**6.841/18.405J: Advanced Complexity Theory** February 20, 2002

# Lecture 05

*Lecturer: Madhu Sudan* *Scribe: Seth Gilbert*

## Contents

## 1 Collapse of the Polynomial Hierarchy

### What does the polynomial hierarchy look like?

It is an infinite tower of increasing complexity, with each level of the hierarchy nested inside the next level of the hierarchy.

- That is, $\Sigma_i^p \subseteq \Sigma_{i+1}^p$ and $\Sigma_i^p \subseteq \Pi_{i+1}^p$
- And similarly, $\Pi_i^p \subseteq \Sigma_{i+1}^p$ and $\Pi_i^p \subseteq \Pi_{i+1}^p$

### Each level of the hierarchy contains a complete problem.

In particular, for the $i^{th}$ level of the hierarchy, $i$-TQBF is complete. The subset of $i$-TQBF whose first quantifier is $\exists$, is complete for $\Sigma_i^p$, and the subset whose first quantifier is $\forall$ is complete for $\Pi_i^p$.

### Collapse

By *collapse of the polynomial hierarchy*, we mean that the entire hierarchy collapses to a finite level. That is, $\forall i > i_c$ $\Sigma_{i_c}^p = \Sigma_i^p$. (The same is true for $\Pi_i^p$.) Note that this does not imply anything about levels of the hierarchy below the collapse point.

**Theorem 1** *The polynomial hierarchy collapses if and only if the existential and the forall hierarchies are equal. That is:*

$$\Sigma_i^p = \Pi_i^p \iff \Sigma_i^p = \Sigma_j^p, \forall j > i$$

These versions of collapse may seem slightly different, as one is a horizontal collapse, and the other is a vertical collapse. But in fact they are equivalent.
**Proof** Once the incremental case comparing $i$ and $i+1$ is proved, it is clear by induction that it holds for the entire hierarchy (above that point) collapsing. That is, we only prove here that if $\Sigma_i^p = \Pi_i^p$, then $\Sigma_{i+1}^p$ (and the converse), and the rest follows by induction.

**Lemma 2** $\Pi_i^p \subseteq \Sigma_i^p \longrightarrow \Pi_i^p = \Sigma_i^p$

**Proof** If we can show inclusion in one direction, then the two classes are equal. This is true because the two classes are complementary.

$$\begin{array}{rcl} L \in \Pi_i^p & \iff & \bar{L} \in \Sigma_i^p \text{ (because they are complements)} \\ \forall L \in \Sigma_i^p & \longrightarrow & \bar{L} \in \Sigma_i^p \text{ (by the inclusion hypothesis)} \\ & \longrightarrow & \Sigma_i^p \subseteq \Pi_i^p \end{array}$$

■

First, assume that the polynomial hierarchy collapses at level $i$. That is, $\Sigma_i^p = \Sigma_{i+1}^p$. Since $\Pi_i^p = \Sigma_{i+1}^p$, this clearly implies that $\Sigma_i^p = \Pi_i^p$.

Next, assume $\Sigma_i^p = \Pi_i^p$. Start with $(i+1)$-TQBF, a complete problem for $\Sigma_{i+1}^p$. We will show how to solve this in $\Sigma_i^p$. Define:

$$L' = \{(\phi, x_1) | \forall x_2 \exists x_3 \cdots \mathrm{Q}_{i+1} x_{i+1}, \phi(x_1, x_2, x_3, \ldots, x_{i+1}) = 1\}$$

That is, $L'$ is the language of (i+1)-TQBF problems and $x_1$'s that allow the TQBF to be true. It is clear that $L' \subseteq \Pi_i^p \subseteq \Sigma_i^p$, as the first quantifier is a $\forall$, and there are only $i$ alternations.. Therefore, we can construct another language (say $L''$) that is in $\Sigma_i^p$. In particular:

$$L' \leq \{\psi | \exists y_1 \forall y_2 \cdots \mathrm{Q}_i y_i, \psi(y_1, y_2, \ldots, y_i)\}$$

This $\psi_{(\phi,x_1)}$ is constructed so that $(\phi, x) \in L' \iff \psi \in i - \text{TQBF}$. The following is the procedure to decide the $(i+1)$-TQBF in $\Sigma_i^p$.

1. Guess $x_1$
2. Compute $\psi_{(\phi,x_1)}$
3. Guess $y_1$
4. ForAll $y_2$, Guess $y_3$ …
5. Verify $\psi(y_1, \ldots, y_i) = 1$

Note that steps (1–3) involve only a single $\exists$ quantifier. That is, only a single alternation. And the rest of the $y_i$ require only $i-1$ alternations. Therefore we can solve $(i+1)$-TQBF with only $i$ alternations, that is, in $\Sigma_i^p$, which proves that $\Sigma_i^p = \Sigma_{i+1}^p$. ■

Note that this only holds for $i > 0$ This proof generally indicates that the ability to switch qualifiers is powerful and carries up the hierarchy, once equivalence is found at any level.

