1 AC_0, Probabilistic Polynomials, and Toda's Theorem (cont.)

In this lecture, we will complete the proof of Toda's theorem that $\mathsf{PH} \subset \mathsf{P}^{\#\mathsf{P}}$. Recall that, en route to Toda's result, we were trying to prove that AC_0 predicates could be computed by "probabilistic polynomials" of polylogarithmic degree using polylogarithmic random inputs.

Before proceeding, we recall the notation from last time and introduce some more notation. For $\overline{u}, \overline{v} \in \{0,1\}^m$, define

$$\delta_{\overline{u}}(\overline{v}) = \begin{cases} 1, & \text{if } \overline{u} = \overline{v} \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\delta_{\overline{u}}$ can be represented by a polynomial of degree m as follows:

$$\delta_{\overline{u}}(\overline{v}) = \prod_{i=1}^{m} (u_i v_i + (1-u_i)(1-v_i))$$

Last time we defined the Valiant-Vazirani function $\mathrm{VV}(\overline{i}, \overline{r})$, which can be written in terms of $\delta_{\overline{u}}$ functions as follows:

$$\mathrm{VV}(\overline{i}, \overline{r}) = \sum_{\overline{u}:\langle \overline{u}, \overline{i} \rangle = 0} \delta_{\overline{u}}(\overline{r}) = \begin{cases} 1, & \text{if } \langle \overline{r}, \overline{i} \rangle = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Thus, for a fixed $\overline{i}$, $\mathrm{VV}(\overline{i}, \overline{r})$ is a polynomial of degree m in $\overline{r}$.

We then defined functions S_j by

$$S_j(x_1, \ldots, x_n, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}) = \sum_{\overline{i}} x_{\overline{i}} \left(\prod_{k=1}^{j} \mathrm{VV}(\overline{i}, \overline{r}^{(k)}) \right).$$

In this definition, $n = 2^m$, and we consider the variables $x_1, \ldots, x_n$ to be labelled by vectors $\overline{i} \in \{0,1\}^m$. Notice that S_j is of degree at most m^2.

Finally, we defined functions T_ℓ of degree polylogarithmic in m by

$$T_\ell(x_1, \ldots, x_n, \overline{r}^{(1,1)}, \ldots, \overline{r}^{(m,1)}, \overline{r}^{(2,1)}, \ldots, \overline{r}^{(m,\ell)}) = \prod_{a=1}^{\ell} \prod_{j=0}^{m} (1 - S_j(x_1, \ldots, x_n, \overline{r}^{(1,a)}, \ldots, \overline{r}^{(m,a)})).$$

In order to clarify the meaning of these functions, let us explicitly describe how they parallel the proof of the Valiant-Vazirani Theorem. Recall that, in that proof, we took a formula ϕ on m variables and repeatedly intersected its set of satisfying assignments with random hyperplanes to obtain new formulae $\phi_1, \phi_2, \ldots, \phi_m$, where ϕ_i is the formula obtained after i restrictions. An "isolation lemma" told us that if ϕ had at least one satisfying assignment then, with probability at least $1/8$, one of the ϕ_i would have at *exactly* one satisfying assignment.

In the above functions, the $\overline{i}$ vectors correspond to the satisfying assignments in the proof of Valiant-Vazirani and the $\overline{r}$ vectors correspond to the random hyperplanes. The $\mathrm{VV}(\overline{i}, \overline{r})$ function tells us whether $\overline{i}$ lies in the hyperplane defined by $\overline{r}$. Think of $x_{\overline{i}}$ as being 1 if $\overline{i}$ is a satisfying assignment and 0 otherwise. Then $\sum_{\overline{i}} x_{\overline{i}}$ corresponds to the number of satisfying assignments. Multiplying $x_{\overline{i}}$ by $\mathrm{VV}(\overline{i}, \overline{r}^{(k)})$ kills $x_{\overline{i}}$ if $\overline{i}$ doesn't lie in the hyperplane defined by $\overline{r}$. Thus S_j corresponds to the number of satisfying assignments after j random restrictions have been imposed. Finally, the inner product in T_ℓ is 1 when the restrictions $\overline{r}^{(1,a)}, \ldots, \overline{r}^{(m,a)}$ only produce formulae with 0 satisfying assignments and is 0 when at least one of the formulae has 1 satisfying assignment. Thus, by the "isolation lemma" we used to prove Valiant-Vazirani, we have the following:

