CS 4510 Automata and Complexity April 10, 2024

Lecture 22: Circuit Complexity

Lecturer: Abrahim Ladha *Scribe(s): Rishabh Singhal*

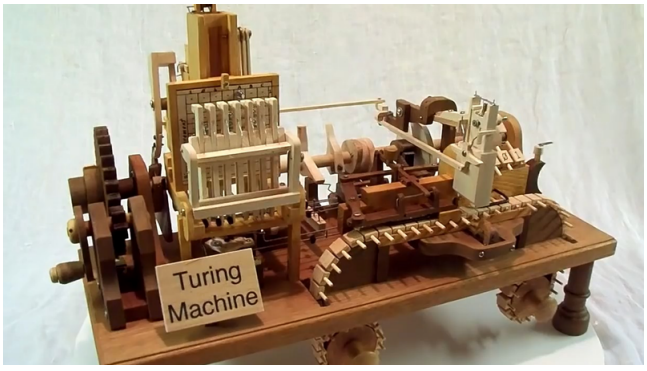
1 Review

Last time we proved that P vs NP has no relativizing proof. We would still hope to resolve the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question, so today, we are going to explore one direction into non-relativizing techniques.

2 Motivation from Hardware

First, why are black box techniques so common? Why were all the early proofs black-box anyway? My hypothesis is that even though a Turing machine is a simplification of computation to its bare essentials, its internals are still too complicated. There is some position of a tape head that moves conditionally on reading the tape. There is some transition function that may require a lookup, so a similar conditional read-move kind of device. It is not impossible to have a non-relativizing proof, using Turing machines simulation, but it appears to be unobvious what possible non-relativizing techniques may even exist for Turing machines.

Go to youtube and search for some homemade Turing machines. If the Turing machine is truly a foundational computer, there should exist some mechanical implementations. However, most of the builds implement a Turing machine on top of some other computational model. These are mostly raspberry-pi style projects. There was only a single mechanical Turing machine I could find which wasn't implemented on top of some other kind of computer. Surprisingly, it was made of wood!

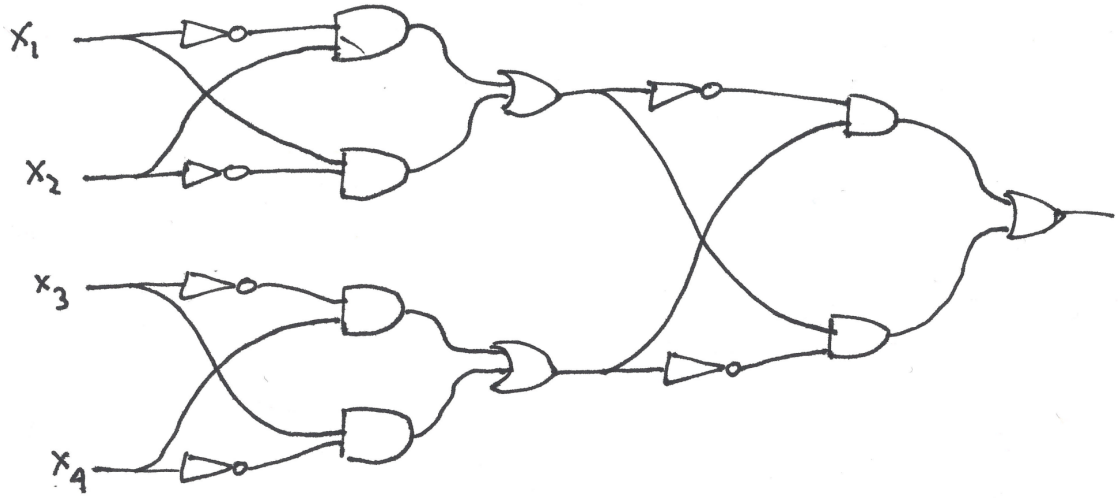


If you are looking for interesting novelty hardware computers, there appears to be many more builds which use boolean circuits as a foundation than Turing machines. There exists boolean circuit constructions from water and tubes, dominoes, marble machines, and so on.

