Notes on Undecidability

Foundations of Computer Science

February 25, 2016

Reminder: a language $A \subseteq \Sigma^*$ is Turing-decidable (or just decidable) when there is a *total* Turing machine M that accepts A.

By the Church-Turing thesis, a language is decidable if and only if it describes a decision problem that can be implemented exactly by some computational process.

A simple counting argument shows that there has to be at least one language that is not decidable. (A language is undecidable when it is not decidable.) Intuitively, there are more languages (over alphabet Σ) than there are Turing machines (over alphabet Σ).

To make this precise, we need a definition of "size" for infinite sets (since there are infinitely many languages and infinitely many Turing machines).

Definition: Two sets A and B are *equipollent* (have the same size), written $A \approx B$, if you can match every element of A with a distinct element of B, and vice versa. Formally, $A \approx B$ when there exists a *one-to-one correspondence* (a bijection) $f : A \longrightarrow B$.

Definition: Set A is "no bigger than" set B, written $A \preceq B$, if you can match every element of A with a distinct element of B. Formally, $A \preceq B$ if there $A \approx C$ for some $C \subseteq B$, or equivalently if there exists a one-to-one function $f : A \longrightarrow B$, or equivalently if there exists an onto function $g : B \longrightarrow A$.

Definition: $A \prec B$ if $A \preceq B$ but $A \not\approx B$.

If $\mathbb{N}$ is the set of natural numbers and $2\mathbb{N}$ the set of even natural numbers, then the bijective function $f : \mathbb{N} \longrightarrow 2\mathbb{N}$ given by $f(n) = 2n$ shows that $\mathbb{N} \approx 2\mathbb{N}$.

A slightly more complex bijective function will show that $\mathbb{N} \approx \mathbb{Q}$, the set of rational numbers.

A classic argument can be used to show that $\mathbb{N} \prec \mathbb{R}$, the set of real numbers.

Rather than show the latter, here's a result that will be more useful for us.

Cantor's Theorem: For any set A, we have $A \prec \wp(A)$, where $\wp(A)$ is the set of subsets of A.

Proof: Clearly, $A \preceq \wp(A)$, by taking $f : A \longrightarrow \wp(A)$ to be $f(x) = \{x\}$.

To show that $A \not\approx \wp(A)$, we argue by contradiction. Suppose that there *were* a bijective function $f : A \longrightarrow \wp(A)$. I'll show that this assumption leads to an absurdity.

Construct the following set:

$$B = \{x \in A \mid x \notin f(x)\}$$

This is a perfectly well defined set. Since f is bijective and therefore onto, there must exist a $b \in A$ such that $f(b) = B$.

Now, does $b \in B$?

If $b \in B$, then by definition of b, $b \notin f(b)$, that is, $b \notin f(b) = B$.

If $b \notin B$, then by definition of b, $b \in f(b)$ (otherwise, b would be in B) and thus $b \in f(b) = B$.

Either way, we get an absurdity. So our assumption that there is a bijective f cannot be. Thus, $A \not\approx \wp(A)$. □

Cantor's Theorem means, in particular, that we get an infinite tower of sets of increasing infinite sizes:

$$\mathbb{N} \prec \wp(\mathbb{N}) \prec \wp(\wp(\mathbb{N})) \prec \wp(\wp(\wp(\mathbb{N}))) \prec \ldots$$

The following result will be useful, and is actually suprisingly difficult to prove.

Cantor-Bernstein Theorem: If $A \preceq B$ and $B \preceq A$, then $A \approx B$.

In fact, what we need is the following consequence of the Cantor-Bernstein Theorem: if $A \prec B$, then $B \not\preceq A$.

Finally, we can show that there must be an undecidable language. Let $T(\Sigma)$ be the set of Turing machines over alphabet Σ. We establish that

$$T(\Sigma) \prec \wp(\Sigma^*)$$

where of course $\wp(\Sigma^*)$ is the set of all languages over alphabet Σ.

First, we show that $T(\Sigma) \preceq \mathbb{N}$. For this, it suffices to encode every Turing machines M into a string of 0 and 1. Intuitively, if $M = (Q, \Sigma, \Gamma, \llcorner, \vdash, \delta, s, acc, rej)$, without loss of generality, we can take $Q = \{1, \ldots, n\}$, and if $\Gamma = \{a_1, \ldots, a_k\}$, we can encode every symbol as an integer in $\{1, \ldots, k\}$, taking $\llcorner = 1$ and $\vdash = 2$. This means that we can encode M as the binary string:

$$10^n 10^k 10^s 10^{acc} 10^{rej} 1\widehat{\delta}$$

where $\widehat{\delta}$ is taken to be the concatenation of the encoding of every tuple (p, a, q, b, d) describing the transition relation δ, where tuple (p, a, q, b, d) for $d \in \{1, 2\}$ is encoded as:

$$10^p 10^a 10^q 10^b 10^d$$

(recall that both states and symbols are represented by integers).