Proposition 1 *If* $\sum_{\overline{i}} x_{\overline{i}} = 0$ *then* $T_\ell(\cdots) = 1$.
If $\sum_{\overline{i}} x_{\overline{i}} > 0$ *then* $\Pr_{\{r^{(k,a)}\}}[T_\ell(\overline{x}, \{\overline{r}^{(k,a)}\}) = 0] > 1 - \left(\frac{7}{8}\right)^\ell$.

Thus T_ℓ probabilistically "computes" the NOR of the $x_{\overline{i}}$'s. By replacing the $x_{\overline{i}}$'s with $1 - y_{\overline{i}}$ or by replacing the T_ℓ with $1 - T_\ell$, we get probabilistic polynomials which compute OR and AND. (NOT is trivial.) We can now prove what we wanted about AC_0.

Proposition 2 *For any* AC_0 *predicate* A *of depth* d*, there exists a family of probabilistic polynomials* P *of degree* $(\log n)^{O(d)}$ *with* $(\log n)^{O(1)}$ *random inputs such that*

$$\Pr_r[P(x,r) = A(x)] > \frac{2}{3}.$$

Proof Idea Starting with the leaves, replace each gate in the AC_0 circuit with the corresponding AND/OR/NOT polynomial in its inputs. Use the same random inputs r for each polynomial, and if a gate is used more than once, simply use the same polynomial. In constructing the polynomials, set $\ell = m^2$, so that the probability of any fixed one giving the wrong answer on random input r is at most $(7/8)^{m^2} = 1/n^{\Omega(\log n)}$. Since there are only $n^{O(1)}$ gates in all, the probability that any one fails on random input r is $n^{O(1)}/n^{\Omega(\log n)} < 1/3$ for sufficiently large n. ■

We've shown how to compute AC_0 predicates with probabilistic polynomials, but it'd be better if we could remove the random inputs from our polynomials. A first attempt at this would be to consider

$$Q(x) = \sum_r P(x,r),$$

where $P(x,r)$ is the probabilistic polynomial which computes an AC_0 predicate A. We know that if $A(x) = 0$, then $P(x,r)$ is zero for most r and that if $A(x) = 1$, then $P(x,r)$ is 1 for most r, so it seems like we should be able to recover $A(x)$ from $Q(x)$. However, for the r on which $P(x,r)$ is incorrect, $P(x,r)$ can take on all kinds of crazy values other than 0 and 1 and completely throw off the summation.

Toda offered a clever solution to this problem. His idea was to work modulo a large power of 2 with not the original values $P(x,r)$, but values $f(P(x,r))$ for a suitably chosen function f. Define $h^{(1)}(z) = 4z^3 + 3z^4$, and inductively define $h^{(c)}(z) = h(h^{(c-1)}(z))$. Then we can relate the value of $h(z)$ to the values of z modulo 2 as follows:

Proposition 3 *If $z \equiv 0 \pmod 2$, then $h^{(c)}(z) \equiv 0 \pmod{2^{2^c}}$.*
If $z \equiv -1 \pmod 2$, then $h^{(c)}(z) \equiv -1 \pmod{2^{2^c}}$.

