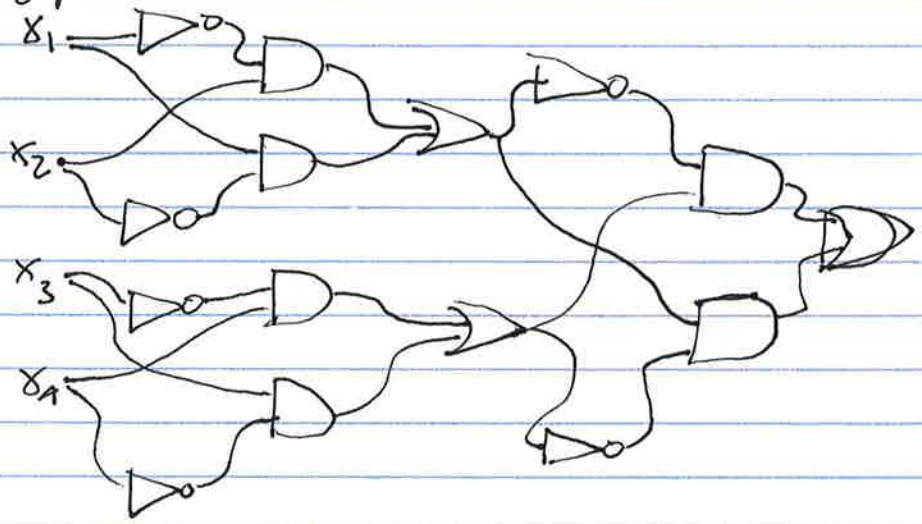
Abrahim Ladha

P/poly

21

Last time we proved that P vs NP has no relativizing proof. Today, we are going to explore how we might want to consider non-relativizing techniques. First, why are black box techniques so seductive? Why were all the early proofs black box anyway? My hypothesis is that even though a Turing machine is a simplification of a computation to its bare essentials, it is still too complicated. There is some state, a moving tape head which moves conditionally on reading the tape. There is some transition function which maybe 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 must be quite difficult.

Enter circuits, stage right. Circuits are so much simpler than a Turing machine from a hardware point of view. The main reason is they have no moving parts.



This circuit computes the parity function on four bits. A boolean circuit is a wiring of gates of fan-in two and unlimited fanout to compute a boolean function $\{0,1\}^n \to \{0,1\}$

We say a model of computation is uniform if its devices accept input of arbitrary size. These include TMs, even DFAs and PDAs. Circuits are fixed, they have a fixed input size, mostly. A 32-bit adder will safely and correctly add inputs of < 32 bits, but not more. Instead to use circuits as a model of computation, we define them to be non-uniform. A circuit family is a collection $\{C_0, C_1, \ldots\}$ where each C_i is a circuit with i input wires. We say a circuit family decides some language L if

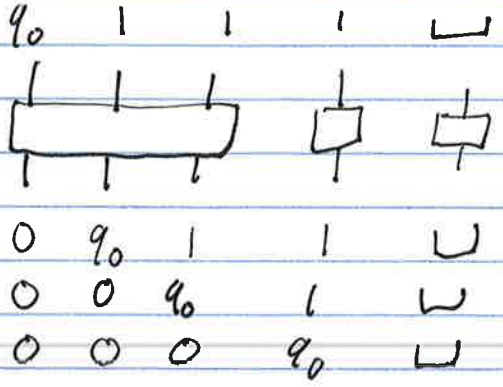
$$w \in L \iff C_{|w|}(w) = 1$$

You pick the circuit of the correct size to run it on. We measure the complexity of a circuit family not as the time of anything, but as the size of the circuit, the number of gates as a function of n. We let the class $CC(f(n))$ denote languages decidable by circuit families of size $f(n)$. Surely this is not so different from time complexity. Is it the case that $CC(f(n)) \subseteq TIME(f(n))$? Here are roadblock. Let $L \in CC(f(n))$, then L has an $f(n)$-size circuit family. So to build a Turing machine to accept $w \in L$, it only need to execute the circuit $C_{|w|}$, where each gate takes constant time so this decider for L takes time $f(n)$ showing $L \in TIME(f(n))$. The problem is you cannot encode infinitely many boolean circuits into a constant sized machine. What if you could compute the circuits? 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 $HALT = \{1^{\langle M,w \rangle} \mid M \text{ halts on } w\}$ has a polynomial sized circuit family since as it turns out, all unary languages have polynomial sized circuit families.

Instead, for a more instructive proof, we prove

$$TIME(f(n)) \subseteq CC(f^2(n))$$

We proceed by converting a computation history of a machine which 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 $\{0, 1, q_0, \sqcup\} \to \{00, 11, 10, 01\}$.



We add in a bunch of these gates to represent the transition from row to row. The tape, the sequential state updates of the machine during its execution, is not nowhere. It has not vanished, neither, it is encoded in the intermediary wires! This circuit exists to correctly simulate some fixed M on w, but w is provided as 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 table is the time, and the width is the space, so the number of gates is $cf(n) \cdot f(n)$, with most gates being identity ones, but the  taking a constant amount more than one to be represented in a reasonable circuit basis. Either way, for $L \in TIME(f(n))$, we see $L \in \mathcal{CC}(f^2(n))$ completing the proof.

For any class C and function f, we use the notation C/F to denote the class of languages decidable by C-machines given access to $f(n)$ bits of advice. There is a second tape in which some string is pre-written, in which the C machine may sequentially read from. Observe that $C/0 = C$, if $f \leq g$ then $C/f \subseteq C/g$. If $f = 1$ infinitely often, it may be the case that $HALT \in P/f$ where f is just the characteristic string of Halt.

We denote P/poly as Turing machines which halt in polynomial time given access to polynomial amount of advice. It turns out that P/poly is also exactly the class of languages which have polynomial sized circuit families! Lets prove it.

$\Leftarrow$ 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 machine on input w simply simulates w on C_n. This takes polynomial time so $L \in P/poly$

$\Rightarrow$ Let $L \in P/poly$. We show there exists a polynomial sized circuit family to decide L. Since $L \in P/poly$ there exists a polynomial time Turing machine with access to polynomial advice. Convert the polytime 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 P/poly as languages with polynomial sized circuit families since we care about those more.

It is true that $P \subseteq P/poly$ by using the advice definition. It is also true using the circuit definition, since if a P machine runs in $f(n)$ time, there exists a $f^2(n)$ sized circuit family. This actually can be improved to $f(n)\log f(n)$ sized circuit family. Note that this containment is strict only since we allow non computable circuit families. We could prove all crazy languages have polynomial sized circuits, including the undecidable ones mentioned previously.