Math 29: Halting Problems

April 20th, 2022

1 Halting Problems

Last time, we observed that the following set is not computable:

$$\{e : \varphi_e(e) \downarrow = 0\}$$

In this case, it cannot be, as it would lead to a contradiction on the inclusion of an index e for its characteristic function: If e is in the set, then $\varphi_e(e) \downarrow = 0$. But since e is an index for the characteristic function, and e is in, then we must also have $\varphi_e(e) \downarrow = 1$, a contradiction. If e is not in the set, then $\varphi_e(e) \downarrow = 0$ by the definition of a characteristic function, which contradicts the fact it is not in the set.

Suppose we try to fix this contradiction by instead looking at

$$\{e : \varphi_e(e) \downarrow = 1\}$$

Now the above contradiction is no longer a problem. If e is an index for the characteristic function of this set, then if e is in the set, $\varphi_e(e) \downarrow = 1$ as expected. If it is not in the set, $\varphi_e(e) \downarrow = 0$ is does not provide a contradiction. Does this mean that this set is computable, while the former is not?

It turns out that the answer is no. Suppose, for the sake of contradiction, that it is computable. Because it is computable, so is its complement. Let e be an index for the characteristic function of its complement. Now consider $\varphi_e(e)$. If e is in the complement, then $\varphi_e(e) \downarrow = 1$, which is a contradiction. If it's not in the complement, then $\varphi_e(e) \downarrow = 0$, which is also a contradiction.

We're starting to see the problem with trying to compute a set whose membership is defined based on convergence information. In fact, very broadly, determining which indices and inputs cause the universal machine to halt are not computable. The most general problems about halting that we can define are the following:

$$M = \{\langle e, k, n\rangle : \varphi_e(k) \downarrow = n\}$$
$$H = \{\langle e, k\rangle : \varphi_e(k) \downarrow\}$$
$$K = \{e : \varphi_e(e) \downarrow\}$$

Theorem 1. *M, H, and K are all not computable.*

Proof: If we can prove that K is not computable, then that implies H is not computable: if it were, then we could compute K by asking if $\langle e, e\rangle$ is in H. However, we cannot compute K as otherwise we'd be able to compute $\{e : \varphi_e(e) = 0\}$ as follows: if $e \notin K$, then it is not in. If $e \in K$, then run $\varphi_e(e)$ until it converges, and check if the output is 0. Since this set is not computable, K is not either. Similarly, if we could compute M, then we could compute this set via checking if $\langle e, e, 0\rangle$ is in M. Therefore, none of the three is computable.

Alternatively, there is a very classic argument to show that K is not computable. Suppose that it is. Then define the computable function

$$f(e) = \begin{cases} 1 & \text{if } e \notin K \\ \uparrow & \text{if } e \in K \end{cases}$$

As K is computable, f is computable, and therefore it has some index i. Now consider whether or not $i \in K$. If $i \in K$, then $\varphi_i(i)\downarrow$. But, looking at the definition of f, this means that $f(i) = 1$, which only occurs when $e \notin K$, a contradiction. Conversely, if $i \notin K$, then $\varphi_i \uparrow$. Again looking at the definition of f, this must mean that $i \in K$, also a contradiction. Therefore, K cannot be computable.

The proof of the above highlights an interesting facet of computability, which is clear when one considers it. If we think of computing a set as solving a problem, then if a specific instance of the problem is not computable, the general case cannot be computable either. A famous example of this is seen in Hilbert's tenth problem, which we discussed on the first day of class. Recall that Hilbert's tenth problem asked to find an algorithm to determine whether or not an input Diophantine equation (a polynomial in finitely many variables with integer coefficients) has integer roots. I.e., it asks if the set of Diophantine equations with integer roots is computable. The reason why the problem didn't ask for a radical equation to compute the roots is because the Abel-Ruffini theorem already guarantees no such thing exists even for single-variable polynomials of degree at least five.

Similarly, in the above argument, we deduced that M and K could not be computable, as that would imply a much narrower problem would also be computable, which we already proved to be impossible.

