CS221: Computational Complexity Prof. Salil Vadhan

Lecture 13: The Polynomial Hierarchy

10/18 *Scribe: Saurabh Sanghvi*

Contents

1 Announcements

- Pick up graded PS1s and the solution. Please note the length of the solutions—this should be a model of how to be more concise with your answers.
- CS Colloquium talk by Yuri Gurevich on abstract state machines on Monday, 4PM.
- Section in MD 323 on Monday, 6PM.

2 Recap

To review, last lecture we defined the following:

- $\mathbf{\Sigma_k P} \triangleq$ Alternating polynomial time, $\leq k-1$ alternations, starting with $\exists$
- $\mathbf{\Pi_k P} \triangleq$ Alternating polynomial time, $\leq k-1$ alternations, starting with $\forall$
- $\mathbf{PH} \triangleq \bigcup_k \mathbf{\Pi_k P}$

We know, of course that $\mathbf{PH} \subseteq \mathbf{PSPACE}$. We can think of the **PH** as a more slight or benign generalization of **NP**, or thinking about the **AP**/**PSPACE** model, as games where you fix the number of rounds.

Why, one might ask, do we care about the complexity of games? A large focus of study is in *protocols*, involving the interaction of algorithms. Thinking about games is a helpful way to analyze these interactions.

3 Complete Problems

Recall the definition of QBF from last lecture. Define, similarly, the following for each k:

$$\mathrm{QBF}_k = \{\varphi(X_1,\ldots,X_k) \ : \ \exists X_1 \forall X_2 \ldots QX_k \ \varphi(X_1,\ldots,X_k), \varphi \text{ an arbitrary Boolean formula } \}$$

Here each X_i represents a group of Boolean variables $(x_{i,1}, x_{i,2}, \ldots, x_{i,n_i})$. Just as we found that QBF was **PSPACE**-complete, we can show the following for each k:

Theorem 1 QBF_k *is* $\mathbf{\Sigma_k P}$*-complete.*

Proof: That $\mathrm{QBF}_k \in \mathbf{\Sigma_k P}$ follows from logic similar to what we saw before for QBF, and so we will just show QBF_k is $\mathbf{\Sigma_k P}$-hard.

Given $L \in \mathbf{\Sigma_k P}$, by definition we have a polynomial time relation R, polynomial p with:

$$x \in L \Leftrightarrow \exists y_1 \forall y_2 \dots Q y_k \; R(x, y_1, \dots y_k), \text{ where each } y_i \in \{0,1\}^{p(|x|)}$$

Last time at this point we used Cook–Levin to replace R with a formula of the form 3-CNF formula. We'll do basically the same thing here, but this time we'll need to be sure that we don't increase the number of alternations in the process.

Then by the proof of the Cook–Levin theorem we can replace $R(x, y_1, \dots, y_k)$ with $\exists z \; \varphi_R(x, y_1, \dots, y_k, z)$. It is easy to see that this replacement will not increase the number of alternations.

Otherwise, suppose $Q = \forall$. Then we can apply Cook-Levin to $\neg R$ to get $\exists z \; \varphi_{\neg R}(x, y_1, \dots y_k, z)$. Negating, and then pushing the negation through, we get $\forall z \; (\neg\varphi_{\neg R})(x, y_1, \dots y_k, z)$ (where $(\neg\varphi_{\neg R})$ will now be in 3-DNF — but this is fine). We can then replace R with this and not increase the number of alternations. ■

4 Oracle Characterization

Another way of defining **PH** is through *oracles*, which you may remember from our definition of Cook reductions. In fact, this is the usual way it is defined. If we let A be some function or language, then we can define the following:

Definition 2 $\mathbf{P}^A$ *is the class of language decided in polynomial time with an "oracle" for A. Similarly,* $\mathbf{NP}^A$ *is the class decided in nondeterministic polynomial time, with access to such an oracle.*

In order to make sense of this, we need to define what it means to compute with access to an oracle:

Definition 3 *Formally, an* oracle TM *M is a TM with an extra "oracle" tape and a special oracle-query state. When M goes into the query state, its oracle is applied to the contents of the oracle tape. Running an oracle TM M with oracle A is denoted M^A.*

Important: The operation of adding oracles is an operation on *machines*, not an operation on language classes. Phrased differently, $\mathbf{C}^A$ is not defined for an arbitrary class $\mathbf{C}$; it depends on the machine used to define $\mathbf{C}$. For instance, complexity classes not defined with respect to properties of the machines deciding them will not make sense with an oracle attached. Additionally, $\mathbf{L}^A$ is not well-defined, because there are disagreements on how to charge for the space used on the query tape, and different formulations are inequivalent.