Proof Straightforward induction on c. ■

Intuitively, if we let $f = h^{(c)}$ for c sufficiently large, then $\sum_r f(P(x,r))$ taken modulo 2^{2^c} should enable us to estimate the probability that $P(x,r) = 1$, because even when $P(x,r)$ takes on nasty integer values other than 0 or 1, $f(P(x,r))$ will still be only 0 or -1 modulo 2^{2^c}, so these bad values can't throw off the sum modulo 2^{2^c} too much. Formally, we have:

Proposition 4 *Let P be a probabilistic polynomial which computes $\mathsf{AC_0}$ predicate A as constructed earlier. Set $c = \lceil \log R \rceil + 1$, where R is the number of random inputs used by P. Then*

$$\sum_r h^{(c)}(P(x,r)) \pmod{2^{2R}} \in [2^{2R} - 2^R, 2^{2R} - \frac{2}{3}2^R]$$

if and only if $A(x) = 1$.

Proof Since $2^c \geq 2R$, 2^{2R} divides 2^{2^c}, so any equation that holds modulo 2^{2^c} also holds modulo 2^{2R}. If $A(x) = 1$, we know that, for at least $(2/3)2^R$ values of r, $P(x,r)$ equals 1 and thus $f(P(x,r))$ equals -1 modulo 2^{2R}. For the other values of r, $f(P(x,r))$ equals 0 or -1 modulo 2^{2R}. Since there are only 2^R choices for r, $\sum_r f(P(x,r))$ must lie in the range $[-2^R, -(2/3)2^R]$ modulo 2^{2R}, which is the same as the range listed in the proposition. Similarly, if $A(x) = 0$, then for at least $(2/3)2^R$ values of r, $f(P(x,r))$ equals 0 modulo 2^{2R}. Thus $f(P(x,r))$ equals -1 for at most $(1/3)2^R$ values of R and $\sum_r f(P(x,r)) \pmod{2^{2R}}$ must lie in the range $[-(1/3)2^R, 0]$, which does not intersect the range listed in the proposition. ■

We now have all the machinery we need to prove Toda's Theorem:

Theorem 5 $\mathsf{PH} \subset \mathsf{P}^{\#\mathsf{P}}$.

Proof Let $L = \mathrm{QBF}_k$, which is complete for Σ_k^p, and let M be the alternating Turing machine with at most k alternations which accepts L. WLOG, we may assume that, on inputs ϕ of length μ, every computation of M has μ existential quantifiers, followed by μ universal quantifiers, followed by μ existential quantifiers, and so on, for a total of k alternations, where these quantifiers correspond to choosing a vector of values $\bar{\imath} \in \{0,1\}^{k\mu}$ for the variables of ϕ (and some dummy variables for this normalization), and that the order in which variables are quantified is independent of the computation path.

The computation of M on input ϕ can be regarded as an $\mathsf{AC_0}$ circuit as follows: replace the universal quantifiers with AND gates, replace the existential quantifiers with OR gates, and consider $\phi(\overline{i}) \in \{0,1\}$ as input at the leaf corresponding to the nondeterministic choices $\overline{i}$. This circuit has depth k and $2^{k\mu}$ inputs. (Group a sequence μ of AND gates into a single AND gate of fan-in 2^{μ}.) The construction we just completed shows that there is a polynomial Q on the $2^{k\mu}$ variables $x_{\overline{i}}$ of degree polynomial in $m = k\mu$ whose value modulo a suitable power of 2 reveals the value of this $\mathsf{AC_0}$ predicate. If we substitute $x_{\overline{i}} = \phi(\overline{i})$, then we can recover the output of M on input ϕ from the value of Q. Since it is a polynomial-time computable function, $f_0(\phi, \overline{i}) = \phi(\overline{i}) \in \mathsf{Gap-P}$. Using the closure properties of $\mathsf{Gap-P}$ from last lecture to analyze our construction of Q, we can see that $g(\phi) = Q(\{\phi(\overline{i})\}) \in \mathsf{Gap-P}$. Thus, a single call to a $\mathsf{Gap-P}$ oracle for g enables us decide L.