One might wonder if the fact that the halting problem is not computable is a side-effect of Rice's Theorem. After all, that broadly showed that attempting to compute which functions were in a certain non-trivial class is an impossible problem. However, we will see that the halting problem is not an index set.

Lemma 2. *K is not an index set.*

Proof. Homework 4 Question 3. □

The halting problem is a central piece of computability theorem. We will come back to it multiple times throughout the course, and it is the most important example of a noncomputable set.

1.1 Fixing the Halting Problem

The fact that H and K are not computable means that, in general, the only way to tell if a program halts is to run it and find out. This leads us to an important insight which will allow us to work around the fact that the halting problem is not computable.

The notation

$$\varphi_{e,s}(n)$$

represents the result of running the machine coded by e on input n for s **stages** - either s-many instructions in a Turing machine, or operating s-many nodes in a register machine. The formal definition of a stage is largely irrelevant - it could be five instructions, seven nodes, etc. As long as it is a finite, predictable amount of time. We use the notation $\varphi_{e,s}(n)\uparrow$ to denote that, after running the e-th machine on input n for s stages, the machine has still not stopped and returned a value yet. It may in the future, but it has not yet. If it has stopped in s or fewer steps and output m, then we write $\varphi_{e,s}(n)\downarrow= m$.

In other words, $\varphi_e(n)\uparrow$ if and only if, for all s, $\varphi_{e,s}(n)\uparrow$. If $\varphi_e(n)\downarrow$, then there exists some s such that $\varphi_{e,s}(n)\downarrow$. We can think of the number of stages as time: if a machine is going to halt, then there is some time at which it will halt. If it does not halt, then it does not halt at any given time. In the meantime, given a specific time, it may not have halted yet, but it could halt in the future.

When stages are taken into account, we can now compute whether or not a program has halted **yet**.

Lemma 3. *The following sets are computable:*

$$\{\langle e,n,m,s\rangle : \varphi_{e,s}(n)\downarrow= m\}$$

$$\{\langle e,n,s\rangle : \varphi_{e,s}(n)\downarrow\}$$

$$\{\langle e,s\rangle : \varphi_{e,s}(e)\downarrow\}$$

Proof: Run the machine coded by e on input n (e) for s-stages. If it has halted (and output n), return true. If it has not halted yet, return false. This process is computable because we can modify the universal machine to have a counter following every node which decrements some register containing the current stage. If the stage counter runs out before we reach a stop node, return 0. If we reach a stop node first, return the original return value plus one. We can then react to the return value accordingly.

Following similar notation, recall that $W_e = \{n : \varphi_e(n) \downarrow\}$. Then $W_{e,s} = \{n : \varphi_{e,s}(n) \downarrow\}$, that is the set of all n which have converged by stage s. It follows that

$$W_e = \bigcup_{s \in \omega} W_{e,s}$$

Often when we construct c.e. sets using priority arguments, we proceed by defining each $W_{e,s}$ inductively. We let $W_{e,0} = \emptyset$, then inductively define $W_{e,s+1}$ by adding some new elements to $W_{e,s+1}$ in some computable fashion. We then let our final set be the union, or "limit," of this whole process. This relies heavily on the fact that c.e. sets can be listed out, but we can't necessarily list out their complement.

1.2 Return to the Halting Problem

Notice that the halting problem is c.e.: we can list out the elements of K by running each $\varphi_e(e)$ in parallel and enumerating e if $\varphi_e(e)$ halts. A terser proof is to note that K is the domain of the function $U(\langle e, e\rangle)$. It turns out that not only is the halting problem c.e., but it is in fact the peak of all c.e. sets. As we shall see later in the course, if we know the contents of K, we will be able to figure out the contents of **ANY** other c.e. set. In the terminology of solving problems, if we can solve the halting problem, we can solve any other c.e. problem.

We now turn our attention to **Post's Problem**: Emil Post asked the following: is there a c.e. set which is not computable, but which does not give us enough information to solve the halting problem? In other words, the halting problem is the most complex c.e. problem in every way, but is that because they are all the same? Or are there some problems which are easier to solve than others? It turns out that all c.e. problems are not the same, and you can find ones with essentially any relationship between them that you want. We will not prove the full scope of this fact, but we will see some specific examples.