Chapter 7

Undecidable Problems

Is there a problem for which no algorithm can produce a correct answer for every input?
Definition (bijection)

- A function $f : A \rightarrow B$ is *one-to-one* if $f$ never assigns the same value to two different elements of its domain.

- It is *onto* if its range is the entire set $B$.

- A function from $A$ to $B$ that is both one-to-one and onto is called a bijection from $A$ to $B$. 
Definition A set $A$ is *countably infinite* (the same size as $\mathbb{N}$) if there is a bijection $f : \mathbb{N} \rightarrow A$, or a list $a_0, a_1, \ldots$ of elements of $A$ such that every element of $A$ appears exactly once in the list. $A$ is countable if $A$ is either finite or countably infinite.

Theorem Every infinite set has a countably infinite subset, and every subset of a countable set is countable.

Question: Is $\mathbb{N}$ countable? Answer: Yes. A corresponding bijection from $\mathbb{N}$ to $\mathbb{N}$ is $f(x) = x$. 
Example
The set $\mathbb{N} \times \mathbb{N}$ is countable.

We can describe the set by drawing a two-dimensional array:

$\begin{array}{cccccc}
(0,0) & (0,1) & (0,2) & (0,3) & \ldots \\
(1,0) & (1,1) & (1,2) & (1,3) & \ldots \\
(2,0) & (2,1) & (2,2) & (2,3) & \ldots \\
(3,0) & (3,1) & (3,2) & (3,3) & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}$

How to count it? In other words, how to find a bijection from $\mathbb{N}$ to $\mathbb{N} \times \mathbb{N}$?
Examples

- The countable union of countable sets is countable. If $S_i$ is countable for every $i \in \mathbb{N}$ then

$$S = \bigcup_{i=1}^{\infty} S_i$$

is countable.

Proof: This is a generalization of the previous example. This time the ordered pair $(i, j)$ in the figure stands for the $j$th element of $S_i$, so that the $i$th row of the two-dimensional array represents the elements of $S_i$. 
Examples

• For a finite alphabet $\Sigma$ (such as $\{a, b\}$), the set $\Sigma^*$ of all strings over $\Sigma$ is countable.

Proof: This follows from previous example, because

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

and each of the sets $\Sigma^i$ is countable.

Corollary 1: Languages are countable sets
Examples

- The set of Turing machines is countable.

Let $\mathcal{T}$ represent the set of Turing machines.

- A TM $T$ can be represented by the string $e(T) \in \{0,1\}^*$, and a string can represent at most one TM.

- Therefore, the resulting function $e$ is one-to-one, and we may think of it as a bijection from $\mathcal{T}$ to a subset of $\{0,1\}^*$.

- Because $\{0,1\}^*$ is countable, every subset is, and we can conclude that $\mathcal{T}$ is countable.
Examples

- What about the set $2^\mathbb{N}$?

The set of all subsets of $\mathbb{N}$. Each element is a subset of $\mathbb{N}$.
Theorem

- The set $2^\mathbb{N}$ is uncountable.

We wish to show that there can be no list of subsets of $\mathbb{N}$ containing every subset of $\mathbb{N}$ that can be enumerated as $\mathbb{N}$. In other words, every list $A_0, A_1, A_2, \ldots$ of subsets of $\mathbb{N}$ must leave out at least one.

A diagonal argument and construction are provided by Cantor. Here is a subset, constructed from the ones in the list, that cannot possibly be in the list:

$$A = \{ i \in \mathbb{N} \mid i \notin A_i \}$$

The reason that $A$ must be different from $A_i$ for every $i$ is that $A$ and $A_i$ differ because of the number $i$, which is in one but not both of the two sets: if $i \in A_i$, then by definition of $A$, $i$ does not satisfy the defining condition of $A$, and so $i \notin A$; and if $i \notin A_i$, then (by definition of $A$) $i \in A$. 
Given $S = \{1, 2, 3, 4, 5, 6\}$ and its subsets $A_1 = \{1, 3, 4, 5\}$, $A_2 = \{2, 4, 5, 6\}$, $A_3 = \{1, 2, 3, 4\}$, and $A_4 = \{5\}$.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristic function of $A_1$</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_2$</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characteristic function of $A_3$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_4$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Flip the diagonal indicators