Even so, most classes we have seen can be thought of in the context of an oracle, and in fact, most of the results we have shown apply within the context of oracles, as well.

We can also talk about machines with access to an oracle in any given class via the following:

Definition 4 *For class* $\mathbf{C}$, $\mathbf{P}^{\mathbf{C}} \triangleq \bigcup_{L \in \mathbf{C}} \mathbf{P}^{L}$ *(which is just* $\mathbf{P}^{L_0}$ *if* L_0 *is complete for* $\mathbf{C}$*).*

Theorem 5 $\mathbf{\Sigma_k P} = \mathbf{NP}^{\mathbf{\Sigma_{k-1}P}}$, $\mathbf{\Pi_k P} = \textbf{co-}\mathbf{NP}^{\mathbf{\Sigma_{k-1}P}}$.

This theorem shows that we can think of oracles for levels of the polynomial hierarchy as defining "worlds upon worlds," where you get a new notion of **P** and **NP** each time you increase the k in the oracle.

To prove it, we require the following lemma:

Lemma 6 $\mathbf{P}^{\mathbf{\Sigma_{k-1}P}} \subseteq \mathbf{\Sigma_k P} \cap \mathbf{\Pi_k P}$

Proof: Since $\mathbf{P}^{\mathrm{QBF}_{k-1}}$ is closed under complement, we don't need to worry about $\mathbf{\Pi_k P}$, and it suffices to show that $\mathbf{P}^{\mathrm{QBF}_{k-1}} \subseteq \mathbf{\Sigma_k P}$.

Suppose we are given L such that a polynomial time oracle algorithm M decides L. We want a $\mathbf{\Sigma_k P}$ algorithm:

1. Simulate $M(x)$.
2. When M makes the i^{th} oracle query φ_i, nondeterministically guess the answer $\sigma_i \in \{0, 1\}$.
3. At the end, we should have queries $\varphi_1, \varphi_2, \ldots, \varphi_m$ and answers $\sigma_1, \sigma_2, \ldots, \sigma_m$.
4. Now, check whether the answers were correct (use $\mathbf{\Sigma_k P}$ algorithm).

This process of checking should be seen in greater detail: For example, we might have something like:

φ_1	φ_2	φ_3
$\sigma_1 = 1$	$\sigma_2 = 0$	$\sigma_3 = 0$

This basically requires checking:
$(\exists x_1 \forall x_2 \ldots Q x_{k-1} \varphi_1(-)) \wedge \neg(\exists y_1 \forall y_2 \ldots Q y_{k-1} \varphi_2(-)) \wedge \neg(\exists z_1 \ldots Q z_{k-1} \varphi_3(-))$.
To verify this, we can just push the negations through and limit the number of alternations by interleaving quantifiers (this is possible because the variables are disjoint). This may take k alternations since a negation could produce a $\forall$ sitting in the front of $k-1$ alternations. Note also that we needed to use a $\exists$ to get the answers to the oracle in the first place, but this can just be combined with the $\exists$ in front. ■

Proof of theorem:
To show $\mathbf{\Sigma_k P} \subseteq \mathbf{NP}^{\mathbf{\Sigma_{k-1}P}}$, it suffices to show that $\mathrm{QBF}_k \in \mathbf{NP}^{\mathrm{QBF}_{k-1}\mathbf{P}}$.
But this is clear:

$$\overbrace{\exists x_1 \underbrace{\forall x_2 \ldots Q x_k \; \varphi(x_1, \ldots, x_k)}_{\text{instance of } \mathrm{QBF}_{k-1}}}^{\text{instance of } \mathrm{QBF}_k}$$

Conversely, suppose $L \in \mathbf{NP}^{\mathbf{\Sigma_{k-1}P}}$. By the certificate characterization of $\mathbf{NP}^A$, $\exists R \in \mathbf{P}^{\mathbf{\Sigma_{k-1}P}}$ such that $x \in L \Leftrightarrow \exists y, |y| \leq p(|x|) R(x, y)$. By the lemma, $R \in \mathbf{\Sigma_k P}$. We can then combine in the $\exists y$, and so $L \in \mathbf{\Sigma_k P}$. ■