18.405J/6.841J Advanced Complexity Theory	February 24, 1998
Lecture 5	
Lecturer: Dan Spielman	*Scribe: David Ratajczak*

1 Introduction

In this lecture we will prove a relationship between the polynomial time hierarchy and another important class we will study: $P/poly$. First, we will define the class $P/poly$. We then introduce the notion of circuit complexity and show its relation to $P/poly$. Finally, we prove the main theorem stating that if NP can be decided by a sequence of polynomial size circuits then the polynomial hierarchy collapses to a very low level.

2 $P/poly$ and Circuit Complexity

We begin by defining the $\mathcal{C}/\mathcal{S}$ notation.[1]

Definition 1 *Let $\mathcal{C}$ be a class of languages and $\mathcal{S}$ be a family of sequences of strings. Then define* $\mathcal{C}/\mathcal{S} = \{L \mid \exists A \in \mathcal{C}$ *and* $s \in \mathcal{S}$ *s.t.* $x \in L \Leftrightarrow (x, s_{|x|}) \in A\}$.

In other words, $\mathcal{C}/\mathcal{S}$ is the complexity class computable by Turing machines of complexity class $\mathcal{C}$, but which can use some appropriately sized *advice string* dependent only on the length of the input to aid in computation. Notice that this is more restrictive than being able to use a different advice string for every input. In this lecture, we will restrict our attention to one example of the $\mathcal{C}/\mathcal{S}$ notation: the class $P/poly$.

Definition 2 $P/poly = \{L \mid \exists A \in P$, *a sequence of strings* $\{s_i\}_{i \in \mathbf{N}}$, *and a constant* k *s.t.* $|s_i| = O\left(i^k\right)$ *and* $x \in L \Leftrightarrow (x, s_{|x|}) \in A\}$.

This definition follows directly from that of the $\mathcal{C}/\mathcal{S}$ notation, where $\mathcal{C} = \mathcal{P}$ and our "advice string" must be of length polynomial in the length of the input string. Notice that $P/poly$ is allowing some non-uniformity in its languages. Every language in $P/poly$ has some "magic string" for every input length that simplifies the computation of the rest of the strings. We would like to know if any interesting classes we already know about have this property. For instance, is $P/poly = NP$? We will soon answer this question, but first we introduce the notion of circuit complexity and its relation to $P/poly$.

Definition 3 *Given a boolean function*

$$f : \{0,1\}^a \longrightarrow \{0,1\}^b,$$

the **circuit complexity** *of f, written as C_f, is the least number m such that there exists a circuit with m wires that computes f*

[1] This definition was not covered during this lecture, but touched upon briefly in the previous lecture.

We point out here that our circuit is fixed in size and must be finite. We now define the circuit complexity class analogous to P.

Definition 4 *A language L has* **polynomial circuit complexity** *if there exists a constant k such that for all n, the function f_n that is 1 if and only if its input (of length n) is in L, has circuit complexity $O\left(n^k\right)$.*

We now reveal the somewhat non-intuitive relationship between polynomial circuit complexity and $P/poly$.

Proposition 5 *The complexity class $P/poly$ is equivalent to the class of languages with polynomial circuit complexity. In other words $L \in P/poly$ if and only if L has polynomial circuit complexity.*

Proof $\Leftarrow$: If L has polynomial circuit complexity, then for each n, there is a circuit of size polynomial in n that decides membership in L for all words of length n. We can encode this circuit in a string, s_n, also of size polynomial in n. Now we can construct a polynomial time TM that takes a string x and $s_{|x|}$ and simulates the circuit described by $s_{|x|}$ running on input x.

$\Rightarrow$: Assume $L \in P/poly$. If the machine M that decides L runs in TIME($O\left(n^k\right)$), then we can construct a circuit C_n of size at most $O\left(n^{2k}\right)$ gates that simulates M running on input strings of length n (as in Cook's construction). Hardwired in each machine will be the advice string s_n, which is constant for each input size n and which grows polynomially in n. ■

3 NP and $P/poly$

We will now state the main theorem of this lecture.

Theorem 6 (Karp-Lipton + Sipser) *If $NP \subseteq P/poly$ then $PH \subseteq \Sigma_2^P$. Thus the polynomial hierarchy would "collapse."*

To begin a proof of this theorem, we can first note that $NP \subseteq P/poly$ iff $SAT \in P/poly$. Therefore $NP \subseteq P/poly$ is equivalent to: $\{\exists A \in P$, a constant k, and a sequence $\{g_i\}_{i\in\mathbf{N}}$ s.t. $|g_i| = O\left((ki)^k\right)$ and $\phi \in SAT \Leftrightarrow (\phi, g|\phi|) \in A\}$. So we are now assuming that $NP \subseteq P/poly$ and that we can obtain $A \in P$ with the above property. For the remaining part of this section we will let M be the polynomial-time Turing machine that decides A.

Definition 7 *We define a sequence of strings α to be* good *if it is of the form $\alpha = (\alpha_1, \ldots, \alpha_l)$ s.t. $\forall \phi : |\phi| \leq l, \phi \in SAT \Leftrightarrow (\phi, \alpha_{|\phi|}) \in A$ and $|\alpha_i| \leq (ki)^k$. $GOOD$ is the language of all string sequences that are* good.

