Lecture 4: Counting and Diagonalization

Anup Rao

Apri 10, 2020

In today's lecture we discuss a matching upper bound and lower bound for boolean circuits. On the one hand, we shall prove that every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n/n)$, and on the other hand we show that for n large enough there is a function that cannot be computed by a circuit of size less than $2^n/(3n)$.

The lower bound—Counting arguments

The lower bound we prove here was first shown by Shanon. He introduced a really simple but powerful technique to prove it, called a *counting argument*.

Theorem 1. *For every large enough n, there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/3n$.*

Proof We shall count the total number of circuits of size s, where $s > n$. To define a circuit of size s, we need to pick the logical operator for each (non-input) gate, and specify where each of its two inputs come from. There are at most 3 choices for the logical operation, and at most s choices for where each input comes from. So the number of choices for each non-input gate is at most $3s^2$. The number of choices for an input gate is at most $n < 3s^2$. So, the total number of choices for each gate is at most $3s^2 + n$, and the number of possible circuits of size s is at most

$$(3s^2+n)^s \le (4s^2)^s = 2^{s\log(4s^2)} < 2^{3s\log s},$$

when $n > 4$.

This means that the total number of circuits of size $2^n/3n$ is less than $2^{3\cdot\frac{2^n}{3n}\cdot n} = 2^{2^n}$. On the other hand, the number of functions $f : \{0,1\}^n \to \{0,1\}$ is exactly 2^{2^n}. Thus, not all these functions can be computed by a circuit of size $2^n/(3n)$. ■

Indeed, the above argument shows that the fraction of functions $f : \{0,1\}^n \to \{0,1\}$ that can be computed by a circuit of size $2^n/4n$ is at most $\frac{2^{\frac{3}{4}\cdot 2^n}}{2^{2^n}} = \frac{1}{2^{2^{n-2}}}$, which is extremely small.

Similar arguments can be used to show that not every function has an efficient branching program (as you will do on your homework).

We can use a recursive construction to prove:

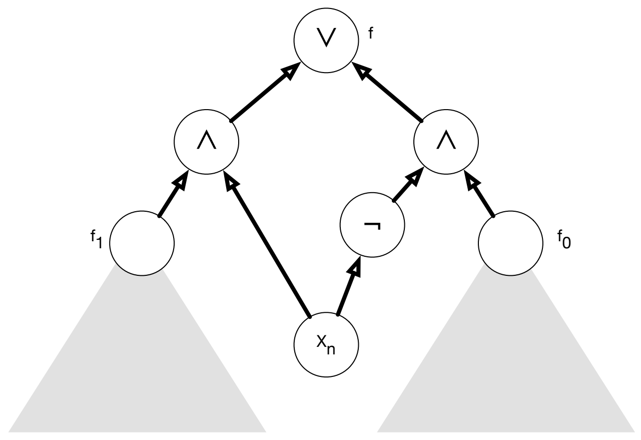


Figure 1: Recursive construction of a circuit for f.

Theorem 2. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n)$.*

Proof We construct the circuit recursively. When $n = 1$, there is clearly a constant sized circuit that computes f, since f must be either a constant, x_1 or $\neg x_1$.

For $n > 1$, let f_0 denote the function on $n-1$ bits given by $f_0(x) = f(x,0)$, and $f_1(x) = f(x,1)$. Then by induction we can compute f_0, f_1 recursively, and combine them using the value of the last bit to obtain f, as in Figure 1. When $x_n = 1$, the circuit outputs $f_1(x_1, \ldots, x_{n-1})$, and when $x_n = 0$, the circuit outputs $f_0(x_1, \ldots, x_{n-1})$.

If S_n is the size of the resulting circuit when the underlying function takes an n bit input, we have proved that

$$S_n \leq 2S_{n-1} + 5.$$

Expanding this recurrence, and using the fact that $S_1 \leq 5$, we get that

$$S_n \leq \sum_{i=1}^{n} 2^i 5 = 5 \cdot (2^{n+1} - 1) < 10 \cdot 2^n,$$

where here we used the formula for computing the sum of a geometric series. ■

Finally, we add one more idea to shave off another factor of n:

Lupanov proved this.

Theorem 3. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed in size $O(2^n/n)$ and depth $O(n)$.*

Proof We will use a recursive construction, but stop the recursion at a certain point. For a parameter t, we start by computing *every* function of the first t bits of the input. There are 2^{2^t} such functions, and each one can be computed by a circuit of size $O(2^t)$ using Theorem 2, so we can compute every function using at most $O(2^t \cdot 2^{2^t}) \leq O(2^{t+2^t})$ gates.

