PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

Lecture 22: Natural Proofs

Lecturer: *Sanjeev Arora* Scribe:*Iannis Tourlakis*

Why have we not been able to prove strong lower bounds for circuits? In this lecture we consider the notion of "natural proofs" for separating complexity classes as put forth by Razborov and Rudich in 1994. In their paper, Razborov and Rudich show that current lowerbound arguments involve "natural" techniques, and show that obtaining strong lowerbound with such techniques would violate a widely believed cryptographic assumptions.

Basically, a natural technique is one that proves a lowerbound for a random function and is "constructive." We will formalize "constructive" later but let us consider why current techniques apply to random functions.

1 Formal Complexity Measures

We begin with an example.

EXAMPLE 1 Recall that a formula is a circuit with one output and in which each gate has in-degree 2 and outdegree 1. How would one go about proving lower bounds on formula size? Perhaps by induction? To that end, suppose we have a function like the one in Figure 1 that we believe to be "hard". If the function computed at the output is "complicated", intuitively it should be that at least one of the functions on the incoming edges to the output should also be complicated.

Let's try to formalize our intuition in the above example. Let μ be a function that maps each boolean function on $\{0,1\}^n$ to a nonnegative integer. (The input to μ is the truth table of the function.) We say that μ is a *formal complexity measure* if it satisfies the following properties: First, $\mu(x_i) \leq 1$ and $\mu(\bar{x}_i) \leq 1$ for all i. Second, we require that

- $\mu(f \wedge g) \leq \mu(f) + \mu(g)$ for all f, g; and
- $\mu(f \vee g) \leq \mu(f) + \mu(g)$ for all f, g.

For instance, the following function ρ is trivially a formal complexity measure

$$\rho(f) = \text{the smallest formula size for } f. \tag{1}$$

In fact, it is easy to prove the following by induction.

THEOREM 1
If μ is a formal complexity measure, then $\mu(f)$ is a lowerbound on the formula complexity of f.

Thus to prove a lowerbound for the formula size complexity of the CLIQUE, we would define a measure μ that is high for CLIQUE. For example, one could try "fraction of inputs for which the function agrees with the CLIQUE function" or some suitably scaled version

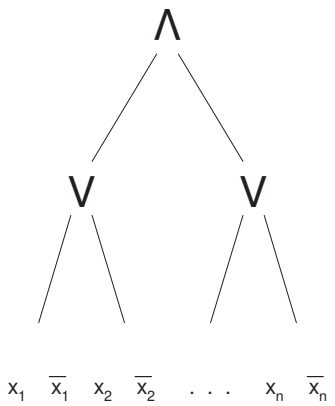

Figure 1: A formula for a hard function.

of this. Unfortunately, such simple measures don't do the job, as you should check using the next few paragraphs. In general, one would imagine that a measure that lets us prove a good lowerbound for CLIQUE would involve some deep theorem about the CLIQUE function. The next lemma seems to show, however, that even though all we care about is the CLIQUE function, we have to in fact reason about random functions.

Lemma 2
Suppose that μ is a formal complexity measure and that there exists a function $f : \{0,1\}^n \to \{0,1\}$ such that $\mu(f) \geq c$ for some large number c. Then for at least $1/4$ of all functions $g : \{0,1\}^n \to \{0,1\}$ we must have that $\mu(g) \geq c/4$.

Proof: Let $g : \{0,1\}^n \to \{0,1\}$ be random. Write f as $f = h \oplus g$ where $h = f \oplus g$. Note that h is also random. So $f = (\bar{h} \wedge g) \vee (h \wedge \bar{g})$. Now suppose that $\{g : \mu(g) < c/4\}$ contains more than $3/4$ of all functions. Then $\Pr[\text{All of } h, \bar{h}, g, \bar{g} \text{ have } \mu < c/4] > 0$. Hence $\mu(f) < c$, which contradicts the assumption. Thus the lemma is proved. □

In fact, the following stronger theorem holds:

Theorem 3
If $\mu(f) > c$ then for all $\epsilon > 0$ and for at least $1 - \epsilon$ of all functions g we have that,

$$\mu(g) \geq \Omega\left(\frac{c}{(n + \log(1/\epsilon))^2}\right).$$

The idea behind the proof of the theorem is to write f as the boolean combination of a small number of functions and then proceed similarly to the proof of the lemma.

2 Natural Properties

A *property* Φ is a map from boolean functions to $\{0,1\}$. A **P**-*natural property useful against* **P**/*poly* is a property Φ such that:

1. $\Phi(f) = 1$ for at least a $1/2^n$ fraction of all functions on n bits (recall that there are 2^{2^n} functions on n bits);

2. $\Phi(f) = 1$ implies that $f \notin \mathbf{P}/poly$ (or concretely, say, f has circuit complexity $n^{\log n}$); and

3. Φ is computable in $2^{O(n)}$ time.

Note that the property is called **P**-natural because of requirement (3). The property is useful against $\mathbf{P}/poly$ because of requirement (2). Note that requirement (3) is somewhat controversial in that there is no broad consensus on whether it is reasonable.

EXAMPLE 2 We have seen a natural property useful against $\mathbf{AC}^0$: this is the property that there does not exist a restriction on $n - n^\epsilon$ inputs function that turns the function into a constant function.

