Lecture 3: Diagonalization, the Incompleteness Theorem and Complexity Classes

Anup Rao

January 19, 2022

Last time, we discussed the fact that there are functions that require large circuits. Today, we use a similar strategy to argue that there are functions that cannot be computed by Turing machines.

Diagonalization

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:

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

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 $\phi(1), \phi(2), \dots$ of elements from S, such that every element of S shows up at least once on the list.

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. ■

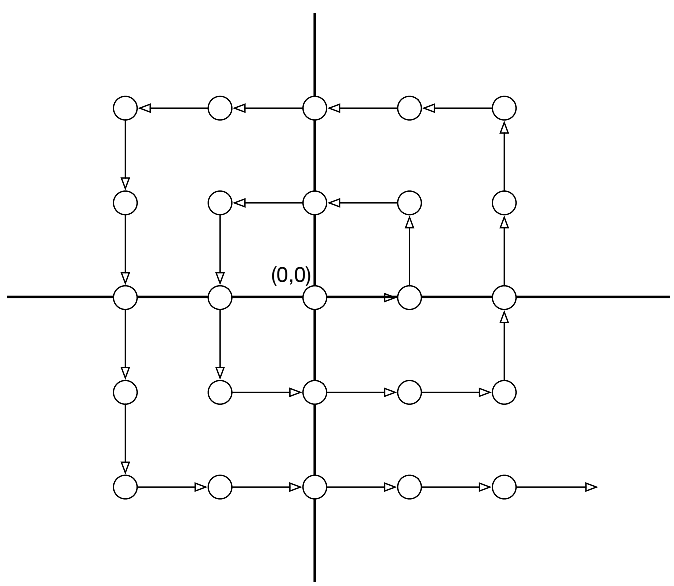

Figure 1: Enumeration of $\mathbb{Z} \times \mathbb{Z}$.

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

It was discovered by Cantor

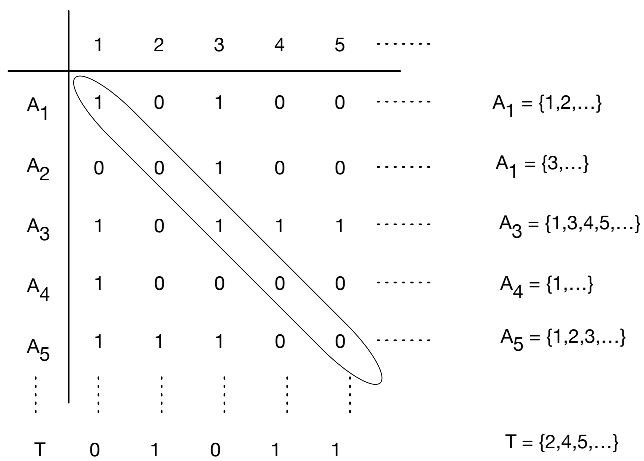

Figure 2: Diagonalization of a list of sets.

A very similar idea can be used to show that the real numbers are not countable:

Fact 7. $\mathbb{R}$ *is not countable.*

Proof Every real number can be thought of as a number with a potentially infinite decimal expansion.

Suppose $r_1, r_2, \ldots$ is an enumeration of the real numbers. Consider the real number $t = 0.d_1 d_2 \ldots$, where the i'th digit d_i is chosen so that d_i is not the same as the i'th digit of r_i. Then t is a real number that does not occur anywhere in the list of r_i's, since it disagrees with the i'th number in the i'th digit after 0. ■

A very similar idea gives an impossibility result for Turing Machines.

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

Before we see the the simple proof, 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.

We shall need some notation before discussing the proof. Given a string α, we write M_α to denote the Turing Machine whose code is α.

Proof Consider the function $f : \{0,1\}^* \to \{0,1\}$ defined as follows:

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

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. ■

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.

Theorem 9. 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

Diagonalization was also used to prove Gödel's famous incompleteness theorem. The theorem is a statement about proof systems. We sketch a simple proof using Turing machines here.

A proof system is given by a collection of axioms. For example, here are two axioms about the integers:

1. For any integers a, b, c, $a > b$ and $b > c$ implies that $a > c$.
2. For any integer a, $a + 1 > a$.

Given a list of such axioms, a proof is a sequence of statements that uses the axioms to prove that a statement is true. For example, to prove that $a > b$ implies that $a + 1 > b$, we can combine the assumption $a > b$ with the axiom $a + 1 > a$ and the first axiom, to prove $a + 1 > b$.

Prior to Gödel's work, mathematicians were trying to axiomatize all of mathematics. They were looking for a set of finite axioms that could be combined to prove any proof statement. Godel proved that this a doomed project.

A set of axioms is *consistent* if the axioms don't contradict each other. The set of axioms is complete if every true statement can be derived from the set of axioms. Godel proved:

Theorem 10. *Every consistent finite set of axioms is incomplete.*

We give an alternate proof due to Chaitin. 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 statement

$$K(x) > N.$$

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 some finite set of axioms A. Consider the following program M_N:

- Enumerate over all pairs (x, α), where $x \in \{0,1\}^*$, $\alpha \in \{0,1\}^*$. If α describes a proof of $S_{x,N}$ using the axioms A, output x.

If the finite set of axioms were complete, M_N would always halt, since it would 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 is a contradiction, and so A must be incomplete.

Complexity Classes

let us talk *complexity classes*. We are interested in classifying functions according to their complexity, so it makes sense to lump functions into sets of similar complexity:

Definition 12. *Define* $\mathsf{DTIME}(t(n))$ *to be the set of functions*

$$\mathsf{DTIME}(t(n)) = \{f : \{0,1\}^* \to \{0,1\} | f \text{ is computable in time } O(t(n))\}.$$

Similarly,

Definition 13. *Define* $\mathsf{DSPACE}(s(n))$ *to be the set*

$$\mathsf{DSPACE}(s(n)) = \{f : \{0,1\}^* \to \{0,1\} | f \text{ is computable in space } O(s(n))\}.$$

Once we have these definitions, we can try to define what it means for a function $f : \{0,1\}^* \to \{0,1\}$ to be *efficiently computable*. A reasonable definition of efficient computation should allow enough time to read all of the input, which takes $\Omega(n)$ time. So we should definitely include $\mathsf{DTIME}(n)$ in our set of efficiently computable functions. Further, if one algorithm calls another as a subroutine, and both are efficient, we would like to say that the combined algorithm is also efficient. The minimal class satisfying these assumptions is the class

Definition 14. $\mathbf{P} = \bigcup_{c \geq 1} \mathsf{DTIME}(n^c)$.

Of course there is a whole spectrum of classes above P. For example:

Definition 15. $\mathbf{EXP} = \bigcup_{c \geq 1} \mathsf{DTIME}(2^{n^c})$.

And,

Definition 16. $\mathbf{E} = \bigcup_{c \geq 1} \mathsf{DTIME}(2^{cn})$.

For space bounded computation, we need to have enough space to manipulate pointers into the inputs, which takes $\log n$ bits, before we get interesting classes. The first such class is:

Definition 17. $\mathbf{L} = \mathsf{DSPACE}(\log n)$.

Definition 18. $\mathbf{PSPACE} = \bigcup_{c \geq 1} \mathsf{DSPACE}(n^c)$.

Obviously if $t(n) = O(t'(n))$, then $\mathsf{DTIME}(t(n)) \subseteq \mathsf{DTIME}(t'(n))$. But is the containment strict? Does giving a Turing Machine more time actually allow it to compute things that it cannot compute without the extra time?

References

[FGT16] Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763. ACM, 2016.

[ST17] Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 696–707. IEEE Computer Society, 2017.