Proof by Reduction

Like all good explanations, we start with a joke that is probably only funny to the storyteller:

> An engineer and a mathematician were hiking when they were suddenly attacked by a bear. The engineer grabbed a stick and, yelling and stabbing wildly with the stick, managed to fight off the bear. The next day, the two went out for another hike, and again, they were attacked by the same bear. This time, the mathematician, realizing that he was the closest to a stick, picked it up and handed it to the engineer, thereby reducing the bear problem to a previously solved problem.

Reduction proofs[25] are a little counterintuitive. When we construct them for the purposes of computability proofs, we will always have two facts in mind:

[25] Reduction proofs should not be confused with lambda reductions. Although they both share the word "reduction," they are completely unrelated topics.

1. We want to disprove a fact about some problem of interest, A (e.g., "A is computable.")
2. We already know a fact something about some other, possibly related problem, B (e.g., "B is not computable").

Like all formal tools, reduction proofs are a template (a "form"). The trick is to recognize when the problem fits the mold. When the tool is used correctly, out pops the answer.

Here's the template we're going to follow. Let P be a logical proposition (a statement that is either true or false), and let Q be another logical proposition implied by P. In other words,

$$P \Rightarrow Q$$

For example, P could be the proposition "it is sunny outside." Q could be "it is not snowing." If we think that one implies the other, then we would read $P \Rightarrow Q$ as "if it is sunny outside, then it is not snowing." This statement is clearly true. P really does imply Q. However, since this is Williamstown, if you all look outside, it might actually be snowing. If that's the case, it cannot be sunny outside.

The above example should suggest to you that one way you might try to disprove a statement P is to show that an implied statement Q is false.

In other words, if we claim $P \Rightarrow Q$, and P really implies Q, and then we show $\neg Q$, then it must also be the case that $\neg P$. (If you're having trouble seeing why I am allowed to use this trick, see the derivation from first principles at the bottom of this section.)

Finding a reduction

Let's apply this template. What is the problem of interest? Consider the following question: Is it possible to write a function, `halt`$_0$? When given a program `p` and an input `i`, `halt`$_0$ returns `true` if and only if `p(i)` does not halt.

Given about what we know about computability (i.e., that `halt` is not computable), we should have a nagging suspicion that `halt`$_0$ is also not computable. But can we set up a logical implication of the above form to *prove* that `halt`$_0$ is not computable? Indeed we can. Remember—the key is to imply something that we know *cannot* be true.

Let's start with a fact that we know cannot be true. Q: "`halt` is computable."

Now, can we show that Q follows logically from the fact that we want to disprove? P: "`halt`$_0$ is computable."

We're going to use the same $P \Rightarrow Q$ logic trick as in our snowing example above, and show that if P is true, P logically implies Q. This is where reductions fit in. A reduction is an *algorithm* that turns one problem into another problem. Why do we want an algorithm? Well, last time I checked, if computers did one thing well, it was logic. So if one can write an algorithm for transforming problems, it's logical, and a computer really could do it.

Remember when you learned about proving things using mathematical induction? When proving the inductive step, which is an implication of the form $P \Rightarrow Q$, recall that you were allowed to assume P. Since we are also attempting to prove an implication, we also get to assume P is true. Remember that P is "`halt`$_0$ is computable."

Here's an algorithm (in Python) that turns instances of `halt` into `halt`$_0$.

```
def halt(p,i):
  return not halt0(p,i)
```

If we assume that P is computable, we really could have a `halt`$_0$ function. Some really smart person could have coded it up and stuck it in a library for us. So the above function, `halt`, should be possible, right? I didn't do anything fancy. I just followed the rules of Python. `halt` just calls `halt`$_0$ and negates the result.

Looking back at our statements,
Q: "`halt` is computable"

and
P: "`halt`$_0$ is computable"
what the reduction just showed is that $P \Rightarrow Q$. We can't avoid $P \Rightarrow Q$, because look, I just made Q happen using P. Therefore, it is true that P implies Q.

But we also know, because I showed you in class (or if you're reading this early, I will), is that Q cannot be true. `halt` is not computable. $\neg Q$.

Therefore `halt`$_0$ is not computable. $\neg P$.

Why does $\neg Q \Rightarrow \neg P$ when $P \Rightarrow Q$?

We can derive what happens to the antecedent (P) of an implication ($P \Rightarrow Q$) when we know that the consequent (Q) is not true. I claim that $P \Rightarrow Q$ is logically equivalent to the statement $\neg P \vee Q$. We can prove this equivalence rigorously by working out a truth table.[26] $\neg P \vee Q$ is easier to work with, because it gets rid of the pesky implication symbol (whatever *that* means).

[26] When I am confused about what conditionals do in code, I sometimes work out truth tables. This is one of those tricks that fellow students occasionally mocked me for doing. "Dan has to write out the truth tables to understand it! Ha ha." This is a silly thing to be elitist about. Programming is hard. I can still solve the problem. More importantly, I am not confused.

P	Q	$\neg P \vee Q$	$P \Rightarrow Q$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

We know that $\neg P \vee Q$ is true, just as we do with our Python program above. Let's start our proof with that fact.

$\neg P \vee Q =$ `true`	given
$\neg P \vee$ `false` $=$ `true`	because Q is false
$\neg P =$ `true`	because "anything" $\vee$ `false` is just "anything"
$P =$ `false`	by negation

Therefore, if $P \Rightarrow Q$ itself is true and Q is false, then P must be false.