Review:

• We have seen that
  – The set of all (Java) programs is countable
Supposed listing of functions: $\mathbb{N}_+ \rightarrow \{0,1,\ldots,9\}$

<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>0</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>0</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>0</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>0</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>0</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>
**Flipped Diagonal Function** \( D: \mathbb{N}_+ \rightarrow \{0,1,\ldots,9\} \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>f_1</strong></td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>f_2</strong></td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>f_3</strong></td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td><strong>f_4</strong></td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td><strong>f_5</strong></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>f_6</strong></td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>f_7</strong></td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td><strong>f_8</strong></td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

**Flipping Rule:**

- If \( f_n(n) \) is 5, make \( D(n) = 1 \)
- If \( f_n(n) \) is not 5, make \( D(n) = 5 \)
Flipped Diagonal Function $D: \mathbb{N}_+ \rightarrow \{0,1,\ldots,9\}$

For all $n$, we have $D \neq f_n$ since $D(n) \neq f_n(n)$.

$\Rightarrow$ list was incomplete

$\Rightarrow \{f \mid f: \mathbb{N}_+ \rightarrow \{0,1,\ldots,9\}\}$ is not countable.
Non-computable functions

• We have seen that
  – The set of all (Java) programs is countable
  – The set of all functions \( f : \mathbb{N} \rightarrow \{0,1,...,9\} \) is not countable

• So...
  – There must be some function \( f : \mathbb{N} \rightarrow \{0,1,...,9\} \) that is not computable by any program!
Back to the Halting Problem

• Suppose that there is a program $H$ that computes the answer to the Halting Problem

• We will build a table with a row for each program (just like we did for uncountability of reals)

• If the supposed program $H$ exists then the $D$ program we constructed as before will exist and so have a row in the table

• We will see that $D$ must have entries like the “flipped diagonal”
  – $D$ can’t possibly be in the table.
  – Only assumption was that $H$ exists. That must be false.
## Some possible inputs \( x \)

<table>
<thead>
<tr>
<th>( P )</th>
<th>( C(P_1) )</th>
<th>( C(P_2) )</th>
<th>( C(P_3) )</th>
<th>( C(P_4) )</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>
</tr>
<tr>
<td>( P_2 )</td>
<td>1</td>
<td>1</td>
<td>0</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>
</tr>
<tr>
<td>( P_4 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>
</tr>
<tr>
<td>( P_6 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>( P_8 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( P_9 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( P )</td>
<td>( x )</td>
<td>( P )</td>
<td>( x )</td>
<td>( P )</td>
<td>( x )</td>
</tr>
</tbody>
</table>

\[ (P, x) \text{ entry is 1 if program } P \text{ halts on input } x \]

\[ \text{ and 0 if it runs forever} \]

\( C(P) \) is shorthand for \( \text{CODE}(P) \).
### Some possible inputs $x$

<table>
<thead>
<tr>
<th>Programs $P_i$</th>
<th>$C(P_1)$</th>
<th>$C(P_2)$</th>
<th>$C(P_3)$</th>
<th>$C(P_4)$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>
</tr>
<tr>
<td>$P_3$</td>
<td>1</td>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>$P_5$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_6$</td>
<td>1</td>
<td>1</td>
<td>0</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>
</tr>
<tr>
<td>$P_8$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$P_9$</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.
recall: code for $D$ assuming subroutine $H$ that solves the halting problem

• Function $D(x)$:

  if ($H(x,x)==true$) {
    while (true); /* loop forever */
  }

  else {
    return; /* do nothing and halt */
  }
<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>
</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>0</td>
<td>1</td>
<td>0</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>0</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>0</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>
</tr>
<tr>
<td>(P_8)</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_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.

Some possible inputs \(x\):

\(<P_1>\) <\(P_2>\) <\(P_3>\) <\(P_4>\) <\(P_5>\) <\(P_6>\) ..... 

\(D\) behaves like flipped diagonal.
recall: code for \textbf{D} assuming subroutine \textbf{H} that solves the halting problem

• Function \textbf{D}(x):

\begin{verbatim}
if (H(x,x)==true) {
    while (true); /* loop forever */
}
else {
    return; /* do nothing and halt */
}
\end{verbatim}

• If \textbf{D} existed it would have a row different from every row of the table

– \textbf{D} can’t be a program so \textbf{H} cannot exist!
Diagram: Using Hypothetical Program H to build D

CODE(P) x

true false

```
String sql = getStatement();
resultSet = statement.executeQuery(sql);
if (resultSet.next()) {
    result = true;
    setstoreId(resultSet.getInt("storeId"));
    storeDescription = resultSet.getString("storeDescription");
    storeType = resultSet.getString("storeType");
}
```
Diagram: Using Hypothetical Program $H$ to build $D$

```
while(true);
/*loop forever*/
return; /*halt*/
```
More than just halting is hard

• We showed
  – if the hypothetical program $H$ deciding the Halting Problem existed, then we could use it to build a program $D$ that cannot possibly exist
  – Since $D$ doesn’t exist, program $H$ cannot exist

• We will use similar approach to show that other important problems are hard
  – if there is a hypothetical program $A$ solving one of these problems, then we could use it to build a program $H$ solving the Halting Problem
  – Since $H$ doesn’t exist, $A$ cannot exist
But first another hard halting-related problem

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

HaltsNoInput Problem:

**Given:** - CODE(Q) for any program Q

**Output:**  
true if Q halts without reading any input  
false if Q reads input or runs forever.
Key idea: Hardcoding an Input

INPUT is “potato”

public String P(String y) {
    return new String(
        Arrays.sort(y.toCharArray());
    )
}  

public String Q() {
    return new String(
        Arrays.sort("potato".toCharArray());
    )
}

Q: Version of P with “hardcoded” input:

Q() behaves the same as P("potato"), except that it doesn’t read any input.