Why are most of the computer builds with circuits and not with Turing machines? I thought the Turing machine was the foundational, most simplistic kind of computer? The answer is the same as why most proofs up to the relativization barrier were black box. The Turing machine has too many moving delicate parts. Even in minecraft, its easier to build a circuit using redstone then a conditionally moving tape head with pistons. What are the internals of a Turing machine? Its hard to say, its basically alive. What are the internals of a circuit? Just more circuits. Basically a glorified pachinko machine. You break a Turing machine into two, you have nothing. You break a circuit into two, you now have two circuits. The fact that the internals are simpler to inspect, this makes them an excellent candidate to sidestep the relativization barrier. It is not even clear how circuits could relativize. Where you could you even "stick the oracle" so to speak.

A second good reason to study circuit complexity is because of their combinatorial nature. Although mathematics has existed for millennia, non-trivial study of discrete mathematics is relatively new. There are far more tools available to those studying say, physics than there are of those studying complexity classes. We need to make use of all the tools available. Currently the only two domains of discrete math appear to be logic and combinatorics, and that is basically it. We need a formulation to be able to apply the combinatorial tools that we have developed.

3 Circuits



Definition 3.1 (Boolean Function)**.** A boolean function $f : \{0,1\}^n \to \{0,1\}$ has n input bits and one output bit.

A boolean function is just an association between the input and output. It is only the truth table.

Definition 3.2 (Boolean Circuit)**.** A boolean circuit is a directed acyclic graph where each node is assigned a gate chosen from some finite basis. Each gate has 2 fan-in and arbitrary fan-out. There are n input wires, and only one output wire. The wiring of gates is used to compute some boolean function.

While we are concerned with circuits which decide something, you may similarly define circuits which compute, and have several output wires. If a circuit has fan-out of one, it is called a formula.

Definition 3.3 (Uniform and non-uniform model of computation)**.** We say a model of computation is uniform if it's devices accept input of any arbitrary size. We say a model of computation is non-uniform if it may only accept inputs of a fixed size. For each possible input, there is a possibly different instance of the machine to handle that case.

Uniform models of computation include Turing machines, even DFA's and PDA's. Circuits however have a fixed input size. A 32-bit adder will safely and correctly add inputs of less than 32 bits, but not more. How can we define a circuit to decide a language?

Definition 3.4 (Circuit Family)**.** A circuit family is a collection $\{C_0, C_1, C_2, ...\}$ where each C_n is a circuit with n input wires and one output wire. We say a circuit family $\{C_0, C_1, ...\}$ decides some language L if

$$w \in L \iff C_{|w|}(w) = 1 \tag{1}$$

Importantly, this definition of computation being non-uniform separates measures of circuit complexity from the complexity to synthesize the circuits.

How should we measure the complexity of a circuit family? The complexity of a circuit family is not something related to time but to the size of the circuit, the number of gates as a function of n. There are other measures, width and depth and so on, but the most natural one is of course the number of gates.

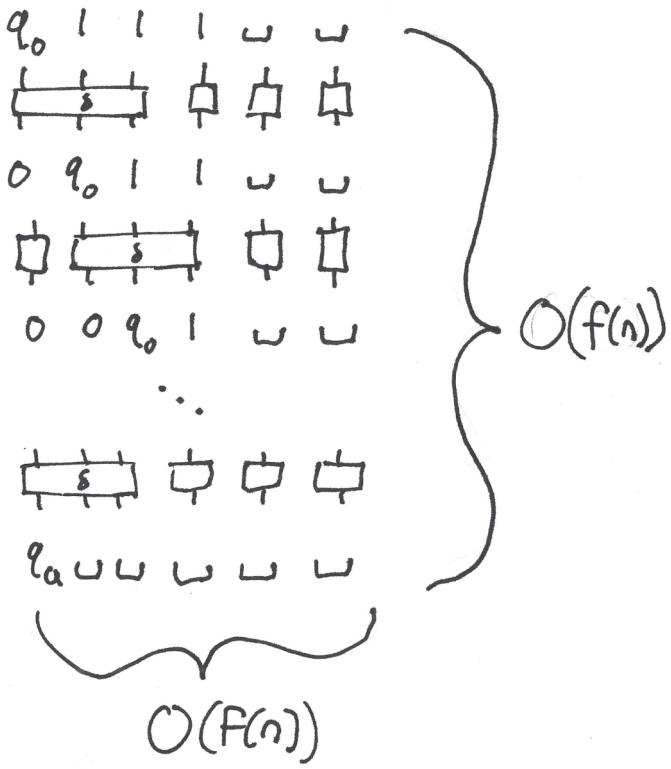
Definition 3.5 (Circuit Complexity)**.** We let the class $\mathsf{SIZE}(f(n))$ denote languages decidable by circuit families of size $f(n)$. The number of gates of $C_n \leq O(f(n))$ for each C_n.

