES

Is there a solution for this collection?

$$\left\{ \left[\frac{abc}{ab} \right], \left[\frac{ca}{a} \right], \left[\frac{acc}{ba} \right] \right\}$$

- No, because there is a 'c' on every top string, but none on the bottom.
- No, because every top string is lower than its bottom string.

Is there a solution for this collection?

$$\left\{ \left[\frac{b}{ca} \right], \left[\frac{a}{ab} \right], \left[\frac{ca}{a} \right], \left[\frac{abc}{c} \right] \right\}$$
$$\left[\frac{a}{ab} \right] \left[\frac{b}{ca} \right] \left[\frac{ca}{a} \right] \left[\frac{a}{ab} \right] \left[\frac{abc}{c} \right]$$

THE BROAD IDEA

We have a domino with the starting state of your algorithm on the top:

$$\left[\frac{\#line = 1, x = 2, y = 10}{\#} \right]$$

For every way there is to progress through the program, you have a domino:

$$\left[\frac{\#line = 2, x = 3, y = 10}{line = 1, x = 2, y = 10\#} \right] \text{ or } \left[\frac{\#line = 7, x = 3, y = 10}{line = 2, x = 3, y = 10\#} \right]$$

And if you have a return statement at a specific state, you would have the following domino:

$$\left[\frac{\#}{line = 42, x = 270, y = 475\#} \right]$$

XKCD #1163: DEBUGGER

It can take a site a while to figure out there's a problem with their 'report a bug' form.