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristic function of $A_1$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_2$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characteristic function of $A_3$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_4$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The new set is $A_5 = \{4\}$

Wait ... maybe this subset is in the list? Let’s check what happens in this case ...
Given $S = \{1, 2, 3, 4, 5, 6\}$ and its subsets $A_1 = \{1, 3, 4, 5\}$, $A_2 = \{2, 4, 5, 6\}$, $A_3 = \{1, 2, 3, 4\}$, and $A_4 = \{5\}$.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Characteristic function of $A_1$</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_2$</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characteristic function of $A_3$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_4$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>New characteristic function</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Flip the diagonal once again. The missing subset is $\{4, 5\}$.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Characteristic function of $A_1$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_2$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characteristic function of $A_3$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Characteristic function of $A_4$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>New characteristic function</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Let us visualize the subsets, and the selection process of missing $A$

This subset contains 0, 2, 5, 9 ... each row is a characteristic (or indicator) function of $A_i$

\[\begin{array}{cccccccccc}
A_0: & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
A_1: & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
A_2: & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
A_3: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_4: & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
A_5: & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
A_6: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
A_7: & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
A_8: & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
A_9: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\ldots
\end{array}\]

we obtain the sequence corresponding to the set $A$ by reversing diagonal: 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, \ldots. Each entry we encounter as we make our way down the diagonal allows us to distinguish the set $A$ from one more of the sets $A_i$. The missing subset $A = \{2, 3, 6, 8, 9, \ldots\}$. 
Same argument works if we want to find a language that cannot be accepted by a Turing machine - in other words, a language that is different from \( L(T) \), for every Turing machine \( T \) with input alphabet \{0, 1\}, i.e.,

there are languages that are not accepted by Turing machines!

Set of TMs is countable. Set of all languages is uncountable.

List of TMs

<table>
<thead>
<tr>
<th>( A_0 )</th>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
<th>( A_6 )</th>
<th>( A_7 )</th>
<th>( A_8 )</th>
<th>( A_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</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>0</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>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>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>
Another conclusion:

There are uncountably many languages because we know that $S=\{0,1\}^*$ is equivalent to $N$ and each subset of $S$ is a language.
# Recursive and recursively enumerable languages

A TM $T$ with input alphabet $\Sigma$ ...

<table>
<thead>
<tr>
<th>accepts</th>
<th>decides</th>
</tr>
</thead>
<tbody>
<tr>
<td>a language $L \subseteq \Sigma^*$ if it accepts the strings in $L$ and no others.</td>
<td>a language $L \subseteq \Sigma^<em>$ if $T$ computes the characteristic function $\chi_L : \Sigma^</em> \to {0,1}$ that returns 1 on strings in $L$ and 0 otherwise.</td>
</tr>
</tbody>
</table>

In both cases, the issue is whether the input string is an element of $L$. However, the second approach may be more informative, because a TM accepting $L$ may not return an answer if the string is not in $L$.

*Recurrsively enumerable (RE)* languages are those that can be *accepted* by a TM.

*Recursive* languages are those that can be *decided* by a TM.

Only in the decision case there is a guaranteed answer to the question: Given a string $x$, is $x$ an element of the language?
Theorem: Every recursive language is recursively enumerable.

Theorem: If $L \subseteq \Sigma^*$ is accepted by a TM $T$ that halts on every input string, then $L$ is recursive.

Theorem: If $L_1$ and $L_2$ are both recursively enumerable languages over $\Sigma$, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable.

Theorem: If $L_1$ and $L_2$ are both recursive languages over $\Sigma$, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursive.

Theorem: If $L$ is a recursive language over $\Sigma$, then its complement is also recursive.