It hasn't been proven that the polynomial hierarchy does not collapse, but we generally believe that it does not collapse. We often therefore assume that the polynomial hierarchy does not collapse, and use this to prove other theorems. This is a strong assumption, however it allows us to reduce many of our conjectures to a single assumption. We will see some conjectures supported by this assumption later.

## 2 Circuit Complexity (Non-Uniform Complexity)

Circuit complexity is slightly different, but essentially the same technically as non-uniform complexity. It provides a new approach to the problem of $P \neq NP$.

## Circuits

Circuits are a model of computation with a fixed number of input bits. A Turing Machine works as follows:

1. Design algorithm.
2. Decide input to algorithm.

A circuit works as follows:

1. Decide length of input, $n$
2. Design algorithm to solve problems of length $n$
3. Decide input to algorithm (of length $n$)

## Formally

- A circuit is a directed, acyclic graph.
- 3 types of nodes:
    1. input: $x_1, \ldots, x_n$
    2. output: $o_1, \ldots, o_n$
    3. computation: OR, AND, NOT
- Each set of nodes is disjoint (for now)
- Fan-in: the number of edges going into a node

| Node Type | Fan-in |
|---|---|
| Input | 0 |
| Output | 1 |
| OR, AND | 2 |
| NOT | 1 |

- Fan-out: the number of edges leaving a node

| Node Type | Fan-out |
|---|---|
| Output | 0 |
| All Others | $\infty$ |

- To determine output:
    1. Topological sort (linear time)
    2. Calculate each level of the tree (linear time)
- Circuit computes some function $f : \{0,1\}^n \to \{0,1\}^n$.
- What resource to measure? Size of circuit = # of gates/nodes
    - Sometimes count # of wires/edges, but one measure is quadratic in the other
    - If you assume maximum fan-out of two, then at most twice as many wires as nodes
- Can compute function in (deterministic) time linear on # of nodes/edges

- Therefore, size is a good measure of how efficiently a circuit can compute.

What is interesting, however, is not determining how big a circuit is for a particular constant size. Instead, we want to define a set of functions, parameterized on the input size, $n$. We want to examine the size of the smallest circuit that can compute the function for that input. This gives us something interesting to measure as $n$ gets large. That is, we can study size as a function of $n$.

Consider a family of boolean functions $\{f : \{0,1\}^n \to \{0,1\}\}_{n=1}^{\infty}$. How does the smallest circuit computing $f_n$ grow with $n$?

This pins down the function once we set $n$. There is an absolute minimum for each $n$. This therefore provides a more concrete and combinatorial approach to examining complexity.

A common example of family circuits: $\phi_n$, the SAT family.

- $f_n$ takes as input CNF formula $\phi$ of length $n$
- $f_n = 1$ if and only if $\phi \in SAT$

If P=NP, the SAT family of circuits will be polynomial sized in $n$

- Draw TM tableau
- Create boolean formula to implement

So if we prove that $f_n$ has no polynomial sized circuit , then $NP \neq P$. Unfortunately, we don't know how to do this.

## Non-Uniform Complexity

- Modify/enhance TM so that we only worry about length $n$ inputs
- TM may work differently on different length inputs
- TM works as follows:
    1. Fix machine M
    2. Given length $n$
    3. Determine advice $a1, \dots, a_n$
        - $a_i$ used to decide language
        - binary strings on $\{0,1\}^{p(n)}$
        - length of advice polynomial in $n$
    4. Given input $x \in \{0,1\}^n$
    5. Run $M(x, a_n)$ to decide if $x \in L$ or not.

**Definition 3** *P/Poly* $= \{L|$

- $\exists M \in P$
- *polynomial* $p(.)$
- $\{a_1, \dots, a_n\}$ *s.t.* $|a_i| = p(i)$

*such that* $\forall x \in \{0,1\}^*, x \in L \iff M(x, a_{|x|})$*accepts*$\}$

This provides a concrete way of asking how to use input length without defining a particular model of computation (i.e. circuits).

**Lemma 4** *P/Poly = class of functions that have polynomial sized circuits*

**Proof** If there is a known polynomial circuit for a given problem, then design $M$ to calculate the value of a circuit, and have the advice strings be the circuits for various sized inputs. $M$ using the advice can then clearly decide the output. Conversely, if we have a TM that can use advice to solve the problem, create a circuit to simulate the computation on the TM and build the advice into the circuit.

**Lemma 5** *Any problem in P has polynomial-sized circuits*

**Proof** Take language $L \in P$ decided by algorithm $A$. The goal is to find $C_n = A(x)$, where $|x| = n$. Construct a tableau for the TM's computation. It is of size $p(n) \times p(n)$. For every cell, for every $\sigma$ contents of the cell, let $x_{c,\sigma} = 1$ if (and only if) the cell has a $\sigma$ on it.

Compute $x_{c,\sigma}$ based on $x_{c_{i-1},\sigma_1}, x_{c_i,\sigma_2}, x_{c_{i+1},\sigma_3} \forall \sigma$. There is clearly a small circuit that will encode this piece of the TM computation. In this way all the variables can be determined by the circuit. If $x_{c_{final},\sigma_{accept}} = 1$, accept, else reject.

