Lecture 4: Counting Arguments for Turing Machines

Anup Rao

April 8, 2021

In the last lecture, we used counting arguments to show that there are functions that cannot be computed by circuits of size $o(2^n/n)$. If we were to try and use the same approach to show that there are functions $f : \{0,1\}^* \to \{0,1\}$ not computable Turing machines we would first try to show that:

$$\text{\# turing machines} \ll \text{\# functions } f.$$

This approach doesn't seem like it makes any sense at first, because both numbers here are infinite. Luckily, mathematicians have long studied how to compare the sizes of infinite sets.

Recall the definitions of the following sets:

$\mathbb{N} = \{1, 2, 3, \ldots\}$	the natural numbers
$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$	the integers
$2^{\mathbb{N}} = \{A \subseteq \mathbb{N}\}$	the set of sets of natural numbers
$\mathcal{Q} = \{i/j : i, j \in \mathbb{Z}, j \neq 0\}$	the rational numbers
$\mathbb{R} = \left\{ \lim_{i\to\infty} x_i : x_1, x_2, \ldots \in \mathcal{Q} \text{ is a convergent sequence} \right\}$	the real numbers

To compare the sizes of these sets, we use the concept of countability. A function $\phi : \mathbb{N} \to S$ is said to be surjective if for every $s \in S$, there is an $i \in \mathbb{N}$ such that $\phi(i) = s$.

Definition 1. *A set S is* countable, *if there is a surjective function $\phi : \mathbb{N} \to S$.*

Equivalently, S is countable if there is a list $\phi(1), \phi(2), \ldots$ of elements from S, such that every element of S shows up at least once on the list. In a sense, if S is countable, that corresponds to asserting that $|S| \leq |\mathbb{N}|$.

Let us try to understand which of the sets we have discussed are countable.

Fact 2. $\mathbb{N}$ *is countable.*

Proof Consider the list $1, 2, 3, \ldots$. This obviously contains every element of $\mathbb{N}$. ■

Fact 3. $\mathbb{Z}$ *is countable.*

Proof Consider the list $0, 1, -1, 2, -2, 3, -3, \ldots$. This obviously contains every element of $\mathbb{Z}$. ■

Fact 4. *$\mathbb{Z} \times \mathbb{Z} = \{(i, j) : i, j \in \mathbb{Z}\}$ is countable.*

Proof Consider the list

$$(0,0),(1,0),(1,1),(0,1),(-1,1),(-1,0),$$
$$(-1,-1),(0,-1),(1,-1),(2,-1),\ldots,$$

shown in Figure 1. This list contains every element of $\mathbb{Z} \times \mathbb{Z}$. Indeed, we are enumerating all pairs (i, j) where the $\max\{|i|, |j|\}$ is 0, then all pairs where $\max\{|i|, |j|\}$ is 1 and so on. Clearly, every pair occurs somewhere in the list. ■

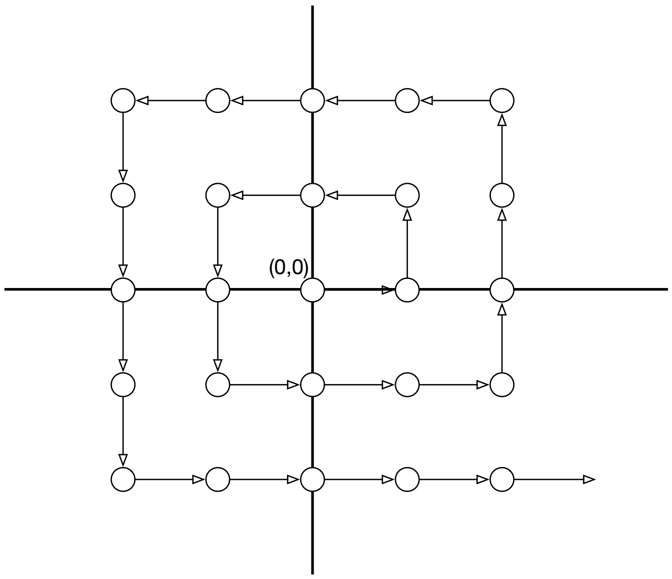

Figure 1: Enumeration of $\mathbb{Z} \times \mathbb{Z}$.

Fact 5. *$\mathcal{Q}$ is countable.*

Proof Since $\mathbb{Z} \times \mathbb{Z}$ is countable, just take the list of all pairs from $\mathbb{Z} \times \mathbb{Z}$, and discard an entry if $j = 0$ and replace it with i/j if $j \neq 0$. This gives an enumeration of $\mathcal{Q}$. ■

The interesting thing is that some sets can be shown to be uncountable, using the technique of *diagonalization*.