Theorem: If $L$ is a recursively enumerable language, and its complement is also recursively enumerable, then $L$ is recursive.
Theorem: *Not all languages are recursively enumerable. In fact, the set of languages over \{0,1\} that are not recursively enumerable is uncountable.*

Proof:
We know that \(2^\mathbb{N}\) is uncountable and we observed that because \(\{0,1\}^*\) is the same size as \(\mathbb{N}\), it follows that the set of languages over \(\{0,1\}\) is uncountable.

We know that the set of RE languages over \(\{0,1\}\) is countable (because the set of TM is countable).

If \(T\) is any countable subset of an uncountable set \(S\) then \(S-T\) is uncountable.
A Language That Can’t Be Accepted, and a Problem That Can’t Be Decided

• Definition: Let
  – $NSA = \{e(T) | T \text{ is a TM and } e(T) \notin L(T)\}$
  – $SA = \{e(T) | T \text{ is a TM and } e(T) \in L(T)\}$
    • ("non-self-accepting" and "self-accepting")

• Theorem:
  – The language $NSA$ is not recursively enumerable
  – The language $SA$ is recursively enumerable but not recursive
• The statement of the theorem says that there is no algorithm to determine whether a given string represents a TM that accepts its own encoding
  – It might seem that for a TM \( T \), deciding whether \( T \) accepts the string \( e(T) \) is particularly difficult, but this is not the right interpretation
  – All we needed for the diagonal argument was a string associated with \( T \); we chose \( e(T) \), but we could just as easily have used something else
• The more correct conclusion is that it’s hard to answer questions about TMs and the languages they accept
Reductions

- We can often solve problems by reducing them to other, simpler ones
- We will reduce one decision problem \( P_1 \) to another \( P_2 \)
- The two crucial features in a reduction \( F \) are:
  - For every instance \( I \) of \( P_1 \) we must be able to obtain an instance \( F(I) \) of \( P_2 \) algorithmically
  - The answer to \( P_2 \) for the instance \( F(I) \) must be the same as the answer to \( P_1 \) for \( I \)

For simplicity, binary
Reductions

**Definition:** Suppose $P_1$ and $P_2$ are decision problems. We say $P_1$ is reducible to $P_2$ ($P_1 \leq P_2$) if there is an algorithm that finds, for an arbitrary instance $I$ of $P_1$, an instance $F(I)$ of $P_2$ such that the two answers are the same, i.e., (the answer to $P_1$ for the instance $I$, and the answer to $P_2$ for the instance $F(I)$)

\[ P_1(I) = P_2(F(I)) \]

Idea 1: For example, I don’t know how to solve $P_1$ but I can solve $P_2$ and know how to map the instances to preserve answers. Then I can solve $P_1$
Reductions

Definition: Suppose $P_1$ and $P_2$ are decision problems. We say $P_1$ is reducible to $P_2$ ($P_1 \leq P_2$) if there is an algorithm that finds, for an arbitrary instance $I$ of $P_1$, an instance $F(I)$ of $P_2$ such that the two answers are the same, i.e., (the answer to $P_1$ for the instance $I$, and the answer to $P_2$ for the instance $F(I)$)

Idea 2: Say, $P_1$ is computationally difficult, and I don’t know the difficulty of $P_2$. If I know how to map the instances then I can state that $P_2$ is at least as difficult as $P_1$.
Example

- Problem $P_2(x, y)$ decides $x < y$ for $x, y \in \{1, \ldots, 5\}$.
- Problem $P_1(a^2, b^2)$ decides $a^2 < b^2$ for $a^2, b^2 \in \{1, \ldots, 25\}$.
- Imagine that $P_1$ is hard (or unknown how) to compute but $P_2$ is not, and let us assume that we can compute a positive $\sqrt{x^2}$.
- For each instance $I$ of $P_1$ we can compute an instance $F(I)$ of $P_2$, and the answers to $P_1$, and $P_2$ are the same.