Surely this is not so different from time complexity. You might think that $\mathsf{SIZE}(f(n)) \subseteq \mathsf{TIME}(f(n))$, but lets see what happens. Let $L \in \mathsf{SIZE}(f(n))$, then L has an $f(n)$ sized circuit family. To build a Turing machine to accept $w \in L$, we simply need to simulate the circuit $C_{|w|}$. Each gate takes constant time so this decides for L in time $f(n)$ so $L \in \mathsf{TIME}(f(n))$. The problem is you cannot encode infinitely many boolean circuits into a constant-sized machine. What if you could compute the circuits? This is our second roadblock, we made no mention of the fact for any circuit family that the function $f : n \to C_n$ need to be computable. We will elaborate on this later. Instead, to relate non-uniform circuit size to uniform time, we constructively prove the following.

Theorem 1.

$$\mathsf{TIME}(f(n)) \subseteq \mathsf{SIZE}(f^2(n)) \tag{2}$$

Proof. We proceed by conversion of computation history of a machine that runs in time $f(n)$ to a circuit of size $f^2(n)$. The proof is basically identical to the Cook-Levin theorem without any polynomial restriction. First, convert $\Gamma \cup Q$ to binary, perhaps like $\{0, 1, \sqcup, q_0, q_h, \} \to \{000, 001, 010, 100, 101\}$. We arrange the configurations sequentially into rows in a table. In between each row, we insert gates which take as input the previous configuration and output the next one.



The tape, the sequential state updates of the machine during its execution, does not seem to appear anywhere. It has not vanished, it is encoded in the intermediary wires! This circuit exists to correctly simulate some fixed M on w. but w is provided on the input wires in a suitable encoding. Since M runs in $f(n)$ time, it can also use at most $f(n)$ space. The height of this circuit is the time, and the width is the space. So the number of gates is $c \cdot f(n) \cdot f(n)$, with the most gates being identity ones, but the others taking a constant more than one to be represented in a reasonable circuit basis. Either way, for $L \in \mathsf{TIME}(f(n))$, we see $L \in \mathsf{SIZE}(f^2(n))$ completing the proof. □

Note that this could be improved to build a circuit of size $f(n) \log f(n)$ with a more careful construction. Either way, we see that not only are circuit families Turing-complete, they are also quite efficient! Efficient computations have efficient circuit families.

4 Turing Machines which take Advice

Definition 4.1. For any class C and function f, we use C/f to denote the class of languages decidable by C-machines given access to $f(n)$ bits of advice. There exists a second tape in which some string of answers is prewritten. The C machine may read from this tape to "take advice". The advice may not be a function of the input w, only the inputs length n. There are no other restrictions on it. It need not even be computable.

