18.405J/6.841J Advanced Complexity Theory	March 9, 2000

Lecture 11

Lecturer: Dan Spielman — *Scribe: Michael Brewer*

1 Old News

First, a clarification of what happened last time. We showed that given an OR gate with arbitrary parity, $\mathrm{OR}(x_1, \ldots, x_n)$, we could replace it with a randomized polynomial $P(r, x_1, \ldots, x_n)$, having degree $O((\log n)^k)$ and with $|r| = O((\log n)^k)$ for some k, such that

$$\forall x \left(\Pr_r(\mathrm{OR}(x_1, \ldots, x_n) = P(r, x_1, \ldots, x_n)) > 1 - \left(\frac{7}{8}\right)^{n^{k-3}} \right).$$

Given an AC_0 circuit C of constant depth c, then, we can replace each OR gate with P, and each AND gate with $1 - P(1 - \overline{x})$ to get a polynomial P_C with degree $O((\log n)^{k+c})$. Given that

$$\#\{\text{gates}\} \times \left(\frac{7}{8}\right)^{n^{k-3}} < \frac{1}{4},$$

we have that

$$\Pr_r(C(x_1, \ldots, x_n) = P_C(r, x_1, \ldots, x_n)) > \frac{3}{4}.$$

Now, assuming that $C = \mathrm{parity}(x_1, \ldots, x_n)$, and transforming our polynomials from $\{0, 1\}$ to $\{1, -1\}$, as discussed last time, we get that $\mathrm{parity}(x_1, \ldots, x_n) = \prod_{i=1}^n x_i$.

Fixing our random bits r, we get that $P_C(r, x_1, \ldots, x_n) = \mathrm{parity}(x_1, \ldots, x_n)$ on 3/4 of the inputs. Let $S = \{\text{inputs on which } P_C \text{ and parity agree}\}$. Then, for the function space $S \to \mathbb{R}$, maps from S to the real numbers, we have $\dim(S \to \mathbb{R}) \geq \frac{3}{4} \cdot 2^n$. On the other hand, using the parity function, we only need monomials of degree at most $\frac{n}{2}$ to express all such functions, using the fact that $(\pm 1)^2 = 1$ and the fact that any monomial is equivalent to one of smaller degree modulo the parity function. This is a contradiction, since $\frac{n}{2}$ is much smaller than $\frac{3}{4} \cdot 2^n$.

In fact, by careful analysis of the proof above, we get that a circuit computing parity needs $2^{n^{\Omega(1/d)}}$ gates.

Now on to the new material:

2 A Relativization for $\mathrm{PH} \neq \mathrm{PSPACE}$

Theorem 1 *There exists an oracle A such that $\mathrm{PH}^A \subsetneq \mathrm{PSPACE}^A$.*

Proof The idea is to find something that a PSPACE machine can do easily, but which is very hard for a PH machine. We will essentially use a very long parity computation.

For an oracle A, define

$$L_A = \{0^n \mid A \text{ has an odd number of strings of length } n\}.$$

- $L_A \in \mathrm{PSPACE}^A$.

  To see whether $0^n \in L_A$, we check whether each word of length n is in A, keeping a running total of how many are, and compute parity of the total.

- $L_A \notin \mathrm{PH}^A$.

  It suffices to show that $L_A \notin \Sigma_k^P$ for any k, and in particular to show that no Σ_k^P machine which runs in time n^c for constant c decides L_A. Suppose, to the contrary, that such a machine M did decide L_A.