$\rightarrow$ Now we need solve $P_2$ instead of $P_1$
Reductions

• Definition:
  – Suppose $P_1$ and $P_2$ are decision problems
    • We say $P_1$ is reducible to $P_2$ ($P_1 \leq P_2$) if there is an algorithm that finds, for an arbitrary instance $I$ of $P_1$, an instance $F(I)$ of $P_2$ such that the two answers (the answer to $P_1$ for the instance $I$, and the answer to $P_2$ for the instance $F(I)$) are the same
  – If $L_1$ and $L_2$ are languages over alphabets $\Sigma_1$ and $\Sigma_2$
    • We say $L_1$ is reducible to $L_2$ ($L_1 \leq L_2$) if there is a Turing-computable function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that for every $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$
Reductions

• Theorem:
  – Suppose $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, and $L_1 \leq L_2$
    • If $L_2$ is recursive, then $L_1$ is recursive (Proof: we can decide whether a string is in $L_1$ by using the reduction and deciding whether the resulting string is in $L_2$)
  – Suppose $P_1$ and $P_2$ are decision problems, and $P_1 \leq P_2$
    • If $P_2$ is decidable, then $P_1$ is decidable (Proof: we can decide whether an instance of $P_1$ is a yes-instance by using the reduction and deciding whether the resulting instance of $P_2$ is a yes-instance)
  – We will be interested in the contrapositive statement: If $P_1$ is undecidable, then $P_2$ is also.
• Consider two decision problems:
  – *Accepts*: Given a TM $T$ and a string $w$, is $w \in L(T)$?
  – *Halts*: Given a TM $T$ and a string $w$, does $T$ halt (either by accepting or by rejecting) on input $w$? (This is called the *halting problem*)

• **Theorem**: Both *Accepts* and *Halts* are undecidable

• For the first statement, we just need to show that *Self-accepting* $\leq$ *Accepts*
  – A reduction from *Self-accepting* to *Accepts* is $F(T) = (T, e(T))$
  – We can compute this algorithmically.

• For the second statement, we can reduce *Accepts* to *Halts* (see the book for the details). *Accepts* is undecidable; therefore, *Halts* is undecidable
• Theorem: The following five decision problems are undecidable:

1. **Accepts-Λ**: Given a TM $T$, is $\Lambda \in L(T)$?
2. **AcceptsEverything**: Given a TM $T$ with input alphabet $\Sigma$, is $L(T) = \Sigma^*$?
3. **Subset**: Given two TMs $T_1$ and $T_2$, is $L(T_1) \subseteq L(T_2)$?
4. **Equivalent**: Given two TMs $T_1$ and $T_2$, is $L(T_1) = L(T_2)$?
5. **WriteSymbol**: Given a TM $T$ and a symbol $a$ in the tape alphabet of $T$, does $T$ ever write an $a$ if it starts with an empty tape?
• \textit{WritesNonblank} problem: Given a TM $T$ with $n$ nonhalting states, does $T$ ever write a nonblank symbol on its tape, if it starts with a blank tape?

• Theorem:
  – The decision problem \textit{WritesNonblank} is decidable.

• Proof sketch:
  – An algorithm to decide \textit{WritesNonblank} is to trace $T$ for $n$ moves, or until it halts, whichever comes first
  – within $n$ moves, either it halts or it enters some nonhalting state $q$ for the second time
  – If by that time no nonblank symbol has been written, none ever will be
• It’s now clear that it’s difficult to answer questions about Turing machines and the strings they accept
• A few more undecidable problems about a TM $T$:
  1. (For some language $L$) *AcceptsL*: Given a TM $T$, is $L(T) = L$?
  2. *AcceptsSomething*: Is there at least one string in $L(T)$?
  3. *AcceptsTwoOrMore*: Does $L(T)$ have at least two elements?
  4. *AcceptsFinite*: is $L(T)$ finite?
  5. *AcceptsRecursive*: is $L(T)$ recursive?

Homework: Read Chapter 8!