Lecture 28: The halting problem and undecidability
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>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_2$</td>
<td>0.3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$r_3$</td>
<td>0.1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$r_4$</td>
<td>0.1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

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

For every $n \geq 1$:
$$r_n = 0.x_{11}x_{22}x_{33}x_{44}x_{55} \ldots$$

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 $f : \mathbb{N}^+ \to \{0, \ldots, 9\}$ 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>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</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>4</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>5</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>2</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>2</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>4</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 \( f : \mathbb{N} \rightarrow \{0, ..., 9\} \) 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>
<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 $f : \mathbb{N} \to \{0, \ldots, 9\}$ 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>$\mathbf{1}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_2$</td>
<td>3</td>
<td>3</td>
<td>$\mathbf{5}$</td>
<td>3</td>
</tr>
<tr>
<td>$f_3$</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>$\mathbf{5}$</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!

$\Rightarrow \{ f \mid f : \mathbb{N} \to \{0,1, \ldots, 9\} \}$ 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 program!
recall our language picture

- All
- Java
- Context-Free
  - Binary Palindromes
- Regular
  - $0^*$
- Finite
  - $\{001, 10, 12\}$
- DFA
- NFA
- Regex

Java

Finite

Regular

Context-Free

All
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?

```
_(_,___,____){___/__ <=1?(_(_,___ +1,____):
_):(!(___%=___)?(_,___ +1,0):___%___ ==___ /___&&!___?(printf("%d\t",___/___),(_(_,___ +1,0))):___%___ >1&&___%___ <___/__?(_,1+___,____ +!(___/__%(___%___))):___ <___*___?
(_,___ +1,____):0;}main(){(_100,0,0);}
public static void 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=1000000000000000001?
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*.

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))?  

“((()))..;AACPSSaaabceeggghiiiiInnnnnnooprrrrrrrrrrrrsssttttttttuuwxxyy{}”
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.
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
\[ H(\text{CODE}(D), x) \text{ is true iff } D(x) \text{ halts, } H(\text{CODE}(D), x) \text{ is false iff not} \]
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?

\( H \) 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.
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

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.

\[
\begin{align*}
\text{public static void } & D(x) \{ \\
& \quad \text{if } (H(x,x) == \text{true}) \{ \\
& \quad \quad \text{while } (\text{true}); /* \text{don't halt} */ \\
& \quad \} \\
& \quad \text{else } \{ \\
& \quad \quad \text{return}; /* \text{halt} */ \\
& \quad \}
\end{align*}
\]
• 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>....</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>1</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>1</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>1</td>
<td>1</td>
<td>1</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>1</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>
<tr>
<td>.</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>$&lt;P_1&gt;$</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>....</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>1</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>0</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>0</td>
<td>1</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>0</td>
<td>1</td>
<td>1</td>
<td>0</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.
- Can use undecidability of the halting problem to show that other problems are undecidable.

- For instance:

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

\[
\text{EQUIV}(P, Q_0)
\]
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
  **DECIDABLE**

• Input CODE(P) and x
  Output: true if P prints “ERROR” on input x after more than 100 steps
  false otherwise
  **UNDECIDABLE**

Compilers Suck Theorem (informal):
Any “non-trivial” property the input-output behavior of Java programs is undecidable.
foundations I, complete (almost)

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!