Lecture 2

Polynomial Time Hierarchy and Nonuniform Complexity

April 1, 2004
Lecturer: Paul Beame
Notes: Ioannis Giotis

2.1 Definition

Recall the following definitions of relativized complexity classes.

Definition 2.1. For any language A define $\mathsf{P}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle TM $\}$ and $\mathsf{NP}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle NTM $\}$.
For any complexity class C define $\mathsf{P}^{\mathsf{C}} = \bigcup_{A\in\mathsf{C}} \mathsf{P}^A$, $\mathsf{NP}^{\mathsf{C}} = \bigcup_{A\in\mathsf{C}} \mathsf{NP}^A$,

Definition 2.2. We define the classes in the polynomial-time hierarchy by

$$\begin{aligned}
\Sigma_0\mathsf{P} &= \Pi_0\mathsf{P} = \Delta_0\mathsf{P} = \mathsf{P} \\
\Delta_{i+1}\mathsf{P} &= \mathsf{P}^{\Sigma_i\mathsf{P}} \\
\Sigma_{i+1}\mathsf{P} &= \mathsf{NP}^{\Sigma_i\mathsf{P}} \\
\Pi_{i+1}\mathsf{P} &= \mathsf{coNP}^{\Sigma_i\mathsf{P}}
\end{aligned}$$

Unwinding the definition we obtain some basic complexity classes.

$$\begin{aligned}
\Delta_1\mathsf{P} &= \mathsf{P}^{\mathsf{P}} = \mathsf{P} \\
\Sigma_1\mathsf{P} &= \mathsf{NP}^{\mathsf{P}} = \mathsf{NP} \\
\Pi_1\mathsf{P} &= \mathsf{coNP}^{\mathsf{P}} = \mathsf{coNP} \\
\Delta_2\mathsf{P} &= \mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{\mathrm{SAT}} \supseteq \mathsf{coNP} \\
\Sigma_2\mathsf{P} &= \mathsf{NP}^{\mathsf{NP}} \\
\Pi_2\mathsf{P} &= \mathsf{coNP}^{\mathsf{NP}}
\end{aligned}$$

An example language in P^{NP} is the following, which is the difference between $\{\langle G,k\rangle \mid G$ has a k-clique$\}$ and $\{\langle G,k\rangle \mid G$ has a $(k+1)$-clique$\}$.

$$\textsc{Exact-Clique} = \{\langle G,k\rangle \mid \text{the largest clique in } G \text{ has size } k\}$$

Observe that, since oracle TMs can easily flip the answers they receive from the oracle, $\mathsf{P}^{\Sigma_i\mathsf{P}} = \mathsf{P}^{\Pi_i\mathsf{P}}, \mathsf{NP}^{\Sigma_i\mathsf{P}} = \mathsf{NP}^{\Pi_i\mathsf{P}}, \mathsf{coNP}^{\Sigma_i\mathsf{P}} = \mathsf{coNP}^{\Pi_i\mathsf{P}}$

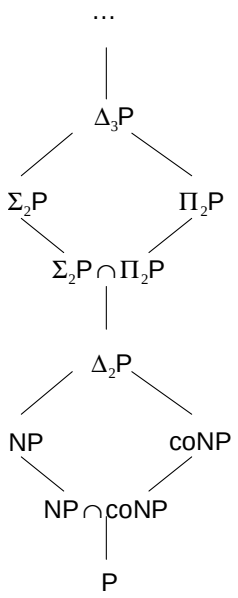

Figure 2.1: The polynomial time hierarchy

Definition 2.3. The *polynomial-time hierarchy* $\mathsf{PH} = \bigcup_k \Sigma_k\mathsf{P} = \bigcup_k \Pi_k\mathsf{P}$

Probably the most important question about the polynomial-time hierarchy is given by the following conjecture.

Conjecture 2.1. $\forall k\ \mathsf{PH} \neq \Sigma_k\mathsf{P}$.

Note that this very natural conjecture generalizes the conjectures that $\mathsf{P} \neq \mathsf{NP}$, since if $\mathsf{P} = \mathsf{NP}$ then $\mathsf{PH} = \mathsf{P} = \Sigma_0\mathsf{P}$, and that $\mathsf{NP} \neq \mathsf{coNP}$, since in that case $\mathsf{PH} = \mathsf{NP} = \Sigma_1\mathsf{P}$. Several results in this course will be conditional based on this conjecture, namely that some natural property will hold unless the polynomial-time hierarchy PH collapses to some level, say $\Sigma_k\mathsf{P}$.

2.2 Alternative Characterization of the Hierarchy Classes

The following characterization of languages in $\Sigma_k\mathsf{P}$ is useful for obtaining a simpler alternative characterization of PH.