Observe the following:

- $\mathsf{C}/0 = \mathsf{C}$

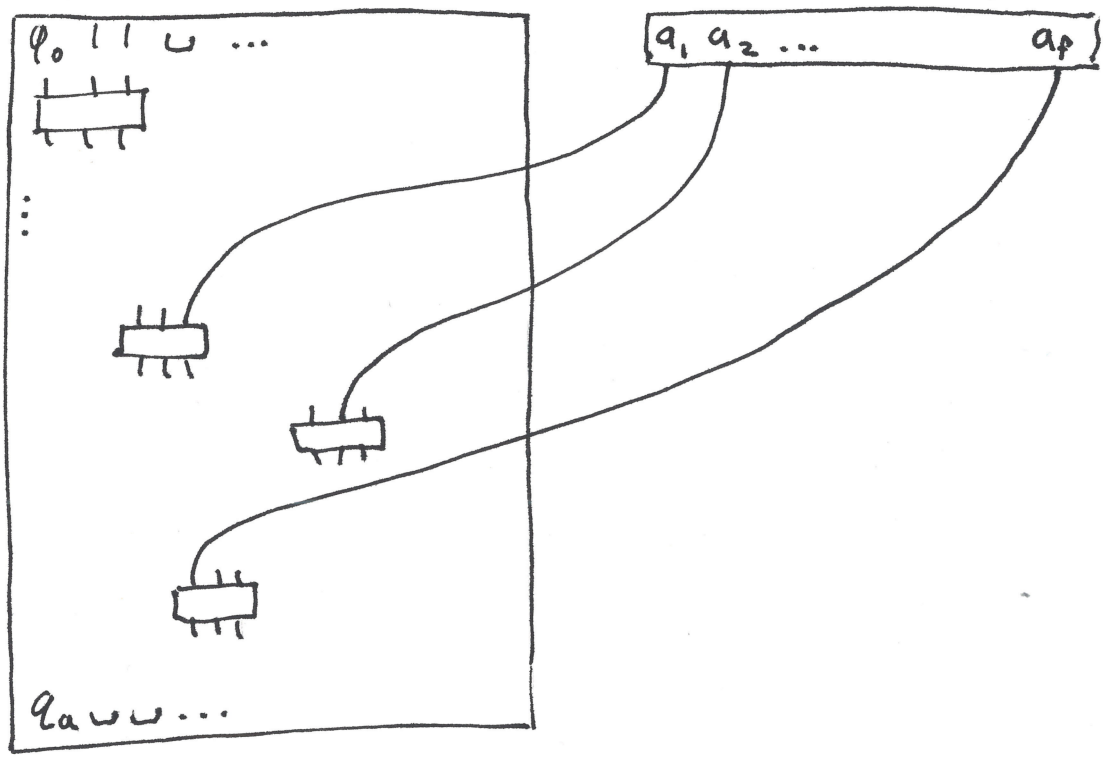
- If $f \leq g$ then $\mathsf{C}/f \subseteq \mathsf{C}/g$
- $\mathcal{P}(\Sigma^*) \subseteq \mathsf{P}/2^n$ For $L \subseteq \Sigma^*$ and $w \in L$, the advice of length 2^n provided to the machine will be which strings of length n are in or not in L. Given this advice, the $\mathsf{P}/2^n$ will simply read the correct bit and accept/reject appropriately.
- If f contains a one infinitely often, perhaps is the characteristic string of some undecidable language, then C/f may contain undecidable languages.

5 P/poly

We denote P/poly as the class of languages decidable by a Turing Machine which halts in polynomial time given access to a polynomial amount of advice. It turns out that P/poly is also exactly the class of languages that have polynomial-sized circuit families. Let's prove it.

Theorem 2. $\mathsf{P}/\mathsf{poly} = \mathsf{SIZE}(poly)$

Proof. We proceed by a double set containment. Let $L \in \mathsf{P}/\mathsf{poly}$. We show there exists a polynomial-sized circuit family to decide L. Since $L \in \mathsf{P}/\mathsf{poly}$ there exists a polynomial time Turing Machine with access to polynomial advice. Convert the polynomial machine to a polynomial-sized circuit, and simply wire in the advice, perhaps at the depth of the input. This resulting circuit is still of polynomial size so we observe that L has a polynomial-sized circuit family and $\mathsf{P}/\mathsf{poly} \subseteq \mathsf{SIZE}(poly)$.