Obviously $\{g_1, g_2, \ldots, g_l\}$ as defined above would be a *good* sequence, though it is not necessarily unique.

For the proof of Theorem 6 we will need to show how to decide $GOOD$ efficiently. As a warm-up, we will first show that $GOOD \in \Pi_2^P$.

Lemma 8 *$GOOD \in \Pi_2^P$.*

Proof Assume we are given some sequence α. Then we can construct an ATM that performs the following steps:
On input $(a_1, \ldots, a_l)$

1. Use $\forall$ to guess all strings ϕ s.t. $|\phi| \leq l$.
2. Run M on the input $(\phi, \alpha_{|\phi|})$.
3. If M accepts then existentially ($\exists$) guess a witness to verify that $\phi \in SAT$. If there is a witness, we accept, if not we reject.
4. If M rejects then universally ($\forall$) guess all witnesses to verify that $\phi \notin SAT$. If any witness shows that $\phi \in SAT$ then reject. Otherwise accept.

The language accepted by this ATM is clearly in Π_2^P. Since this ATM directly verifies the condition defined above for a sequence to be *good*, we have $GOOD \in \Pi_2^P$. ■

Before we can prove Theorem 6, we will need to prove a slightly stronger result than the one above. We will require one additional concept, *self-reducibility*, to accomplish this.

4 Self-reducibility

Definition 9 *We define a language L to be* self-reducible *if there exists an oracle polynomial time TM $M^?$ that on inputs of length n only asks its oracle questions of length less than n and that accepts a string w iff $w \in L$. In other words a self-reducible language has a well-formed recursive definition.*

Proposition 10 *SAT is self-reducible.*

Proof To prove this, we note that any instance of SAT is equivalent to deciding the truth of the statement

$$\exists x_1 \exists x_2 \ldots \exists x_n : \phi(x_1, x_2, \ldots, x_n)$$

for a given boolean expression ϕ. We now fix x_1 to be 0 and 1 respectively resulting in a union of two smaller expressions

$$\{\exists x_2 \ldots \exists x_n : \phi(1, x_2, \ldots, x_n)\} \cup \{\exists x_2 \ldots \exists x_n : \phi(0, x_2, \ldots, x_n)\}.$$

This expression is equivalent to the first but is composed of two expressions of smaller size which can be fed to the recursive oracle. After receiving the oracle's answer for the two subexpressions, it is easy to accept if one of the queries returns an accept. We reject otherwise. ■

Using the self-reducibility of SAT, we can now prove a stronger result for $GOOD$ than we proved above.

Proposition 11 *$GOOD \in coNP = \Pi_1^P$.*

Proof The proof involves using self-reducibility to replace the use of $\exists$ in the weaker proof above. Specifically, given a sequence α, we decide whether $\alpha \in GOOD$ as follows:

1. Use $\forall$ to pick all strings ϕ s.t. $|\phi| \leq l$.
2. Run M on input $(\phi, \alpha_{|\phi|})$. If $|\phi| = 1$ then we accept or reject based on whether or not M's answer agrees with ϕ (which must be 0 or 1). Otherwise we continue.
3. If M rejects, then we proceed as we did previously and universally ($\forall$) check all witnesses to be sure that none of them imply that $\phi \in SAT$.
4. If M accepts, then we use the self-reducibility of SAT to generate 2 smaller subexpressions ϕ_1 and ϕ_2. We use $\forall$ to recursively call line 2 with $(\phi_1, \alpha_{|\phi_1|})$ and $(\phi_2, \alpha_{|\phi_2|})$ as inputs. And then accept if either of them accept.

Since we only use $\forall$ branching we can agglutinate all universal quantifiers and see that the machine described above starts in a $\forall$ state and never switches to a $\exists$ state. Therefore $GOOD \in \Pi_1^P = co - NP$. ■

5 Proving the theorem

We can now prove Theorem 6. First, we restate the theorem as follows:

Theorem 12 *If $NP \subseteq P/poly$ then $\Sigma_3^P \subseteq \Sigma_2^P$. This implies that the polynomial hierarchy would collapse to Σ_2^P.*

Proof Consider an instance of QBF_3, which we will remember is complete in Σ_3^P. The instance will have the form $\exists x \forall y \exists z : \phi(x, y, z)$. If we assume $NP \subseteq P/poly$ then we can solve the instance of QBF_3 in Σ_2^P by the following steps:

1. Use $\exists$ to guess each possible advice sequence α.
2. Use $\exists$ to guess each x.
3. Use $\forall$ to guess every y.
4. Test if $\alpha \in GOOD$.
5. If so, use α to decide $(\exists z : \phi(x, y, z), \alpha_{|\exists z:\phi(x,y,z)|}) \in A$.

As we showed above, $GOOD \in co - NP$. We can therefore decide whether α is good using only $\forall$. Furthermore, $(\exists z : \phi(x, y, z), \alpha_{|\exists z:\phi(x,y,z)|}) \in A$ can be decided in P. Therefore, an ATM requires only one alternation starting from $\exists$ to decide an instance of QBF_3, showing $\Sigma_3^P \subseteq \Sigma_2^P$. This completes the proof of Theorem 6. ■