In fact, all known combinatorial techniques for obtaining circuit lowerbounds essentially involve constructing a natural property useful against the circuit class in question. In the case of $\mathbf{AC}^0$, the natural property is quite explicit in the proof, whereas for other classes, the natural property is hidden deeper inside the proof.

The following theorem by Razborov and Rudich possibly explains why we have not been able to use the same techniques to obtain an upper bound on $\mathbf{P}/poly$: constructing a natural property useful against $\mathbf{P}/poly$ violates widely believed cryptographic assumptions.

THEOREM 4 (RAZBOROV, RUDICH)
Suppose there exists a **P***-natural property* Φ *useful against* $\mathbf{P}/poly$*. Then there are no strong pseudorandom function generators. In particular, there exist subexponential algorithms for factoring and discrete log.*

Before giving the proof we define pseudorandom function generators. Note that we will be tailoring a more general definition for our narrow purposes. We begin by defining pseudorandom function ensembles (again tailored to our needs).

DEFINITION 1 *A* function ensemble *is a sequence* $F = \{F_n\}_{n=1}^{\infty}$ *of random variables where* F_n *assumes values in the set of functions mapping* $\{0,1\}^n$ *to* $\{0,1\}$*. In the special case where* F_n *is uniformly distributed over all such functions,* F *is called the* uniform function ensemble *and is denoted by* $H = \{H_n\}_{n=1}^{\infty}$*. A function ensemble* $F = \{F_n\}_{n=1}^{\infty}$ *is* pseudorandom *if for each Turing machines* M *running in time* $2^{O(n)}$*, and for all sufficiently large* n*,*

$$|\Pr[M(F_n) = 1] - \Pr[M(H_n) = 1]| < \frac{1}{2^{n^2}}.$$

DEFINITION 2 *A* pseudorandom function generator *is a function* $f(k,x)$ *computable in polynomial time where the key* k *has* n^c *bits and the input* x *has* n *bits with the following property: if we let* F_n *be the random variable defined by uniformly selecting* $k \in \{0,1\}^{n^c}$ *and setting* F_n *to* $f(k,\cdot)$*, then the function ensemble* $F = \{F_n\}_{n=1}^{\infty}$ *is pseudorandom. We will denote* $f(k,\cdot)$ *by* f_k*.*

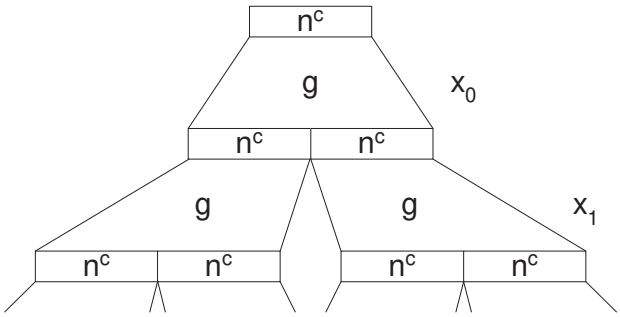

Figure 2: Constructing a pseudorandom function generator from a pseudorandom generator.

Intuitively, the above definition says that given a pseudorandom function generator f, if we fix k randomly, then with high probability the function f_k "looks like a random function" to all Turing machines running in time $2^{O(n)}$. Note that it takes $2^{O(n^c)}$ time to guess a key whereas the adversary Turing machine is only allowed time $2^{O(n)}$.

As an aside, note that we can construct a pseudorandom function generator $f(k, x)$ using a pseudorandom generator g that stretches n^c random bits to $2n^c$ pseudorandom (also see Figure 2): Let $g_0(k)$ and $g_1(k)$ denote, respectively, the first and last n^c bits of $g(k)$. Then the following function is a pseudorandom function generator, where $\mathrm{MSB}(x)$ refers to the first bit of a string x:

$$f(k, x) = \mathrm{MSB}(g_{x_n} \circ g_{x_{n-1}} \circ \cdots \circ g_{x_2} \circ g_{x_1}(k)).$$

This construction is correct if the security parameter of g is large enough. Such a pseudorandom generator exists —by the Goldreich Levin theorem— if a oneway permutation exists that is extremely hard to invert. The discrete log problem —a permutation— is conjectured to be oneway. Many researchers believe that there is a small $\epsilon > 0$ such that the discrete log problem cannot be solved in 2^{n^ϵ} time. (They also believe the same for factoring.) If this belief is correct, then pseudorandom function generators exist as outlined above. (Exercise.)

We can now give the proof of the above theorem:

Proof:[Theorem 4] Suppose the property Φ exists, and f is a pseudorandom function generator. We show that a Turing machine can use Φ to distinguish f_k from a random function. First note that $f_k \in \mathbf{P}/poly$ for every k (just hardwire k into the circuit for f_k) so the contrapositive of property (2) implies that $\Phi(f_k) = 0$. In addition, property (1) implies that $\Pr_{H_n}[\Phi(H_n) = 1] \geq 1/2^n$. Hence,

$$\Pr_{H_n}[\Phi(H_n)] - \Pr_{k \in \{0,1\}^{n^c}}[\Phi(f_k)] \geq 1/2^n,$$

and thus Φ is a distinguisher against f. □

We finish by noting that there is one technique that we have seen for proving lower bounds that does *not* fall under the category of natural proofs: diagonalization. Sadly, as we saw in Lecture 3, diagonalization can only be used to prove relativizing results.