We saw that the real numbers between 0 and 1 are uncountable.

Suppose, for the sake of contradiction, that there is a list of them:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_1</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f_2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>f_3</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>f_4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

For every \( n \) \( \in \mathbb{N} \):
- If \( f_n(n) = 5 \), set \( D(n) = 1 \)
- If \( f_n(n) \neq 5 \), set \( D(n) = 5 \)

We have \( D \neq f_n \) for any \( n \) and the list is incomplete.

⇒ (\( f \colon \mathbb{N} \to \{0, 1 \} \)) is not countable

We have seen that:
- [last time] The set of all (Java) programs is countable
- The set of all functions \( f : \mathbb{N} \to \{0, ..., 9\} \) is not countable

So: There must be some function \( f \colon \mathbb{N} \to \{0, ..., 9\} \) that is not computable by any program!
Students should write a Java program that:
- Prints "Hello" to the console
- Eventually exits

Gradelt, Pracitclt, etc. need to grade the students.

How do we write that grading program?

What does this program do?

```java
public static void collatz(n) {
    if (n == 1) {
        return 1;
    } else if (n % 2 == 0) {
        return collatz(n/2);
    } else {
        return collatz(3n + 1);
    }
}
```

follow up question #2

What does this program do?

- on n=5?
- on n=10000000000000000001?

some notation

We're going to be talking about Java code.

CODE(P) will mean "the code of the program P"

So, consider the following function:

```java
public String P(String x) {
    return new String(Arrays.sort(x.toCharArray()));
}
```

What is P(CODE(P))?

"(((())..AACPSSaaabceggghiiiilnnnnnooprrrrrrrrrrrsssttttttuuwxxyy)"
the Halting problem

Given: - CODE(P) for any program P
- input x

Output: true if P halts on input x
false if P does not halt on input x

It turns out that it isn’t possible to write a program that solves the Halting Problem.

proof by contradiction

• Suppose that H is a Java program that solves the Halting problem. Then we can write this program:

```
public static void D(x) {
    if (H(x,x) == true) {
        while (true); /* don't halt */
    } else {
        return; /* halt */
    }
}
```

• Does D(CODE(D)) halt?

\[H\] solves the halting problem implies that
\[H(CODE(D),x)\] is true iff \[D(x)\] halts, \[H(CODE(D),x)\] is false iff not

Suppose \[D(CODE(D))\] halts.
Then, we must be in the second case of the if.
So, \[H(CODE(D), CODE(D))\] is false
Which means \[D(CODE(D))\] doesn’t halt

Suppose \[D(CODE(D))\] doesn’t halt.
Then, we must be in the first case of the if.
So, \[H(CODE(D), CODE(D))\] is true.
Which means \[D(CODE(D))\] halts.

Contradiction!

Suppose \[D(CODE(D))\] halts.
Then, we must be in the second case of the if.
So, \[H(CODE(D), CODE(D))\] is false
Which means \[D(CODE(D))\] doesn’t halt

Suppose \[D(CODE(D))\] doesn’t halt.
Then, we must be in the first case of the if.
So, \[H(CODE(D), CODE(D))\] is true.
Which means \[D(CODE(D))\] halts.

Contradiction!
We proved that there is no computer program that can solve the Halting Problem.
- There was nothing special about Java* [Church-Turing thesis]
- This tells us that there is no compiler that can check our programs and guarantee to find any infinite loops they might have.

Some possible inputs:

<table>
<thead>
<tr>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_4</th>
<th>P_5</th>
<th>P_6</th>
<th>P_7</th>
<th>P_8</th>
<th>P_9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(P,x) entry is 1 if program P halts on input x and 0 if it runs forever

Rice's theorem

Not every problem on programs is undecidable!
Which of these is decidable?
- Input CODE(P) and x
  Output: true if P prints "ERROR" on input x after less than 100 steps
  false otherwise
- Input CODE(P) and x
  Output: true if P prints "ERROR" on input x after more than 100 steps
  false otherwise

Compilers Suck Theorem (informal):
Any non-trivial property the input-output behavior of Java programs is undecidable.

Reductions

- Can use undecidability of the halting problem to show that other problems are undecidable.
- For instance: Given two programs P and Q, is it true that P(x) = Q(x) for every input x?

Rice's theorem

What's next?


The final exam is Monday, Jun 8, 2015, 2:30-4:20 p.m. in MLR 301.
Notes: One page of notes allowed, front and back.
Review sessions:
- Saturday, June 6th, 2015: 1pm in EEB 105 (James)
- Sunday, June 7th, 2015: 2pm in EEB 105 (TAs)

And then... summer!