To compute the function f, we use the recursive construction defined in the proof of Theorem 2 for $n-t$ steps. After $n-t$ steps, we have put down $O(2^{n-t})$ gates, and need to compute functions on the first t bits, but since we have already computed every such function, we are done. The size of the final circuit is thus $O\left(2^{t+2^t}+2^{n-t}\right)$. Setting $t=\log n-1$, we get a circuit of size

$$O(2^{\log n-1+n/2}+2^{n-\log n+1}) \leq O(2^n/n).$$

■

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:

$\mathbb{N}=\{1,2,3,\dots\}$	the natural numbers
$\mathbb{Z}=\{\dots,-2,-1,0,1,2,\dots\}$	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
$\mathbb{R}=\left\{\lim\limits_{i\to\infty} x_i : x_1,x_2,\ldots\in\mathcal{Q} \text{ is a convergent sequence}\right\}$	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 4. *A set S is* countable, *if there is a surjective function ϕ : $\mathbb{N}\to S$.*

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.

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

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

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

Fact 6. $\mathbb{Z}$ *is countable.*

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

Fact 7. $\mathbb{Z} \times \mathbb{Z} = \{(i, j) : i, j \in \mathbb{Z}\}$ *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 2. 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. ■

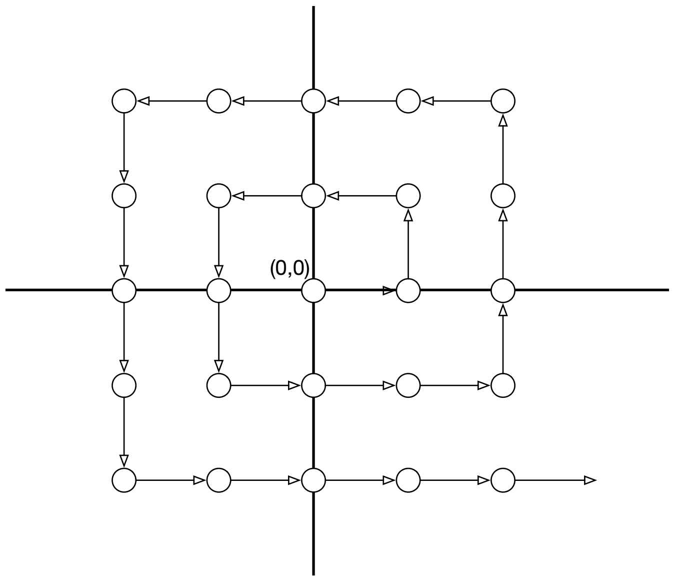


Figure 2: Enumeration of $\mathbb{Z} \times \mathbb{Z}$.

Fact 8. $\mathcal{Q}$ *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 9. $2^{\mathbb{N}}$ *is not countable.*

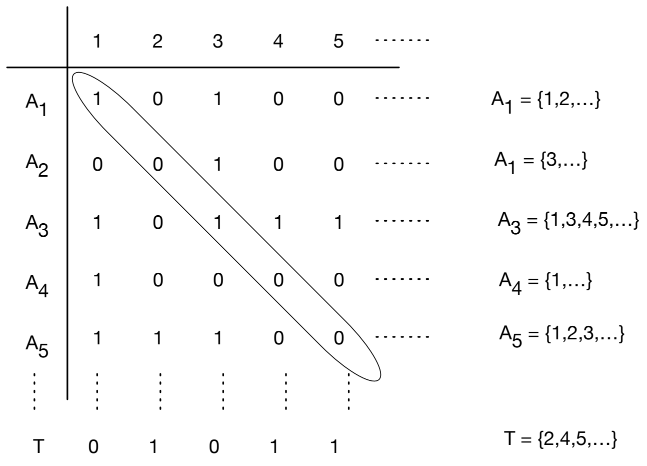
Proof Suppose there was some list of sets $A_1, A_2, \ldots$. Then consider the set

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

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 3. We encode each set in our list using a binary string. The set T

It was discovered by Cantor



1 2 3 4 5
A1 1 0 1 0 0 A1 = {1,2,...}
A2 0 0 1 0 0 A1 = {3,...}
A3 1 0 1 1 1 A3 = {1,3,4,5,...}
A4 1 0 0 0 0 A4 = {1,...}
A5 1 1 1 0 0 A5 = {1,2,3,...}
T 0 1 0 1 1 T = {2,4,5,...}

Figure 3: 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 real numbers are not countable:

Fact 10. $\mathbb{R}$ *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 $t = 0.d_1d_2\ldots$, 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 11. *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 12. 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 13. *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 14. *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.