Three Proofs on the Existence of Uncomputable Functions

We give three proofs that there are functions that are not computable.

A simple counting argument

In short: there are too many functions for there to be programs to compute them all. To make things concrete, let’s consider only functions from $\text{Simple-functions} = \{ f \mid f : \mathbb{N} \to \{0, 1\} \}$. And, by “program”, we mean a (syntactically valid) Python program, or C++ program, or program in your choice of programming language. The argument works for any choice. We use Python only as an example.

We’ll show that the set of Simple-functions is uncountably infinite, whereas the set of Python programs is only countably infinite.

Simple-functions is uncountably infinite: Each function $f \in \text{Simple-functions}$ is uniquely specified by an infinite bitstring $b_i$, where the $i^{th}$ bit $b_i$ gives the value of $f(i)$. Put a decimal point in front of such a bitstring, and obtain a real number, written in binary, in the interval $[0,1]$. Since each real number in this interval has at least one such bitstring, and the cardinality of $[0,1]$ is uncountably infinite, it follows that Simple-functions is uncountably infinite.

The set of all (Python) programs is only countably infinite: Recall from CS 173, or from wherever, that if an (infinite) set can be enumerated so that each element of the set appears in the enumeration at least once, then the set is countably infinite. We show how to enumerate all possible Python programs.

Each Python program is a finite length sequence of characters over a finite alphabet (say, all 128 ASCII characters). We can simply enumerate all character sequences of length 0, then of length 1, then of length 2, etc. (Clearly there are a finite number of character sequences for any given length $i$ - namely $128^i$ of them.) We cross off any sequences that do not correspond to valid Python programs. We are left with a listing of all possible Python programs.

A diagonal argument

Using again the fact that there are only countably many Python programs, let $P_i$ denote the $i^{th}$ program on the listing above. Thus, $P_0, P_1, P_2, \ldots$ is a listing of all possible Python programs. We will construct a function that is not computed by any of the $P_i$. Towards this end, consider the behavior of $P_i(n)$ for any $n \in \mathbb{N}$. $P_i(n)$ might equal 0, it might be undefined (because $P_i$ gets into an infinite loop), or it might equal some value other than 0. More briefly, for each $n$, either $P_i(n) = 0$ or $P_i(n) \neq 0$.

Define $f(n)$ as follows: $f(n) = 1$ if $P_n(n) = 0$. Otherwise, $f(n) = 0$. (There is a nice “diagonal” picture of this which you should have seen in Cantor’s proof of the uncountability of the reals, and which I am too lazy to typeset. This is where the name “diagonalization” comes from.)

Now observe that the function $f$ is not computed by any Python program. How could it be? Could it be $P_0$? No, because for each $i$, $f(i) = 0$ if and only if $P_i(i) \neq 0$. Thus $f$’s output differs from the behavior of every Python program.

A proof by contradiction that a particular function cannot be computed

In class we introduced the definition of the function minlength(), which on input of a natural number $n$, outputs the length of the shortest Python program that when started with no input, outputs $n$ and halts. For example, $\text{minlength}(100) \leq 20$ because the following program prints “100” and halts, and has length 20.

```python
def x():
    print(100)
```

And, $\text{minlength}(10^{1,000,000}) \leq 100$, because the program below has length at most 100.
def x():
    n = 1
    for i in range(1000000):
        n = n*10
    print n

to show that minlength() is not computable, we produce what is essentially a more formal version of Berry's paradox: What is the least nonnegative integer that cannot be described in fewer than fourteen words? There must be one, since there are an infinity of nonnegative integers, and only finitely many sentences of fourteen words or less. Suppose m is in fact the least such one. Then a perfectly reasonable description of m is “the least nonnegative integer that cannot be described in fewer than fourteen words”. Go ahead and count and you'll see that we described m in thirteen words. Arghh! A paradox.

To formalize this via function minlength(), let's suppose by way of contradiction that there was a Python program that computed the function:

def minlength(n):
    blah blah blah n blah blah blah
    blah blah blah
    blah blah blah result blah blah blah
    return result

we will show that this assumption leads to a contradiction.

This program must have some length, call it m. Now consider the above code for minlength(), taken together with the following additional code

def explodeMyHead():
    smallsize = m + 500 + 2*log m  # 'm' is the constant from above, not the character.
    i = 0
    while minlength(i) <= smallsize:
        i = i+1
    print i

Two questions: (1) What does explodeMyHead, together with minlength, do? (2) How long are the two functions taken together as a program?

(1) The two functions together print out the smallest i such that the shortest program to print i is large, meaning the length exceeds smallsize.\(^1\) Thus, what is printed is the first i that can only be printed by "long" programs - those of size exceeding smallsize.

(2) BUT, the two functions together require FEWER THAN smallsize characters. To see this, note that it takes m characters for the code minlength. And, besides the two appearances of the constant m in explodeMyHead, there are no more than 400 characters (actually, way fewer than that). Finally, the constant m appears twice, taking 2log\(_{10}\) m characters. So, the total length for the two combined is at most m + 400 + 2log\(_{10}\) m < m + 500 + 2log\(_{10}\) m = smallsize.

We thus have a program of size less than smallsize, printing out a number for which the smallest program that can print it out has size exceeding smallsize. Arghh! A contradiction, and the only assumption made was that code for minlength() existed, so we conclude that there is no such program.

\(^1\)You may want to first convince yourself that there must be some i satisfying that criterion, so that the while loop does terminate. This must be so because smallsize is a constant, and there are only finitely many programs of size at most smallsize, yet there are an infinite number of i's whose programs we are considering.