Fall 2015
Lecture 28: The halting problem and undecidability
last time

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>$r_1$</td>
<td>0.51000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r_2$</td>
<td>0.33333</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r_3$</td>
<td>0.14258</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r_4$</td>
<td>0.14151</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Flipping rule:**
- If digit is 5, make it 1.
- If digit is not 5, make it 5.

For every $n \geq 1$:

$r_n \neq 0.x_1x_2x_3x_4x_5\ldots$

because the numbers differ on the $n$th digit!

So the list is incomplete, which is a contradiction.

Thus the real numbers between 0 and 1 are uncountable.
the set of all functions $f : \mathbb{N} \rightarrow \{0, \ldots, 9\}$ is uncountable

Supposed listing of all the functions:

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>5</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>
<td>...</td>
</tr>
<tr>
<td>$f_2$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$f_4$</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$f_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$f_6$</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>$f_7$</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$f_8$</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
the set of all functions $f : \mathbb{N} \rightarrow \{0, \ldots, 9\}$ is uncountable

Supposed listing of all the functions:

<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>5 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_2$</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$f_4$</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$f_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$f_6$</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_7$</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>$f_8$</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Flipping rule:

- If $f_n(n) = 5$, set $D(n) = 1$
- If $f_n(n) \neq 5$, set $D(n) = 5$
the set of all functions $f : \mathbb{N} \rightarrow \{0, \ldots, 9\}$ is uncountable

Supposed listing of all the functions:

<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>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_2$</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$f_4$</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$f_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$f_6$</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_7$</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Flipping rule:
- If $f_n(n) = 5$, set $D(n) = 1$
- If $f_n(n) \neq 5$, set $D(n) = 5$

For all $n$, we have $D(n) \neq f_n(n)$. Therefore $D \neq f_n$ for any $n$ and the list is incomplete!

$\Rightarrow \{f \mid f : \mathbb{N} \rightarrow \{0,1, \ldots, 9\}\}$ is not countable
We have seen that:

- [last time] The set of all (Java) programs is countable
- The set of all functions \( f : \mathbb{N} \rightarrow \{0, \ldots, 9\} \) is not countable

So: There must be some function \( f : \mathbb{N} \rightarrow \{0, \ldots, 9\} \) that is not computable by any Java program!
recall our language picture

- Regular
  - 0*
    - DFA
    - NFA
    - Regex
- Context-Free
  - Binary Palindromes
- All
- Java

\{001, 10, 12\}
Students should write a Java program that:

- Prints “Hello” to the console
- Eventually exits

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

How do we write that grading program?
What does this program do?

```c
___(___,___,____){___/__ <=1?___(___,___+1,___
):!(___%___)?(___,___+1,0):___%___ ==___ / 
___&&!___?(__)?:___%___ >1&&___%___ <___/___?___(___,1+
____,_____+!(___/___%(___%___)):____<____ * ___
?___(___+1,____):0;}main(){(100,0,0);}"
public static int collatz(n) {
    if (n == 1) {
        return 1;
    }
    if (n % 2 == 0) {
        return collatz(n/2)
    } else {
        return collatz(3n + 1)
    }
}

What does this program do?

... on n=5?
... on n=10000000000000000001?
a cse 141 assignment

Students should write a Java program that:

– Prints “Hello” to the console
– Eventually exits

Gradelt, Practicelt, etc. need to grade the students. How do we write that grading program?
We’re going to be talking about *Java code*.

\[ \text{CODE}(P) \] 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 \( P(\text{CODE}(P)) \)?

“((()))..;AACPSSaaabceeggghiiiiIlnnnnnooprrrrrrrrrrrrssssttttttttluuwxxyy{}”
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
        or CODE(P) is invalid

It turns out that it isn’t possible to write a program that solves the Halting Problem.
• Suppose that $H$ is a Java program that solves the Halting problem. Then we can write this program:

```java
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

\( H(\text{CODE}(D), x) \) is true iff \( D(x) \) halts, \( H(\text{CODE}(D), x) \) is false iff not

