Lecture 4: Diagonalization

Anup Rao

January 12, 2023

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. But first, let us discuss the robustness of the definition of Turing Machines.

Robustness of the model: Extended Church-Turing Thesis

The reason Turing machines are so important is because of the *Extended Church-Turing Thesis*. The thesis says that *every* efficient computational process can be simulated using an efficient Turing machine as formalized above. Here we say that a Turing machine is efficient if it carries out the computation in polynomial time.

The original (non-extended) thesis made a much tamer claim: that any computation that can be carried out by a human can be carried out by a Turing machine.

The Church-Turing Thesis is not a mathematical claim, but a wishy washy philosophical claim about the nature of the universe. As far as we know so far, it is a sound one. In particular if one changed the above model slightly (say by providing 10 arrays to the machine instead of just 3, or by allowing it to run in parallel), then one can simulate any program in the new model using a program in the model we have chosen.

Claim 1. *A program written using symbols from a larger alphabet* Γ *that runs in time* $T(n)$ *can be simulated by a machine using the binary alphabet in time* $O(\log|\Gamma| \cdot T(n))$.

Sketch of Proof We encode every element of the old alphabet in binary. This requires $O(\log|\Gamma|)$ bits to encode each alphabet symbol. Each step of the original machine can then be simulated using $O(\log|\Gamma|)$ steps of the new machine. ■

Claim 2. *A program written for an* L*-tape machine that runs in time* $T(n)$ *can be simulated by a program for a* 3*-tape machine in time* $O(L \cdot T(n)^2)$.

Sketch of Proof The idea is to encode the contents of all the new work arrays into a single work tape. To do this, we can use the first L locations on the work tape to store the first bit from each of the L arrays, then the next L locations to store the second bit from each of the L arrays, and so on. To encode the location of the pointers, we

increase the size of the alphabet so that exactly one symbol from each tape is colored red. This encodes the fact that the pointer points to this symbol of the tape. The actual pointer in the new Turing machine will then do a big left to right sweep of the array to simulate a single operation of the old machine. ■

The following theorem should not come as a surprise to most of you. It says that there is a machine that can compile and run the code of any other machine efficiently:

Theorem 3. *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.*

We shall say that a machine runs in time $t(n)$ if for every input x, the machine halts after $t(|x|)$ steps (here $|x|$ is the length of the string x). Similarly, we can measure the space complexity of the machine. The crucial point is that small changes to the model of Turing machines does not affect the time/space complexity of computing a particular function in a big way. Thus it makes sense to talk about the running time for computing a function f, and this measure is not really model dependent.

Finally we have the following theorem relating Turing machines to circuits:

Theorem 4. *If M is a Turing machine that halts within T steps on inputs of legnth n, there is a Boolean circuit of size $O(T \log T)$ that simulates the execution of M on inputs of lenth n.*

To prove the theorem, you construct the Boolean circuit by building layers. Each layer simulates a single step of execution of the Turing machine. The circuit maintains a collection of gates that encode the values of x, y, z, the locations of the pointers, and the line of code that is about to be executed. Then, it uses a small number of gates to compute the updated value of all of these variables after one step of the Turing machine. Putting together all these layers gives the final circuit. In this class we will not discuss the details of this simulation.

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:

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

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 5. *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 6. *$\mathbb{N}$ is countable.*

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

Fact 7. *$\mathbb{Z}$ is countable.*

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

Fact 8. *$\mathbb{Z} \times \mathbb{Z} = \{(i,j) : i,j \in \mathbb{Z}\}$ is countable.*

Proof Consider the list

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

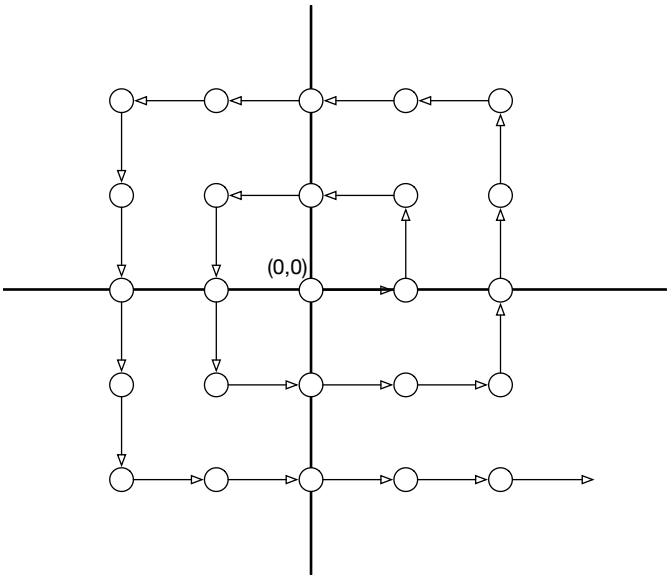

Figure 1: Enumeration of $\mathbb{Z} \times \mathbb{Z}$.

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. ■

Fact 9. *$\mathcal{Q}$ 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 10. *$2^{\mathbb{N}}$ 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, $T \neq A_j$. ■

It was discovered by Cantor

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.

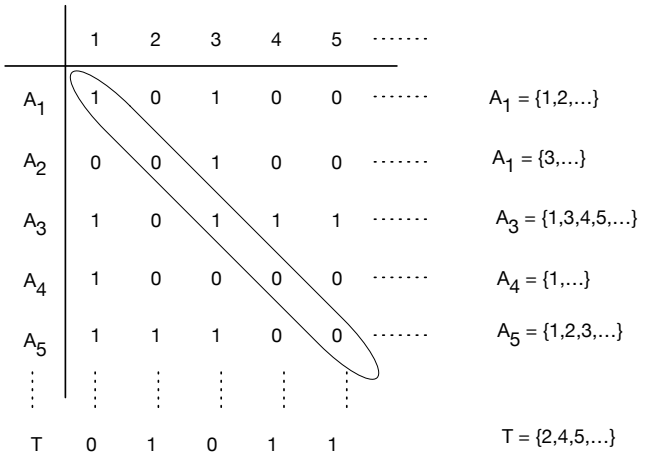

Figure 2: Diagonalization of a list of sets.

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

Fact 11. $\mathbb{R}$ *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, we shall see the proof next time.