**CS 4510 Automata and Complexity** April 19, 2023

# Lecture 19: P/poly

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

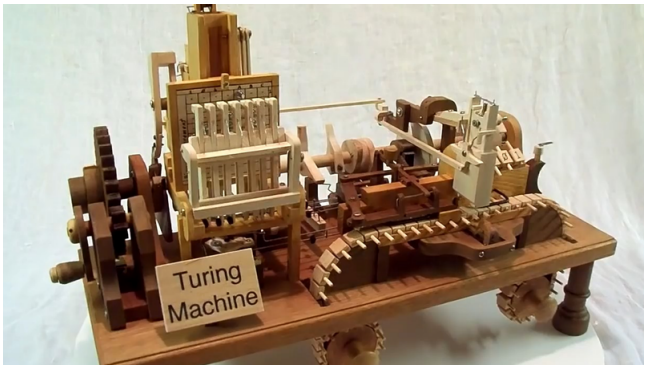
## 1 Review

Last time we proved that P vs NP has no relativizing proof. 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? 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 base essentials, it is still too complicated. There is some state of a moving tape head that moves conditionally are reading the tape. There is some transition function that may requires a lookup, so a similar conditional read-move kind of device. It is not impossible to have a non-relativizing proof, using Turing Machines, but it appears to be unobvious.

Go to youtube and search for some homemade Turing-machines. If the Turing machine is truly a foundational computer, there should exist people building them. However, most of the builds implement a Turing machine on top of some other computer! 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 hardware computers, there appears to be a million builds which use boolean circuits as a foundation instead of a Turing machine. 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.

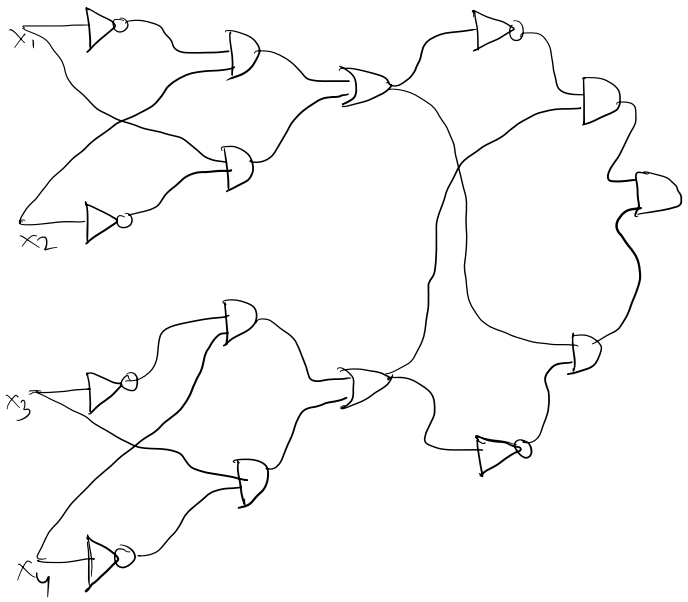



Figure 1: Circuit

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.

## 3 Circuits

A boolean circuit is a wiring of gates that is used to compute a boolean function $\{0,1\}^n \to \{0,1\}$. We say a model of computation is uniform if it's devices accept input of any arbitrary

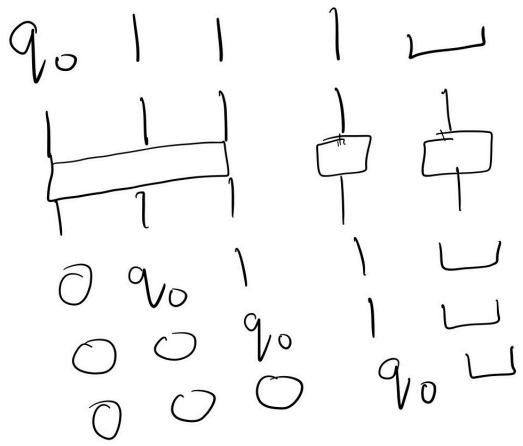


Figure 2:

size. These include TM's, 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. A circuit family is a collection $\{C_0, C_1, C_2, ....\}$ where each $C_i$ is a circuit with i input wires and one output wire. We say a circuit family $\{C_0, C_1, ...\}$ decides some language $L$ if

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

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 $n$. We let the class $\mathsf{SIZE}(f(n))$ denote languages decidable by circuit families of size $f(n)$. Surely this is not so different from time complexity. You might think that $\mathsf{SIZE}(f(n)) \subseteq TIME(f(n))$, 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 $L \in TIME(f(n))$. 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 $f(n) \to C_n$ need to be computable. In fact, the language HALT in unary HALT $= \{1^{\langle M,w \rangle} | M \text{ halts on } w\}$ has a polynomial-sized circuit family since as it turns out, all unary languages have polynomial-sized circuit families. We will elaborate on this later.

Instead, for a more constructive proof, we prove

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

## 4 Cook-Levin

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 $\{0, 1, q_0, w\} \to \{00, 11, 10, 01\}$. We add in a bunch of these gates to represent the transition from now to now. 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 $L \in TIME(f(n))$, we see $L \in \mathsf{SIZE}(f^2(n))$ completing the proof. Note that this could be improved to build a circuit of size $f(n) \log f(n)$. Either way, we see that not only are circuit families Turing-complete, they are also quite efficient!

## 5 Turing Machines which take Advice

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 sequentially read from this tape to "take advice". Observe the following:

- $C/0 = C$
- If $f < g$ then $C/f < C/g$
- $\mathcal{P}(\Sigma^*) \subseteq \mathsf{P}/2^n$
- If $f$ contains a one infinitely often, perhaps is the characteristic string of some undecidable language, then $C/f$ may contain undecidable languages.

## 6 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.

Let $L$ be decidable by a polynomial-sized circuit family then there exists a polynomial-sized circuit for each input size. Choose a description of this circuit to be the advice. The $M$ on input $w$ simply simulates w on $C_n$. This takes polynomial time so $L \in \mathsf{P}/\mathsf{poly}$.

Let $L \in \mathsf{P}/\mathsf{poly}$. We show there exists a polynomial-sized circuit family to be decidable $L$. 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 hard

code the advice, perhaps at the depth of the input. This resulting circuit is still polynomial sized so we observe that $L$ has a polynomial-sized circuit family.

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.

It is true that $\mathsf{P} \subseteq \mathsf{P/poly}$ by the advice definition. Note that this containment is strict only since we allow non-computable circuit families. We could prove all unary languages have polynomial-sized circuits, including the undecidable ones previously mentioned. Since $\mathsf{P}$ can only contain decidable languages, the containment $\mathsf{P} \subsetneq \mathsf{P/poly}$ must be strict.