This circuit is at most a polynomial size larger than the TM; it blows up no more than $p^2$. (And, as before, Cooke was able to reduce this to something like $n \log n$. ■

Using this, we can easily see that it is possible to build a polynomial sized circuit that solves the problem using advice. ■

## Question: Is P/Poly more powerful than P?

- *BPP* $\subseteq$ *P/Poly*
- Every unary language (where all strings are of the form $1^n$) is in P/Poly
  - Even undecidable unary languages in P/Poly
  - There is only one string of a given length, so hardwire the advice string
  - There is one advice string per input string, so the advice string indicates whether the appropriate unary string is in the language
- Every unary language is *not* in P. (Some unary language are undecidable.)
- P/Poly is not contained in P, NP, PSPACE
- P/Poly includes undecidable languages
- Conclusion: P/Poly is more powerful than P??

## Question: Is $NP \subseteq$ P/Poly?

Can you give polynomial advice to solve SAT?
Belief: $NP \nsubseteq P/Poly$ We want to relate all our beliefs to a single belief: the polynomial hierarchy collapse.

## 3 Karp-Lipton Theorem

**Theorem 6** *If NP $\subseteq$ P/Poly, then the Polynomial Hierarchy collapses.*

**Proof** For the sake of contradiction, suppose NP $\subseteq$ P/Poly. This implies that that we have $M, P, (a_1, \ldots, a_n)$ such that:

$$M(\phi, a_{|\sigma|}) = 1 \iff \phi \in SAT$$

Goal: Show $\Sigma_{i+1}^p \subseteq \Sigma_i^p$. We need a Turing Machine with $i$ alternations. We can compress $M$ and $P$, but we cannot compress all $a_n$ since there are infinitely many advice strings. So we need to figure out how to generate/encode an infinite sequence of advice strings.
Given $\phi \in (i+1)-$TQBF, need to compute right $a_n$ for that $\phi$.

**Definition 7** *$a_n$ is* GOOD *if it functions as right advice.*

As long as it causes $M$ to compute correctly, it is good, regardless of whether it is the same as the original $a_n$. That is, $M(\phi, a_n) = 1 \iff \phi \in SAT$, for all $\phi$ of length $n$. We can decide this property efficiently: in the 2nd level of the polynomial hierarchy.

**Lemma 8** *$a_n \in$* GOOD *can be decided in $\Pi_2^p$.*

**Proof** We know how to check all $\phi$ of length $n$ using a *forall* quantifier (that is, it is in co-NP).

$$\forall \phi \text{ of length } n, M(\phi, a_n) = \text{“}\phi \in SAT\text{”}$$

But we know how to check if $\phi \in SAT$ using a single existential quantifier. We just ask an NP oracle. And we already know that co-NP$^{NP} = \Pi_2^p$. ■

Further, we can then find $a_n$ in $\Sigma_3^P$, since we can use the first existential state to guess the advice string, and then use the remaining two levels to verify.

**Lemma 9** *If $a_n \in$* GOOD *is in $\Pi_{i-1}^p$, then $\Sigma_{i+1}^p \subseteq \Sigma_i^p$.*

**Proof** Let $i$ be even. Take an example of $(i+1)-$TQBF. We want to decide:

$$\{\psi | \exists x_1 \forall x_2 \cdots Q_{i+1} x_{i+1}, \psi(x_1, x_2, \ldots, x_{i+1}\}$$

Given a set of $x_1, \ldots, x_n$, define $\phi$ as a function of the last variable of $\psi$. That is:

$$\phi(y) = \psi(x_1, x_2, \ldots, x_i, y)$$

In order to decide whether $\phi \in SAT$, we can use the P/Poly machine M, along with the advice $a_{|\phi|}$. Given $a_n \in$ GOOD, we can determine whether $\phi \in SAT$ in deterministic polynomial time.

How do we determine $a_n$? By the lemma above, we can guess and verify $a_n$ is three alternations. However, in parallel, we also need to guess and verify $x_1, \ldots, x_n$ using $n$ alternations.

The following is a $\Sigma_i^p$ machine for deciding if $\psi \in (i+1)-$TQBF.

1. Guess $a_n$
2. Guess $x_1$
3. ForAll: $a_n$ is GOOD
4. ForAll: $x_2$
5. Guess $x_3$

6. Repeat alternating ForAll and Guess
7. ForAll: $x_i$
8. Calculate $\phi$, given $x_1, \ldots, x_i$ guessed above
9. Calculate in deterministic polynomial time $M(\phi, a_n)$ and accept if and only if $\phi \in SAT$.

Notice that this machine uses only $i$ alternations. Steps 1–2 use a single $\exists$, and steps 3–4 use a single $\forall$. The guessing and verifying of $a_n$ happen in parallel with guessing and verifying $x_1, \ldots, x_i$. By using the P/Poly deterministic machine to solve the last SAT problem, we are avoiding one level of the hierarchy. As a result, this $\Sigma_i^p$ machine can solve $(i+1)-$TQBF, showing that the polynomial hierarchy collapses. ■

■