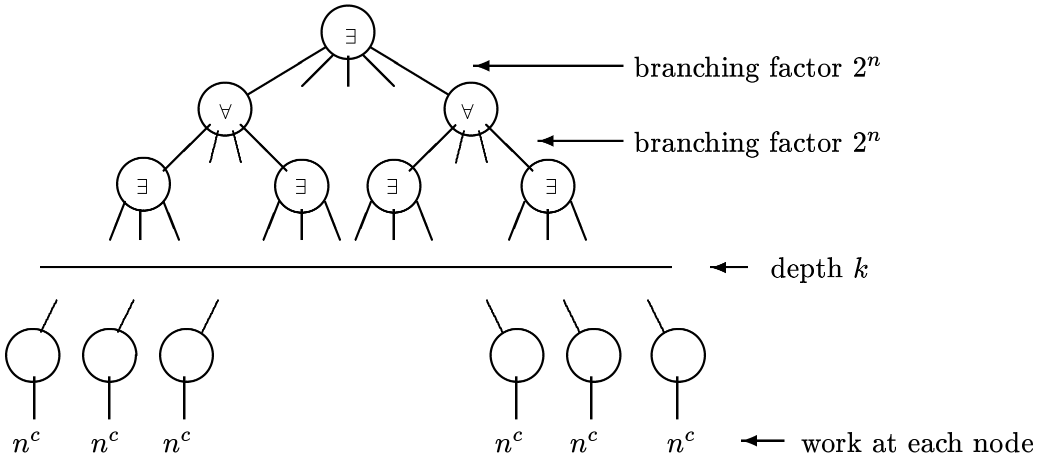

Figure 1: Circuit representing M.

We can represent M with an alternating circuit C, a loose diagram of which is given in Figure 1, which starts with an OR gate, has branching factor 2^{n^c} at each node, and does polynomial (n^c) work at each leaf.

Note the size of this circuit isn't quite polynomial in $2^n = N$. Its size is bounded by $2^{k(\log N)^c} < 2^{N^{1/k}}$ for sufficiently large N, however, so we will be able to apply our circuit lower bound on parity.

At each branch, C can access the oracle polynomially many times. To apply our lower bound, we'll need to correct this:

Claim 2 *For any oracle machine $M^? \in \Sigma_k^P$, there exists $M'^? \in \Sigma_{k+2}^P$, equivalent, which reads the oracle only once on each branch.*

Sketch of Proof We want M' to simulate M. We can assume M passes through all $\exists$ and $\forall$ states before asking questions of the oracle. M' will follow M through this stage. At this point, M' will:

1. Use $\exists$ state to guess which questions M will ask of the oracle, and what the answers the oracle gives will be.
2. Simulate M using these guesses, rejecting if M asks a question that M' didn't guess the answer to above.
3. Use $\forall$ state to choose one question from (a), ask the oracle that question, and accept if and only if its guess agreed with the oracle.

■

Now, for any $M \in \Sigma_k^P$, for each input length n, on input 0^n, we get some depth $k+2$ AND/OR circuit of size $2^{poly(n)}$, with inputs of whether or not a single word of length 2^n is in the oracle, and NOTs just above the inputs. We can feed the circuit the words in the oracle of length n to decide whether or not the circuit accepts.

From our circuit lower bound, we know that for some sufficiently large n, there exists A such that the circuit gets parity wrong. That is, either $M^A(0^n)$ accepts when $0^n \notin L_A$, or $M^A(0^n)$ rejects when $0^n \in L_A$.

To diagonalize, build the following A. Enumerate machines M_i, and find for each i sufficiently large n_i so that M_j, $j < i$, asks no questions about words of length at least n_i on input 0^{n_j}, and so that there exists some language A_i for which M_i fails to compute parity. Make A agree with A_i on words of length n_i.

■

3 $\#P$ and Gap-P

We previously defined $\#P$. Note that this class is inflexible: the sum or product of two functions is in $\#P$, but the difference is not. We define something a bit easier to work with. Write

$$\begin{aligned} \mathrm{acc}_M(x) &= \#\{\text{accepting paths of } M \text{ on } x\} \\ \mathrm{rej}_M(x) &= \#\{\text{rejecting paths of } M \text{ on } x\} \end{aligned}$$

Definition 3 *A function $f \in$ Gap-P if there exists a polynomial-time non-deterministic TM M such that*

$$\forall x (f(x) = \mathrm{acc}_M(x) - \mathrm{rej}_M(x)).$$

Theorem 4 $P^{\#P} = P^{\text{Gap-P}}$.

Proof

- $P^{\text{Gap-P}} \subset P^{\#P}$.

  We must compute in $P^{\#P}$, for a polynomial time NTM M, $\#\{\text{accepting paths of } M\} - \#\{\text{rejecting paths of } M\}$. Using the $\#P$ oracle will give us the number of accepting paths. To get the number of rejecting paths, we swap the accept and reject states for M, creating a new NTM M', and call the $\#P$ oracle on M'.

- $P^{\#P} \subset P^{\text{Gap-P}}$.