We now show a bit more explicitly, why the function g described above is in $\mathsf{Gap-P}$. We already mentioned that $f_0(\phi, \overline{i}) \in \mathsf{Gap-P}$. $\mathrm{VV}(\overline{i}, \overline{r})$ is also in $\mathsf{Gap-P}$, because it too is polynomial-time computable. (Just have a machine with one computation which accepts if the function is 1 and rejects if the function is 0.) For the same reason,

$$f_1(k,j) = \begin{cases} 1, & \text{if } k \leq j \\ 0, & \text{otherwise} \end{cases}$$

is also in $\mathsf{Gap-P}$.

Repeatedly applying the closure properties from last lecture, we get

$$\begin{aligned}
f_2(\overline{i}, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}, j) &= \prod_{k=1}^{j} \mathrm{VV}(\overline{i}, \overline{r}^{(k)}) \\
&= \prod_{k=1}^{m} f_2(k,j) VV(\overline{i}, \overline{r}^{(k)}) \in \mathsf{Gap-P} \\
f_3(\phi, \overline{i}, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}, j) &= f_0(\phi, \overline{i}) f_2(\overline{i}, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}, j) \in \mathsf{Gap-P} \\
f_4(\phi, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}, j) &= S_j(\{\phi(\overline{i})\}, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}) \\
&= \sum_{\overline{i}} f_3(\phi, \overline{i}, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}, j) \in \mathsf{Gap-P} \\
f_5(\phi, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}) &= \prod_{j=0}^{m} (1 - f_4(\phi, \overline{r}^{(1)}, \ldots, \overline{r}^{(m)}, j)) \in \mathsf{Gap-P} \\
f_6(\phi, \overline{r}^{(1,1)}, \ldots, \overline{r}^{(m^2,m)}) &= T_{m^2}(\{\phi(\overline{i})\}, \overline{r}^{(1,1)}, \ldots, \overline{r}^{(m^2,m)}) \\
&= \prod_{a=1}^{m^2} f_5(\phi, \overline{r}^{(a,1)}, \ldots, \overline{r}^{(a,m)}) \in \mathsf{Gap-P}
\end{aligned}$$

This shows that the a single AND/OR gate in our circuit can be replaced by a $\mathsf{Gap-P}$ function of ϕ and the random variables $\overline{r}^{(a,k)}$. Notice that the above argument didn't use any special properties of the function $f_0(\phi, \overline{i}) = \phi(\overline{i})$ other than it being in $\mathsf{Gap-P}$. Thus, the same argument tells us that the polynomials for gates higher up in our $\mathsf{AC_0}$ circuit are in $\mathsf{Gap-P}$, since we can use $\mathsf{Gap-P}$ functions for their inputs. This can be written out formally as above, but the notation gets messy. So, we have shown that the probabilistic polynomial $P(\{\phi(\overline{i})\}, \{\overline{r}^{(a,k)}\})$ is a $\mathsf{Gap-P}$ function $p(\phi, \{\overline{r}^{(a,k)}_{(a,k)}\})$ of ϕ and the random inputs. We would now like to conclude that

$$g(\phi) = Q(\{\phi(\bar{i})\}) = \sum_{\{\overline{r}^{(a,k)}\}} h^{(R)}(P(\{\phi(\bar{i})\}, \{\overline{r}^{(a,k)}\})$$

is a $\mathsf{Gap-P}$ function of ϕ, where $R = m^{O(1)}$. Unfortunately, none of our closure properties say anything about iterated composition, which is how we defined $h^{(R)}$. Fortunately, since $h(z) = 4z^3 + 3z^4$ has such a simple form, the monomials in $h^{(R)}$ can be described explicitly, *i.e.* the function $C(d, 1^R)$ which gives the coefficient of z^d in $h^{(R)}$ can be computed in time polynomial in R. Thus, we have

$$g(\phi) = \sum_{\{\overline{r}^{(a,k)}\}} \sum_{d} C(d, 1^R) p(\phi, \{\overline{r}^{(a,k)}\}) \in \mathsf{Gap-P}.$$

and we are finished. ■