Let L be decidable by a polynomially sized circuit family, so $L \in \mathsf{SIZE}(poly)$. Then there exists a polynomial sized circuit for each input size. The $\mathsf{P/poly}$ machine on input of length n is provided with advice C_n. The polynomial sized advice provided to the machine will simply be the circuit! The advice machine simply simulates its input on the polynomial sized circuit and outputs appropriately. This takes polynomial time so $L \in \mathsf{P/poly}$. We observe that $\mathsf{SIZE}(poly) \subseteq \mathsf{P/poly}$.

□

From here on, we may simply refer to $\mathsf{P/poly}$ as languages with polynomial-sized circuit families since we care about those more than machines which take advice.

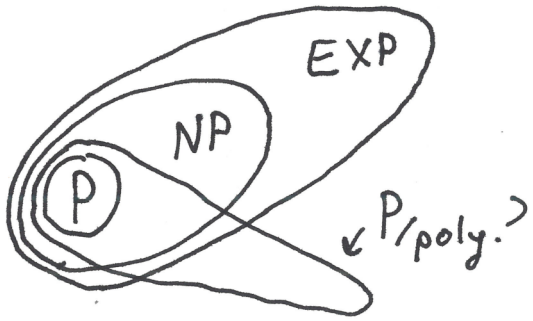
Theorem 3. $\mathsf{P} \subsetneq \mathsf{P/poly}$

Proof. We prove the containment using both definitions. It is true that $\mathsf{P} \subseteq \mathsf{P/poly}$ by the advice definition since a machine may simply ignore its advice. It is also true by the circuit definition following the Cook-Levin style construction we did earlier.

Now we describe why this containment is strict. There is no requirement that the function $f(n) = C_n$ need be computable at all. It is known, but we won't prove, that all unary languages have polynomial sized circuit families. This includes even some undecidable ones. Consider the language $UHALT = \{1^{\langle M,w \rangle} | M \text{ halts on } w\}$. It is undecidable by a simple reduction, yet has a polynomial sized circuit family. Since P can only contain decidable languages, the containment $\mathsf{P} \subsetneq \mathsf{P/poly}$ must be strict.

□

It is hard to represent a venn diagram relationship among uniform and non-uniform classes, but we could still try.



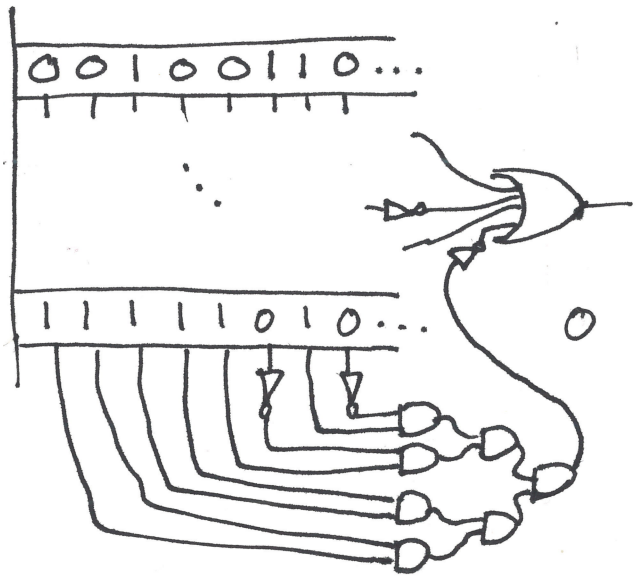
6 Circuit Upper Bounds

Given a specific boolean function, we are concerned with the smallest circuit which can compute the boolean function.

Theorem 4. Every boolean function $f : \{0,1\}^n \to \{0,1\}$ has a circuit of size $O(n2^n)$.

Proof. Encode the $2^n \times n$ truth table into a circuit in the following way. For each row, add a tree of $O(n)$ and-gates. Add $O(n)$ not-gates on the appropriate wires. With one such tree

per row, you will have 2^n output wires. Exactly and only one of these wires will light up for the 2^n ways to have n inputs. Now add a tree of at most $O(2^n)$ or-gates, only wiring in the rows in which the boolean function outputs a 1. If the boolean function outputs a 0, all the wires connected to the last or-gate tree will be 0 and the circuit will output a 0. If the boolean function outputs a 1, then exactly one wire on the last or-gate tree will be a 1 and the circuit outputs 1. We see then the circuit computes the boolean function.