  Given a polynomial time NTM M, replace our reject states as follows:

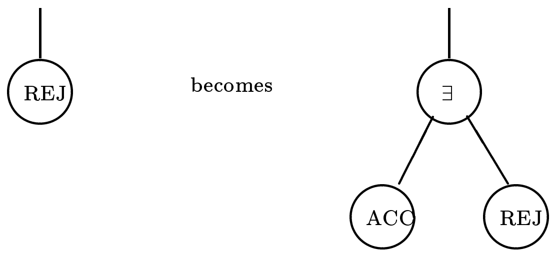

Figure 2: Change in reject states of M.

Call the resulting machine M'. Then

$$\mathrm{acc}_{M'}(x) - \mathrm{rej}_{M'}(x) = \mathrm{acc}_M(x).$$

■

Some great things about Gap-P:

Lemma 5 *Let $f, g \in$ Gap-P and $h \in$ FP, the class of polynomial-time-computable functions. Then*

1. $f + g \in$ Gap-P
2. $-f \in$ Gap-P
3. $f \cdot g \in$ Gap-P
4. $h \in$ Gap-P

Proof

Suppose that M_f and M_g are defining NTMs of f and g, respectively. Then

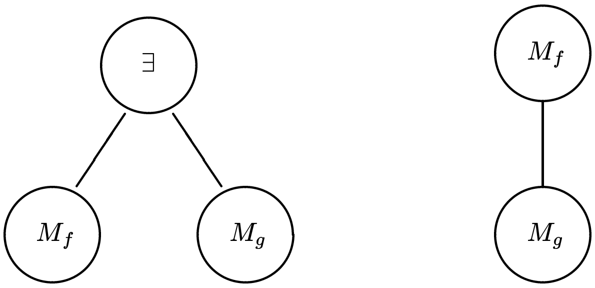

Figure 3: Defining NTMs for $f + g$ and $f \cdot g$, respectively.

1. $f + g$ has as defining NTM the machine on the left hand side of Figure 3. Clearly the number of accepting and rejecting states add.
2. $-f$ has as defining NTM the machine generated from M_f by swapping accept and reject states.
3. The defining NTM for $f \cdot g$ has computation structure as on the right hand side of Figure 3. We modify the machine so that it accepts if both M_f and M_g accept or both reject. Call the result M'. We have on input x,

$$\begin{aligned}
\mathrm{acc}_{M'}(x) - \mathrm{rej}_{M'}(x) &= \mathrm{acc}_{M_f}(x)\,\mathrm{acc}_{M_g}(x) + \mathrm{rej}_{M_f}(x)\,\mathrm{rej}_{M_g}(x) \\
&\qquad - \mathrm{acc}_{M_f}(x)\,\mathrm{rej}_{M_g}(x) - \mathrm{rej}_{M_f}(x)\,\mathrm{acc}_{M_g}(x) \\
&= (\mathrm{acc}_{M_f}(x) - \mathrm{rej}_{M_f}) \cdot (\mathrm{acc}_{M_g}(x) - \mathrm{rej}_{M_g}) \\
&= f(x) \cdot g(x)
\end{aligned}$$

   as desired.

4. Define M to, on input x, deterministically compute $h(x)$. If $h(x) = 0$, then M will branch once, accepting on one branch and rejecting on the other. If $h(x) > 0$ (< 0, respectively), then M will branch according to the following recursive rule applied to $|h(x)|$, accepting (resp.,

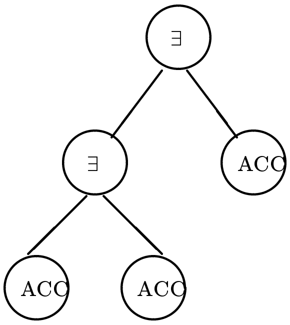

Figure 4: Branching diagram for M when $h(x) = 3$.

rejecting) on each leaf. If M needs to create 1 node, it will not branch, otherwise M will branch, creating $\lceil |h(x)| \rceil$ leaves on its left branch, and $\lfloor |h(x)| \rfloor$ leaves on the right branch.

In all M will create $|h(x)|$ leaves, so that $\mathrm{acc}_M(x) - \mathrm{rej}_M(x) = h(x)$. An example for M's branching process when $h(x) = 3$ is in Figure 4.

■