Fact 6. *$2^{\mathbb{N}}$ is not countable.*

Proof Suppose there was some list of sets $A_1, A_2, \ldots$, then consider the set

$$T = \{i : i \in \mathbb{N}, i \notin A_i\}.$$

We claim that T is not in the list. Indeed, suppose $T = A_j$ for some j. Then if $j \in A_j$, $j \notin T$ by our construction, and if $j \notin A_j$, then $j \in T$. In either case, $T \neq A_j$. ■

The proof we just used is called a proof by diagonalization, because we can think of doing it using the picture described in Figure 2. We encode each set in our list using a binary string. The set T

It was discovered by Cantor

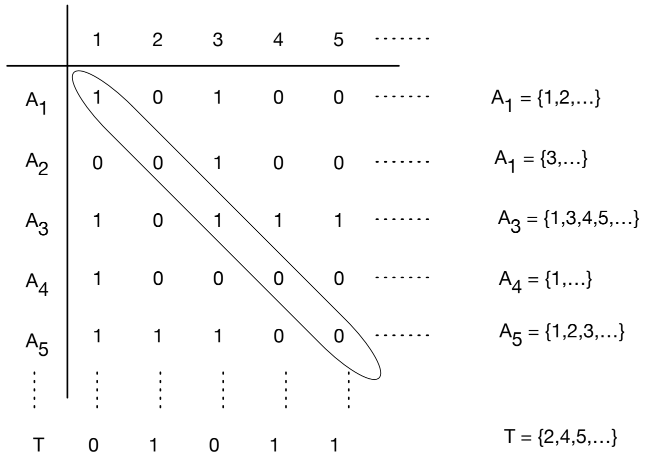

Figure 2: Diagonalization of a list of sets.

we picked is obtained by taking the set that is obtained by choosing something that disagrees with the diagonal in the picture.

A very similar idea can be used to show that the set of functions $f : \{0,1\}^* \to \{0,1\}$ is not countable

Can you use a similar idea to show that the real numbers are not countable?

Fact 7. *The set of functions mapping* $\mathbb{N}$ *to* $\{0,1\}$ *is not countable.*

Proof Suppose there is an enumeration $f_1, f_2, \dots$ of all the functions. Then consider the function f defined by

$$f(i) = \begin{cases} 1 & \text{if } f_i(i) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Now f cannot appear in the list, because for every i, $f(i) \neq f_i(i)$, so $f \neq f_i$. ■

Now, note that the set of binary strings $\{0,1\}^*$ is itself countable. You can enumerate them by enumerating the strings of length 0, then the strings of length 1, and so on. But this means that the same proof as above shows that the set of functions $f : \{0,1\}^* \to \{0,1\}$ is *not* countable. Indeed, given any list of functions $f_1, f_2, \dots$, the function f defined below cannot be on the list:

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is the } i\text{'th binary string and } f_i(x) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

But this last fact already implies that there must be a function that cannot be computed by any Turing machine! Indeed, the set of Turing machines *is* countable, because we can enumerate them by enumerating the set of all programs. This means that the set of functions that are computable by Turing machines is also countable. For concreteness we give a direct proof below:

Theorem 8. *There is a function that is not computed by any Turing Machine.*

Proof Given a string α, we write M_α to denote the Turing Machine whose code is α. Consider the function $f : \{0,1\}^* \to \{0,1\}$ defined as follows:

$$f(\alpha) = \begin{cases} 1 & \text{if } M_\alpha(\alpha) = 0 \\ 0 & \text{else.} \end{cases}$$

Note that above, if the machine M_α does not halt on input α, then $f(\alpha) = 0$.

No Turing Machine can compute this function, for if there was some machine that could, then let γ denote the binary encoding of its code. Then we have that $M_\gamma(\gamma) = f(\gamma)$, but this contradicts the definition of f, since if $f(\gamma) = 0$, then $M_\gamma(\gamma)$ cannot be 0, and if $f(\gamma) = 1$, $M_\gamma(\gamma)$ cannot be 1. ■

Let us point out that this is philosophically a very powerful fact. A consequence of it is that assuming the Church-Turing Thesis is true, there are some ways to manipulate information that can never occur in the universe. It seems hard to imagine a physical process that violates the Church-Turing thesis, and it also seems hard to stomach the fact that the universe cannot manipulate information in a particular way, yet one of those two (admittedly wishy washy) strange things must happen.

You may object that the uncomputable f that we found above is very unnatural, but actually it is not hard to come up with natural examples that are also impossible to compute using Turing Machines.

