CSE 311: Foundations of Computing

Lecture 27: Undecidability

```
DEFINE DOES IT HALT(PROGRAM):
{
    RETURN TRUE;
}
```

THE BIG PICTURE SOLUTION TO THE HALTING PROBLEM
A set $S$ is **countable** iff we can order the elements of $S$ as $S = \{x_1, x_2, x_3, \ldots \}$

**Countable sets:**

- $\mathbb{N}$ - the natural numbers
- $\mathbb{Z}$ - the integers
- $\mathbb{Q}$ - the rationals
- $\Sigma^*$ - the strings over any finite $\Sigma$
- The set of all Java programs

*Shown by “dovetailing”*
Theorem [Cantor]:
The set of real numbers between 0 and 1 is \textit{not} countable.

Proof using “diagonalization”.

Last time: Not every set is countable
Last time: Proof that \([0,1)\) is not countable

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

| r_1  | 0.5 1 0 0 0 |
| r_2  | 0.3 3 5 3 3 |
| r_3  | 0.1 4 2 5 8 |
| r_4  | 0.1 4 1 5 |

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

For every \(n \geq 1\):
\[
r_n \neq d = 0.\hat{x}_{11}\hat{x}_{22}\hat{x}_{33}\hat{x}_{44}\hat{x}_{55} \ldots
\]
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 not countable: “uncountable”
A note on this proof

- The set of rational numbers in [0,1) also have decimal representations like this
  - The only difference is that rational numbers always have repeating decimals in their expansions 0.33333... or .25000000...
  - \( \overline{0.142857142857} \)

- So why wouldn’t the same proof show that this set of rational numbers is uncountable?
  - Given any listing (even one that is good like the dovetailing listing) we could create the flipped diagonal number \( d \) as before
  - However, \( d \) would not have a repeating decimal expansion and so wouldn’t be a rational #
    - It would not be a “missing” number, so no contradiction.
Last time:
The set of all functions $f : \mathbb{N} \to \{0, \ldots, 9\}$ is uncountable

Supposed listing of all the functions:

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  
|---|---|---|---|---|---|---|---|---|---|---|
| $f_1$ | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   
| $f_2$ | 3 | 3 | 5 | 3 | 3 | 3 | 3 | 3 | 3 |   
| $f_3$ | 1 | 4 | 2 | 5 | 8 | 5 | 7 | 1 | 4 |   
| $f_4$ | 1 | 4 | 1 | 5 | 1 | 9 | 2 | 6 | 5 |   
| $f_5$ | 1 | 2 | 1 | 2 | 2 | 5 | 1 | 2 | 2 |   
| $f_6$ | 2 | 5 | 0 | 0 | 0 | 0 | 0 | 5 | 0 |   
| $f_7$ | 7 | 1 | 8 | 2 | 8 | 1 | 8 | 2 | 5 |   

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:

– The set of all (Java) programs is countable
– The set of all functions \( f : \mathbb{N} \to \{0, \ldots, 9\} \) is not countable

So: There must be some function \( f : \mathbb{N} \to \{0, \ldots, 9\} \) that is not computable by any program!

Interesting... maybe.

Can we come up with an explicit function that is uncomputable?
A “Simple” Program

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

What does this program do?

... on n=11?

... on n=10000000000000000001?
A “Simple” Program

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

What does this program do?

... on n=11?

... on n=1000000000000000001?

Nobody knows whether or not this program halts on all inputs!

Trying to solve this has been called a “mathematical disease”.

Recall our language picture

- **All**
- **Java**
- **Context-Free**
  - **Binary Palindromes**
- **Regular**
  - **DFA**
  - **NFA**
  - **Regex**
- **Finite**
  - `{001, 10, 12}`

Diagram shows the hierarchy of language classes with DFA, NFA, and Regex at the bottom, followed by regular languages, context-free languages, and finally all languages.
Some Notation

We’re going to be talking about Java code.

\textit{CODE}(\textit{P}) \text{ will mean “the code of the program } \textit{P} \text{”}

So, consider the following function:

\begin{verbatim}
public String P(String x) {
    return new String(Arrays.sort(x.toCharArray()));
}
\end{verbatim}

What is \textbf{P(CODE(P))}?

“((((()))..;AACPSSaaabceegghiiiiIlnnnnnoooprrrrrrrrrrsssttttttuuwwxxyy}”
The Halting Problem

CODE(P) means “the code of the program P”

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
Undecidability of the Halting Problem

CODE(P) means “the code of the program P”

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

Theorem [Turing]: There is no 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:

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

• Does $D(CODE(D))$ halt?
Does $D(CODE(D))$ 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 $D(x)$ halts.

Suppose that $D(\text{CODE}(D))$ halts.

Note: Even though the program $D$ has a while(true), that doesn’t mean that the program $D$ actually goes into an infinite loop on input $x$, which is what $H$ has to determine.
H solves the halting problem implies that
\( H(\text{CODE}(D),x) \) is \textbf{true} iff \( D(x) \) halts, \( H(\text{CODE}(D),x) \) is \textbf{false} iff not

Suppose that \( D(\text{CODE}(D)) \) \textbf{halts}.
Then, by definition of \( H \) it must be that

\( H(\text{CODE}(D), \text{CODE}(D)) \) is \textbf{true}.