This mean that to every Turing machine M we can associate an encoding $\langle M \rangle$ over $\{0, 1\}$ which we can interpret as a binary string and thus as an integer, giving us a map from $T(\Sigma)$ to $\mathbb{N}$, which with some effort we can show is injective. So $T(\Sigma) \preceq \mathbb{N}$.

Next, we can show that $\mathbb{N} \preceq \Sigma^*$. That's actually pretty easy. Since $\Sigma \neq \varnothing$, then there is an $a \in \Sigma$. The map $f : \mathbb{N} \longrightarrow \Sigma^*$ given by $f(n) = a^n$ is clearly one-to-one. So $\mathbb{N} \preceq \Sigma^*$

Finally, Cantor's Theorem gives us that $\Sigma^* \prec \wp(\Sigma^*)$. Putting all these inequalities together we get $T(\Sigma) \prec \wp(\Sigma^*)$.

Now, consider the function $L : T(\Sigma) \longrightarrow \wp(\Sigma^*)$ that associates to every Turing machine the language it accepts. By Cantor-Bernstein, if $T(\Sigma) \prec \wp(\Sigma^*)$, then $\wp(\Sigma^*) \not\preceq T(\Sigma)$, and therefore there is no onto function $T(\Sigma) \longrightarrow \wp(\Sigma^*)$. Since L is a function of that form, L cannot be onto. Therefore, there must be a language $A \in \wp(\Sigma^*)$ such that there is no M with $L(M) = A$. That is, A is undecidable.

A Specific Undecidable Language

The argument above shows that there must be at least one undecidable language. It doesn't help us identify one.

First, we observe that Turing machine a property of *universality*. It is possible to write a *universal* Turing machine U that takes as input an encoding of $\langle M \rangle$ of a Turing and an input string w, and simulates running Turing machine M on input string w, accepting when M accepts and rejecting when M rejects. I'm not going to give the description of such a Turing machine U, but I'll put some pointers on the course web page. Note that this is no stranger than writing, say, a Python interpreter in Python. The key here is that Turing machines can simulate other Turing machines.

With that in mind, consider the following language, called the Halting Problem:

$$HP = \{\langle M \rangle \texttt{\#} w \mid M \text{ halts on input } w\}$$

For an input not of the form $\langle M \rangle \texttt{\#} w$, perhaps because the first part of the input is not a valid encoding for a Turing machine, that input is not in HP. Without loss of generality, we can take the encoding of M to be over $\{0, 1\}$.

I claim that HP is undecidable, that is, there is no total Turing machine M that accepts HP. It turns out to be a similar argument than the one used for Cantor's Theorem.

First, some notation. If $y \in \{0, 1\}^*$, let M_y be the Turing machine with encoding y, so that $\langle M_y \rangle = y$. If y is not a valid encoding of a Turing machine, then pick an arbitrary but fixed Turing machine M^* and take $M_y = M^*$ for any such y.

We argue that HP is undecidable by contradiction. Assume it *were* decidable. Then we'd have a total Turing machine K that accepted HP. That is, K accepts

$\langle M \rangle$#w exactly when M halts on w, and rejects when M does not halt on w.

Construct another Turing machine I as follows:

On input x:

1. Overwrite input tape with $\langle M_x \rangle$#x
2. Simulate K on $\langle M_x \rangle$#x.
   Accept if K rejects, go into an infinite loop if K accepts.

Clearly, we can implement I as a Turing machine if we have a total K. Since I is a Turing machine, it has an encoding $\langle I \rangle$. Also, we can ask if I halts on any given input. In particular, we can ask if it halts on input $\langle I \rangle$.

So does I halt on $\langle I \rangle$?

There are two possibilities. The first is that I halts on input $\langle I \rangle$. But this happens only when K rejects $\langle I \rangle$#$\langle I \rangle$ by the definition of I. And K rejecting $\langle I \rangle$#$\langle I \rangle$ means that I does *halt* on $\langle I \rangle$ by definition of K, though. So if I halts on $\langle I \rangle$, then I does not halt on input $\langle I \rangle$.

The other possibility is that I does not halt on input $\langle I \rangle$. But this happens only when K accepts $\langle I \rangle$#$\langle I \rangle$ by the definition of I. And K accepting $\langle I \rangle$#$\langle I \rangle$ means that I halts on $\langle I \rangle$ by definition of K. So if I does not halt on $\langle I \rangle$, then I halts on input $\langle I \rangle$.

Either way, we get an absurdity. So our assumption that K existed must be wrong. There is no such K. And thus *HP* is undecidable.