18.405J/6.841J Advanced Complexity Theory February 19, 1998

Lecture 4

Lecturer: Daniel A. Spielman *Scribe: Leonid Reyzin*

Polynomial Hierarchy

Let $\Sigma_k T(n)$ be a class of languages L accepted by an Alternating Turing Machine (ATM) that begins in an existential state, alternates between existential and universal states at most $k-1$ times, and halts (on all branches) within time $O(T(n))$ (we will call such an ATM a k-ATM). Define

$$\Sigma_k P = \bigcup_{i=1}^{\infty} \Sigma_k n^i.$$

The class of complements of languages within $\Sigma_k P$ is called $\Pi_k P$:

$$\Pi_k P = \text{co}-\Sigma_k P.$$

Alternatively, $\Pi_k P$ can be defined the same way as $\Sigma_k P$ except that the ATM begins in a universal state rather than an existential state (we will call such an ATM a co-k-ATM). Note that, by definition, $\Sigma_k P \subseteq \Pi_{k+1} P$ and $\Pi_k P \subseteq \Sigma_{k+1} P$. Also note that $\Sigma_1 P = NP$ and $\Pi_1 P = \text{co}-NP$.

Definition 1 *The polynomial hiearchy* $PH = \bigcup_k \Sigma_k P = \bigcup_k \Pi_k P$.

Many believe that $(\forall k)\Sigma_k P \neq \Pi_k P$ and $(\forall k)\Sigma_k P \neq \Sigma_{k+1} P$. In fact, these two are equivalent.

Theorem 2 $\Sigma_k P = \Pi_k P$*, for some* k*, if and only if* $(\forall j > k)\Sigma_k P = \Sigma_j P$ *(if so, we say that the polynomial hierarchy "collapses to level* k*.")*

Proof Assume $\Sigma_k P = \Pi_k P$. We will prove that $\Sigma_k P = \Sigma_{k+1} P$. The assumption implies that for every k-ATM accepting a language $L \in \Sigma_k P$, there exists a co-k-ATM that also accepts L. Consider a language $L \in \Sigma_{k+1} P$. There exists a $(k+1)$-ATM M that accepts L. M begins in a $\exists$ state, and eventually goes to the first $\forall$ state. Consider the machine right before the first $\forall$ state—from this point on, it is just a co-k-ATM, with the current configuration of the tape as input. The question of whether M accepts is equivalent to the question of whether the co-k-ATM accepts what is written on the tape. By the assumption, there exists a k-ATM accepting the same lagnuage. So we can replace the co-k-ATM by a k-ATM. We can put back together the k-ATM with with the states of M before the first $\forall$ state. The result will be a k-ATM. Thus, $\Sigma_k P = \Sigma_{k+1} P$. Similarly we can prove that $\Pi_k P = \Pi_{k+1} P$, thus $\Sigma_{k+1} P = \Sigma_k P = \Pi_k P = \Pi_{k+1} P$, and we can go up by induction.

The converse follows immediately from the fact that $\Sigma_k P \subseteq \Pi_{k+1} P$ and $\Pi_k P \subseteq \Sigma_{k+1} P$. ■

Corollary 3 *If* $\Sigma_k P \neq \Pi_k P$ *for some* $k \geq 1$*, then* $P \neq NP$

Proof By contradiction. Assume $P = NP$. Then $P = \text{co}-NP$, so $NP = \text{co}-NP$, so $\Sigma_k P = \Pi_k P$. ■

We will now provide an equivalent definition of PH, without proving equivalence in this lecture. The definition uses Oracle Turing Machines (OTMs): a machine with an oracle for a language L gets to ask questions about membership in L and gets answers in a single step. An OTM M is denoted by $M^?$ in general and M^L when supplied with an oracle for a language L.

Define

$$P^{SAT} = \{L | \exists \text{ PTIME OTM } M^{?} \text{s.t. } L \text{ is accepted by } M^{SAT}\}.$$

Note that $NP \subseteq P^{SAT}$ and co$-NP \subseteq P^{SAT}$.

Since SAT is NP-complete, we can use the notation P^{NP} instead of P^{SAT}. That is, if L is accepted by a PTIME machine with an oracle for an NP language, then L is also accepted by a PTIME machine with an oracle for SAT.

So, an alternative definition of PH is as follows (we will use superscripts instead of subscripts since we have not proved their equivalence). Let $\Sigma^1 P = NP$, and $\Sigma^i P = NP^{\Sigma^{i-1}P}$ for $i > 1$. Define $\Pi^k P = \text{co}-\Sigma^k P$. Then for $k > 1$, $\Pi^k P = \text{co}-NP^{\Sigma^{k-1}P} = \text{co}-NP^{\Pi^{k-1}P}$ (the last statement is true because having an oracle for a language is the same as having an oracle for its complement). As before, we define PH to be the union of $\Sigma_k P$ for all k.

One can prove that $\Sigma^k P$ and $\Pi^k P$ have complete problems, for each k (not proven in this lecture).

Proposition 4 *If there exists a language L that is complete for PH, then there exists k such that the hierarchy collapses to level k.*

Proof Assume L is complete for PH. Then $L \in PH$, so $L \in \Sigma^k P$ for some k. Let $A \in \Sigma^j P$, for $j > k$. Since L is complete, there exists a polynomial-time computable function f s.t. $w \in A$ iff $f(w) \in L$. So $\Sigma^k P = \Sigma^j P$. ■

Capturing Non-Uniformity in Languages

So far, the Turing machines we've studied do not capture the non-uniformity that can occur in languages as the string size grows. Here is a new class of languages that captures a limited form of non-uniformity:

Definition 5 *$P/poly$ is the class of languages C such that $L \in C$ if there exists a polynomial-time Turing Machine M and a sequence of strings $\{s_i\}_{i \in \mathbf{N}}$ such that $|s_i| = i^{O(1)}$, and*

$$x \in L \quad \Leftrightarrow \quad M(x, s_{|x|}) \textit{accepts.}$$

$P/poly$ is the set of languages L accepted by "a polynomial-time Turing machine with polynomial advise". That is, on inputs of size n, the machine recieves an arbitrarily specified string as an additional input. This string depends only on the length of the machine's input, and does not depend on the contents of the input. The machine can use this additional string to decide whether it will accept the input.

The following two theorems will be proven in the next lecture.

Theorem 6 *$P/poly$ is a set of languages L such that there exists a sequence of circuits $c_0, c_1, \ldots$ such that the number of gates in c_i is bounded by a polynomial in i and $w \in L$ if and only if $c_{|w|}(w)$ accepts.*

Theorem 7 *If $NP \subseteq P/poly$, then $\Sigma_2 P = \Pi_2 P$.*