Which by the definition of \( D \) means \( D(\text{CODE}(D)) \) \textbf{doesn’t halt}. 

Suppose that \( D(\text{CODE}(D)) \) \textbf{doesn’t halt}.

\[ \therefore H(\text{CODE}(D), \text{CODE}(D)) \text{ is false by} \]

\[ \therefore D(x) \text{ halt for } x = \text{CODE}(D) \text{ i.e} \]

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

**H** solves the halting problem implies that

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

Suppose that \( D(CODE(D)) \) halts.
Then, by definition of \( H \) it must be that

\[ H(CODE(D), CODE(D)) \text{ is } \text{true} \]

Which by the definition of \( D \) means \( D(CODE(D)) \) doesn’t halt

Suppose that \( D(CODE(D)) \) doesn’t halt.
Then, by definition of \( H \) it must be that

\[ H(CODE(D), CODE(D)) \text{ is } \text{false} \]

Which by the definition of \( D \) means \( D(CODE(D)) \) halts

```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 $H(CODE(D), x)$ is **true** iff $D(x)$ halts, $H(CODE(D), x)$ is **false** iff not $D(x)$ halts.

Suppose that $D(CODE(D))$ halts. Then, by definition of $H$ it must be that $H(CODE(D), CODE(D))$ is **true**. Which by the definition of $D$ means $D(CODE(D))$ doesn’t halt.

Suppose that $D(CODE(D))$ doesn’t halt. Then, by definition of $H$ it must be that $H(CODE(D), CODE(D))$ is **false**. Which by the definition of $D$ means $D(CODE(D))$ halts.

The **ONLY** assumption was that the program $H$ exists so that assumption must have been false. 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.
Where did the idea for creating D come from?

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

D halts on input code(P) iff \( H(code(P),code(P)) \) doesn’t halt iff P doesn’t halt on input code(P)

Therefore for any program P, D differs from P on input code(P)
### Connection to diagonalization

This listing of all programs really does exist since the set of all Java programs is countable.

The goal of this “diagonal” argument is not to show that the listing is incomplete but rather to show that a “flipped” diagonal element is not in the listing.

---

<table>
<thead>
<tr>
<th>All 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>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₈</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₉</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>
</tr>
</tbody>
</table>

Write \(<P>\) for CODE(P)
Connection to diagonalization

<table>
<thead>
<tr>
<th>All 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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
<tr>
<td>P_9</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.

Write \(<P>\) for \(\text{CODE}(P)\).
### Connection to diagonalization

<table>
<thead>
<tr>
<th>All 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>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0$^1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>1</td>
<td>1$^0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>1</td>
<td>0</td>
<td>1$^0$</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0$^1$</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>$P_5$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1$^0$</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$^1$</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$^1$</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>
</tr>
<tr>
<td>$P_9$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$(P,x)$ entry is 1 if program $P$ halts on input $x$ and 0 if it runs forever</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Halting Problem isn’t the only hard problem

- Can use the fact that the Halting Problem is undecidable to show that other problems are undecidable

General method:

Prove that if there were a program deciding $B$ then there would be a way to build a program deciding the Halting Problem.

“$B$ decidable $\rightarrow$ Halting Problem decidable”

Contrapositive:

“Halting Problem undecidable $\rightarrow$ $B$ undecidable”

Therefore $B$ is undecidable
A CSE 141 assignment

Students should write a Java program that:

- Prints “Hello” to the console
- Eventually exits

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

How do we write that grading program?

WE CAN’T: THIS IS IMPOSSIBLE!
A related undecidable problem

• **HelloWorldTesting Problem:**
  – Input: CODE(Q) and x
  – Output:
    
    True if Q outputs “HELLO WORLD” on input x
    False if Q does not output “HELLO WORLD” on input x

• **Theorem:** The HelloWorldTesting Problem is undecidable.

• **Proof idea:** Show that if there is a program T to decide HelloWorldTesting then there is a program H to decide the Halting Problem for code(P) and x.
A related undecidable problem

• Suppose there is a program $T$ that solves the HelloWorldTesting problem. Define program $H$ that takes input CODE($P$) and $x$ and does the following:
  
  – Creates CODE($Q$) from CODE($P$) by
    
    (1) removing all output statements from CODE($P$), and
    (2) adding a System.out.println(“HELLO WORLD”) immediately before any spot where $P$ could halt

  Then runs $T$ on input CODE($Q$) and $x$.

• If $P$ halts on input $x$ then CODE($Q$) prints HELLO WORLD and halts and so $H$ outputs true (because $T$ outputs true on input CODE($Q$))
• If $P$ doesn’t halt on input $x$ then CODE($Q$) won’t print anything since we removed any other print statement from CODE($Q$) so $H$ outputs false

We know that such an $H$ cannot exist. Therefore $T$ cannot exist.