Can write a program Hardcoder that, given CODE(P) and an input string x, produces CODE(Q)
Suppose that hypothetical program $A$ solves $\text{HaltsNoInput}$ problem. Combine with Hardcoder:

- If $A$ existed then $H'$ would solve the Halting Problem: Impossible

$H'$ outputs true on inputs $\text{CODE}(P)$ and $x$ iff $A$ outputs true iff $Q()$ reads no input and (always) halts iff $P(x)$ halts
Some notation: Decision problems as sets

Every decision problem can be written as asking about membership in a set.

If a program “decides” a set, then it must output true on all inputs in the set and false on all inputs not in the set.

$\text{Halt} = \{(\text{CODE}(P), x) : P \text{ is a program that halts on input } x\}$

$\text{HaltsNoInput} = \{\text{CODE}(Q) : Q \text{ is a program that halts without reading any input}\}$
Convenient pictures

- Rather than continue to come up with more names like H and A for our hypothetical programs...
- Given a decision problem SET we use the following picture to denote any hypothetical program that solves decision problem SET with ANS denoting its output.
Showing **HELLO** is Undecidable

Consider the set:

**HELLO** = \{CODE(R) : R is a program that reads no input, prints “Hello”, and always halts\}

**Question:** Does Q() halt?

**Step 1:** Remove all `System.out.print`/`println` statements from CODE(Q).

**Step 2:** Append `System.out.println("Hello")` at the end of the program code.

Call the new program R

**Question:** Does R() print “Hello” and halt?

Answering with ANS would solve HaltsNoInput!
Showing **HELLO** is Undecidable (Full Proof)

Suppose for contradiction that HELLO is decidable. Then, there is a program HLO(CODE(R)) that returns true when CODE(R) ∈ HELLO and false otherwise.

Consider an arbitrary program Q. We will now construct a program that decides if Q ∈ HaltsNoInput. Define a new program R by applying the following transformations to Q’s code:

1. Step 1: Remove all System.out.print/println statements from CODE(Q).
2. Step 2: Append System.out.println(“Hello”) at the end of the code (and before all return statements).

Then, we can create the following program:

```
HNI(Q) { return HLO(R); }
```

We claim HNI solves HaltsNoInput. Suppose HNI(Q) = true. Then, it should be the case that Q halts. Note that HLO(R) must be true by definition of HNI. So, R ∈ HELLO; it follows that R reads no input, prints “Hello”, and halts. Note that Q also reads no input. Furthermore, Step 2 ensures that Q prints “Hello”, and Step 1 ensures it prints nothing else. It follows that when R ∈ HELLO, R must halt. But R halts exactly when Q halts, because we didn’t change anything that effects whether it halts or not.

We make a very similar argument for when HNI(Q) = false. Then, since we’ve solved HaltsNoInput, which is undecidable, we’ve reached a contradiction. So, it follows that HELLO is undecidable as well.
A Decision Problem We Can Solve

\[ \text{REGEQUIV} = \{(R_1, R_2) : R_1 \text{ and } R_2 \text{ are equivalent regexps}\} \]

In this case the hypothetical program does exist:
Convert both to NFAs then DFAs, minimize and compare

\[ 00^* (10)^* 11^* \quad 0^* (01)^* 1^* \quad 00^* (10)^* 11^* \quad 0^* (01)^* 01^* 1 \]

\[ \rightarrow \text{False} \quad \text{True} \]
Showing **EQUIV** is Undecidable

Consider the set:

\[ \text{EQUIV} = \{ (\text{CODE}(P), \text{CODE}(R)) : P, R \text{ are programs, } P(x) = R(x) \text{ for all inputs } x \} \]

**Step 1:** Construct P:

```java
public static boolean P()
{return true;}
```

**Step 2:** Construct R:

- **Step a:** Replace return type of Q with `boolean`
- **Step b:** Replace all return values with `true`
- **Step c:** Add "return true;" to the end of the program

Call this program R

**Question:** Does Q() halt?

**Question:** Are P and R Equivalent?

Answering with ANS would solve HaltsNoInput!
Pitfalls

• Not every problem on programs is undecidable! Which of these is decidable?

• Input \text{CODE}(P) \text{ and } x
  Output: true if $P$ prints “ERROR” on input $x$
  after less than 100 steps
  false otherwise

• Input \text{CODE}(P) \text{ and } x
  Output: true if $P$ prints “ERROR” on input $x$
  after more than 100 steps
  false otherwise