For example, we can define the function $\mathsf{HALT} : \{0,1\}^* \to \{0,1\}$ that takes as input two strings α, x, and then decides whether $M_\alpha(x)$ halts or runs forever. This seems like a very useful function to compute, but it is also uncomputable.

To reason about halting, we first need a basic fact about Turing machine—there is a machine that can compile and run the code of any other machine efficiently:

Theorem 9 (Universal Simulator)**.** *There is a turing machine M such that given the code of any Turing machine α and an input x as input to M,* *if α takes T steps to compute an output for x, then M computes the same output in $O(CT \log T)$ steps, where here C is a number that depends only on α and not on x.*

Now we are ready to prove that HALT is not computable.

Theorem 10. HALT *is not computable by a Turing Machine.*

Proof Suppose it was. Then consider the machine M that on input α first simulates $\mathsf{HALT}(\alpha, \alpha)$. If the answer is that $M_\alpha(\alpha)$ halts, then M simulates $M_\alpha(\alpha)$ and outputs the opposite of its output. If $M_\alpha(\alpha)$ does not halt, then M outputs 0. Then M computes the uncomputable function f above. ■

Gödel's Incompleteness Theorem

GÖDEL'S FAMOUS INCOMPLETENESS HEOREM is one of the most significant mathematical results of the last century. The appeal of the result is that (like many of the results in complexity theory) it proves something that seems to address the nature of our universe. If I were to put the statement of the theorem into words it would be

There is a truth that cannot be proved.

This result was very disturbing to mathematicians and scientists, because it attacks a core assumption of the scientific enterprise. The goal of modern science is to boil down the functioning of the universe to a small set of rules. The hope was that all phenomena that occur in the universe can eventually be explained as consequences of a small set of rules. Gödel's theorem is a dagger to the heart of that hope—it sounds eerily similar to something a religious leader might say in lieu of giving a scientific explanation:

This is true because it is God's will.

Now, let us get more rigorous and prove the theorem. I should note here that I am not going to prove the incompleteness theorem in the form that it was initially conceived and proved. Gödel proved these results in 1931, before he knew the definition of Turing machines, and that meant he had to make all kinds of complicated arguments and definitions. Armed with the definitions we have already seen in this course, we will be able to take a much more straightforward path to proving essentially the same thing.

Diagonalization was used to prove Gödel's incompleteness theorem in its original form.

We need to start by making more rigorous what we mean by truths that *can/cannot be proved*. Formally this is done by defining a *proof system*. For us it is enough to think of a proof system as a Turing machine M with the property that given a theorem τ and a candidate proof π, $M(\tau, \pi)$ outputs 1 only if the statement τ is true. Intuitively, the Turing machine M is a checker for all proofs. Given a theorem τ and a proof for the theorem π, the machine simply checks that each line of the proof follows from the previous one and eventually leads to the statement of the theorem τ. To make this completely formal, we need to specify exactly how theorems and proofs will be encoded as binary strings, but us ignore such details here.

Our goal is to show that for every such M, there is a theorem τ which is true, yet $M(\tau, \pi)$ will never output 1, no matter what π is. So, there is no proof π that convinces the checker that τ is true.

Now, let us turn to finding the true theorem that we will use. Crucial to the proof is a beautiful measure of the complexity of binary strings, which is interesting in its own right.

The argument given here is due to Chaitin, who proved this in the 1968.

Given $x \in \{0,1\}^*$, its Kolmogorov complexity $K(x)$ is the length of the shortest program α such that $M_\alpha(.) = x$. Namely it is the length of the shortest program that outputs x. For each $x \in \{0,1\}^*, N \in \mathbb{N}$, let $S_{x,N}$ be the assertion

$$S_{x,N} : K(x) > N.$$

Fact 11. *For every N, there is an x for which $S_{x,N}$ is true.*

Proof There are only a finite number of programs of length N, so for each N, there are only a finite number of x's such that $K(x) \leq N$. This means that almost all statements $S_{x,N}$ are true. ■

To prove Godel's theorem, suppose there is a proof checking machine M. Consider the following program M_N:

- Enumerate over all pairs (x, α), where $x \in \{0,1\}^*$, $\alpha \in \{0,1\}^*$. If $M(S_{x,N}, \alpha) = 1$, output x.

If every statement $S_{x,N}$ has a proof convincing M, then M_N would always halt, since it would eventually find some string x and a proof α proving $S_{x,N}$. But the program M_N can be described using just $O(\log N)$ bits, and it outputs a string x for which $K(x) > N$. For N large enough, this contradicts the definition of $K(x)$.