---

**Does \( D(\text{CODE}(D)) \) halt?**

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

Does \( D(\text{CODE}(D)) \) halt?
H solves the halting problem implies that
$H(\text{CODE}(D),x)$ is true iff $D(x)$ halts, $H(\text{CODE}(D),x)$ is false iff not

Suppose $D(\text{CODE}(D))$ halts.
Then, we must be in the second case of the if.
So, $H(\text{CODE}(D), \text{CODE}(D))$ is false
Which means $D(\text{CODE}(D))$ doesn’t halt
**Does \( D(\text{CODE}(D)) \) halt?**

**Public static void** \( D(x) \) {  
    if (\( H(x,x) \) == true) {  
        while (true); /* don’t halt */  
    }  
    else {  
        return; /* halt */  
    }  
}

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

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

Suppose \( D(\text{CODE}(D)) \) doesn’t halt.  
Then, we must be in the first case of the if.  
So, \( H(\text{CODE}(D), \text{CODE}(D)) \) is true.  
Which means \( D(\text{CODE}(D)) \) halts.
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!
• 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.
### connection to diagonalization

<table>
<thead>
<tr>
<th>programs P</th>
<th>[&lt;P_1&gt;]</th>
<th>[&lt;P_2&gt;]</th>
<th>[&lt;P_3&gt;]</th>
<th>[&lt;P_4&gt;]</th>
<th>[&lt;P_5&gt;]</th>
<th>[&lt;P_6&gt;]</th>
<th>[\ldots]</th>
<th>Some possible inputs x</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P_2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P_3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P_5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P_6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P_7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P_9</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Programs P</th>
<th>&lt;P₁&gt;</th>
<th>&lt;P₂&gt;</th>
<th>&lt;P₃&gt;</th>
<th>&lt;P₄&gt;</th>
<th>&lt;P₅&gt;</th>
<th>&lt;P₆&gt;</th>
<th>....</th>
<th>Some possible inputs x</th>
</tr>
</thead>
</table>
| P₁         | 0₁   | 1₁   | 1₁   | 0₁   | 1₁   | 1₁   | 0₀   | 0₀ 0₁ 1₁ 1₁ 1₁ 1₁ 0₁ 0₁ ...
| P₂         | 1₁   | 1₀   | 0₁   | 0₁   | 1₀   | 1₁   | 0₁   | 1₁ 1₁ 0₁ 1₁ 0₁ 1₁ 0₁ 1₁ ...
| P₃         | 1₀   | 1₀   | 0₁   | 0₀   | 1₀   | 0₀   | 0₀   | 0₀ 0₀ 0₀ 0₀ 0₀ 0₀ 0₀ 0₀ 1₁ ...
| P₄         | 0₁   | 1₁   | 0₁   | 1₀   | 1₁   | 0₁   | 1₁   | 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ ...
| P₅         | 0₁   | 1₁   | 1₀   | 1₀   | 1₁   | 0₁   | 0₀   | 0₀ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ ...
| P₆         | 1₁   | 0₁   | 0₀   | 0₁   | 0₁   | 1₀   | 1₁   | 0₁ 1₁ 0₁ 1₁ 0₁ 1₁ 1₁ 1₁ 0₁ ...
| P₇         | 1₀   | 1₁   | 0₁   | 0₁   | 0₀   | 0₁   | 0₀   | 0₀ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ 0₁ ...
| P₈         | 0₁   | 1₁   | 1₁   | 0₁   | 1₀   | 1₁   | 0₁   | 1₀ 1₀ 0₁ 1₀ 0₁ 1₀ 0₁ 1₀ 0₁ ...
| P₉         |       |      |      |      |      |      |      |                      ...
|           |      |      |      |      |      |      |      |                      ...
|           |      |      |      |      |      |      |      |                      ...

(P,x) entry is 1 if program P halts on input x and 0 if it runs forever
- Can use undecidability of the halting problem to show that other problems are undecidable.

- For instance:

  \[ \text{EQUIV}(P, Q) : \begin{cases} 
  \text{True} & \text{if } P(x) = Q(x) \text{ for every input } x \\
  \text{False} & \text{otherwise} 
  \end{cases} \]
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.