Lecture 15: Identity testing and the Permanent

Anup Rao

May 18, 2021

Last time, we saw how to prove the Schwartz-Zippel lemma, which allows us to give a simple randomized test to check whether or not a polynomial is 0: just evaluate the polynomial on a random point. This idea has many interesting consequences. Today, we begin by exploring some consequences for *arithmetic circuits*.

An arithmetic circuit is the same as a boolean circuit, except that the gates are replaced by arithmetic operations like $+$ and $\times$. Formally, the circuit is a directed acyclic graph. Every vertex in the graph corresponds to a variable x_i, or $+$ or $\times$, or the constants 0 or 1. The gates that correspond to $+$ or $\times$ have exactly two edges coming into them, and all other gates have no edges coming into them. The circuit is said to compute a polynomial $f(x_1, \ldots, x_n)$ if there is some gate in the circuit whose output corresponds to the polynomial f. Suppose we are given a circuit computing f and another circuit computing g, both of size at most s. How can we test whether or not $f = g$?

The Schwartz-Zippel lemma suggests a simple algorithm: let S be a large set of numbers, and pick $x_1, \ldots, x_n$ uniformly at random from S. Evaluate $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$. If the two polynomials are equal, these evaluations will be equal. If the two are different, we should expect to be able to use the lemma to argue that the evaluations will be different with high probability.

There are a couple of complications with getting this to work. First, to use the lemma, we need to bound the degrees of f, g. If f, g contain at most s multiplication gates, we see by induction that the degree of f, g can be at most 2^s, since each multiplication can potentially double the degree. So, to use the Schwartz-Zippel lemma, we need to use a set S with $|S| \gg 2^s$. This is not that big a problem: we can use the integers up to 2^{2s}. However, then we face another issue: the value of $f(x_1, \ldots, x_n)$ could be as large as $2^{(2s)^s}$. Thats because every multiplication gate could square the magnitude of the integers we are working with. Eventually, the integers will become so big that we need exponential space just to write them down, so the complexity of the algorithm becomes exponential in s.

To rescue this algorithm, we use arithmetic modulo a prime number. We need the following fact:

Fact 1. *Let $t(N)$ denote the number of primes in $\{1, 2, \ldots, N\}$. Then* $\lim_{N \to \infty} \frac{t(N)}{N/\ln N} = 1$.

Our final algorithm is as follows. Pick $S = \{1, 2, \ldots, 2^{2s}\}$ p to be a random prime in the set $\{1, 2, \ldots, 2^{s^2}\}$. Then pick the elements $x_1, \ldots, x_n$ from S at random and check that $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n) \mod p$.

If k distinct primes divide $f(x_1, \ldots, x_n)$, then we must have $2^{(2s)^s} \geq f(x_1, \ldots, x_n) \geq 2^k$, so $k \leq 2^s$. Since the number of primes in our set is $\Omega(2^{s^2}/s^2)$, we get that the probability that the random prime divides $f(x_1, \ldots, x_n)$ is at most

$$O((2s)^s/(2^{s^2}/s^2)) = O(2^{s\log(2s)-2^{s^2}}/s^2) \ll 1/10.$$

Determinant vs Permanent

Recall our definition of the determinant:

$$\det(M) = \sum_{\pi \in S_n} \mathsf{sign}(\pi) \prod_{i=1}^{n} M_{i,\pi(i)}.$$

The determinant can be computed in time $O(n^3)$ using Gaussian elimination. Faster algorithms are known as well. In particular, there is an algorithm to compute it using a circuit of polynomial size and depth $O(\log^2(n))$.

The determinant measures the volume of the parallelopiped generated by the rows of the matrix M.

The permanent has a very similar formula:

$$\mathsf{perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} M_{i\pi(i)}.$$

Surprisingly, the permanent seems much harder to compute. Indeed, there is a good reason for this. One can reduce 3SAT to computing the permanent!

When the matrix is the adjacency matrix of a directed graph, the permanent counts the number of *cycle covers* of the graph. A cycle cover of the graph is a collection of disjoint cycles that cover all the vertices of the graph. Try to prove this!

Let us define the complexity class #**P** as follows. We say that a function f is in #**P** if and only if there is a polynomial time turing machine M and a polynomial p such that

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x, y) = 1\}|.$$

Thus if one thinks of M as the verifier to an **NP** problem, f counts the number of witnesses to x. Examples of problems in #**P** include #SAT, where $\#\mathsf{SAT}(\phi)$ is the number of satisfying assignments to the boolean formulat ϕ.

We shall not prove the following theorem, but it shows that an efficient algorithm for the permanent would prove that **P** = **NP**.

Theorem 2. *Every f in* #**P** *can be reduced to* $\#\mathsf{SAT}(\phi)$ *in polynomial time. Every f in* #**P** *can be reduced to* perm *in polynomial time.*