This circuit takes a linear amount of gates per each row giving us a circuit of size $O(n2^n)$. □

Can this be improved? Lupanov in 1952 was able to get a recursive representation for boolean functions using only $O(2^n/n)$ gates. This is still quite high. Can we really not do any better?

7 Circuit Lower Bounds

This section should really be titled "the lack of circuit lower bounds". Given a specific ...

7.1 Shannon Muller Theorem

Although we don't have lower bounds for explicit boolean functions, we can determine that *most* boolean functions require exponential size circuits.

Theorem 5. Almost all boolean functions on $f : \{0,1\}^n \to \{0,1\}$ require circuits of size $\Omega(2^n/n)$

Proof. We prove this nonconstructively with a counting argument. We will compute the ratio of circuits of size $2^n/n$ to n-bit boolean functions. We will show there are way more boolean functions, and that with the limit $n \to \infty$ this ratio would converge to zero.

How many boolean functions are there? A boolean function on n inputs may be characterized by its truth table, the last column of length 2^n, there are $2^{(2^n)}$ possible truth tables and therefore, exactly this many boolean functions.

How many circuits of size $2^n/n$ are there? First we compute this number for an unspecified number t, then later evaluate it for $t = 2^n/n$, just to make the computation simpler. Suppose we have t gates and n inputs. Let us inspect some gate.

If B is the circuit basis, then there are $|B|$ choices for the gate itself. Its inputs could come from another gate or an input, so there are also $(t + n - 1)$ possible choices for each input. We may upperbound the number of circuits of size t by

$$|B|^t((t+n-1)^2)^t$$

This overestimates quite a bit. There are many equivalent circuits this upper bound counts as distinct, simply if you could consider the gates in a different order. We must divide by the number of ways you could put the t gates into an order, so

$$\frac{|B|^t(t+n-1)^{2t}}{t!}$$

You may recall Stirling's approximation.

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

We replace $t!$ by this approximation to get

$$\frac{|B|^t(t+n-1)^{2t}e^t}{t^t\sqrt{2\pi t}}$$

Note that $n - 1 \leq t$ since there are n inputs, there should be atleast $n - 1$ gates. Each input should be used, otherwise consider a smaller n. We may include this upperbound in our calculation.

$$\frac{|B|^t(t+n-1)^{2t}e^t}{t^t\sqrt{2\pi t}} \leq \frac{|B|^t(2t)^{2t}e^t}{t^t\sqrt{2\pi t}} = \frac{(4e|B|t)^t}{\sqrt{2\pi t}} < (ct)^t$$

for some constant c. Now we may evaluate at $t = 2^n/n$

$$(ct)^t = \left(\frac{c2^n}{n}\right)^{2^n/n} = \left(\frac{c}{n}\right)^{2^n/n}(2^n)^{2^n/n} = \left(\frac{c}{n}\right)^{2^n/n}2^{(2^n)}$$

We may now compute the ratio

$$\frac{\#\text{circuits of size } 2^n/n}{\#\text{ boolean functions of } n \text{ inputs}} < \frac{\left(\frac{c}{n}\right)^{2^n/n}2^{(2^n)}}{2^{(2^n)}} = \left(\frac{c}{n}\right)^{2^n/n}$$

As we take the limit of n note that $2^n/n$ will get very very large, and c/n will get very very small. We see then that the limit of our ratio must go to zero. We may then conclude there are far more boolean functions than there are circuits of size $2^n/n$ so most boolean functions should require circuits larger than or equal to this and the result is proven. □

We may combine this with the Lupanov representation to get that most boolean functions have circuit complexity $\Theta(2^n/n)$. We can prove nonconstructively that most boolean functions require large circuits, yet we cannot prove for any specific boolean function that it even requires circuits of size $\Omega(n)$.

8 Natural Proofs Era

[TBD]