Theorem 2.2. *$L \in \Sigma_{i+1}\mathsf{P}$ if and only if there exists a language $R \in \Pi_i\mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $L = \{x|\ \exists^{p(|x|)}y.\ (x, y) \in R\}$.*

Proof. By induction. Base case $i = 0$ follows from the alternate characterization of NP: $L \in \mathsf{NP} \Leftrightarrow$ there is some $R \in \mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $L = \{x \mid \exists^{p(|x|)}y.\ (x, y) \in R\}$

For the induction step, $i > 0$ let $L \in \Sigma_{i+1}\mathsf{P}$. We will show how to build a certificate y and a relation $R \in \Pi_i\mathsf{P}$ suitable for the characterization. By definition

$$L \in \mathsf{NP}^{\Sigma_i\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}^{\Pi_{i-1}\mathsf{P}}}$$

which is well-defined since $i - 1 \geq 0$. This means that there is a polynomial-time oracle NTM $M^?$ such that if $M^?$ is given an oracle for a $\Sigma_i P$ set A, $L(M^A) = L$.

We consider the various parts of the computation on input x for L. $M^?$ has nondeterministic moves so part of the certificate y for x will consist of the nondeterministic moves of $M^?$; the certificate y will also contain the values of all the answers that $M^?$ receives from the oracle for A. Let these parts of y be $\tilde{y}$. Given $\tilde{y}$ in polynomial time we can check that the computation of $M^?$ given the oracles answers could follow the nondeterministic moves and accept.

The problem is that we don't know if the oracle answers purported to be according to A are correct. Given that the computation of $M^?$ is consistent with $\tilde{y}$, we have fixed the polynomially many oracle queries $z_1 \ldots, z_m$, say, that will be made to the oracle for A. The rest of the certificate y will be certificates that each of the answers given for A on these strings is actually correct.

We can verify each of the *yes* answers to A as follows: By applying the inductive hypothesis to A there exists a set $R' \in \Pi_{i-1}\mathsf{P}$ (and a polynomial q) such that

$$z_i \in A \Leftrightarrow \exists^{q(|x|)} y_i.(z_i, y_i) \in R'$$

If the answer to $z_i \in A$ is *yes* then we include this y_i in the certificate y. Since $R' \in \Pi_{i-1}\mathsf{P} \subseteq \Pi_i\mathsf{P}$ the algorithm for R can simply check that $(z_i, y_i) \in R'$ for each query z_i to A that is answered *yes*.

If the answer to whether or not $z_i \in A$ is *no* then $z_i \in \overline{A} \in \Pi_i\mathsf{P}$. Thus the new $\Pi_i\mathsf{P}$ machine R will check $\tilde{y}$, y_i if the answer was *yes*, and z_i directly if the answer for z_i was *no*. □

Corollary 2.3. *$L \in \Pi_{i+1}\mathsf{P}$ if and only if there is a relation $R \in \Sigma_{i+1}\mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $L = \{x|\ \forall^{p(|x|)} y.\ (x, y) \in R\}$.*

Corollary 2.4. *$L \in \Sigma_i\mathsf{P}$ if and only if there is a polynomial-time computable set R and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that*

$$L = \{x|\ \exists^{p(|x|)} y_1 \forall^{p(|x|)} y_2 \cdots Q y_i.\ (x, y_1, y_2, \ldots, y_i) \in R\}$$

where $Q = \begin{cases} \exists^{p(|x|)} & \text{if } i \text{ is odd,} \\ \forall^{p(|x|)} & \text{if } i \text{ is even.} \end{cases}$

2.2.1 Some $\Sigma_i\mathsf{P}$- and $\Pi_i\mathsf{P}$-complete Problems

Definition 2.4. Define
$\Sigma_i\mathrm{TQBF} = \{\langle\psi\rangle|\ \psi \in \mathrm{TQBF}$ and the quantifiers of ψ are of the form $\overrightarrow{\exists} y_1 \overrightarrow{\forall} y_2 \cdots Q y_k \psi Q\}$;
i.e. there are k groups of alternating quantifiers beginning with $\exists$. Similarly we can define $\Pi_i\mathrm{TQBF}$ with k groups of alternations beginning with $\forall$.

Theorem 2.5. *For $i \geq 1$, $\Sigma_i\mathrm{TQBF}$ is $\Sigma_i\mathsf{P}$-complete and $\Pi_i\mathrm{TQBF}$ is $\Pi_i\mathsf{P}$-complete.*

Proof Sketch. We apply the characterizations of $\Sigma_i\mathsf{P}$ and $\Pi_i\mathsf{P}$ from Corollary 2.4. The quantifiers match up perfectly except that the polynomial-time computable set R has been replaced by a simple Boolean formula. We cannot in general replace polynomial-time computation by Boolean formulas however, we can replace it in a way that is similar to the tableau from Cook's proof of the NP-completeness of SAT.

First suppose that the last quantifier for the complexity class is an $\exists$ quantifier. We can add additional Boolean variables z that represent the internal states of the computation of R and create a formula φ such that $(x, y_1, \ldots, y_i) \in R$ if and only if $\exists z \varphi(x, y_1, \ldots, y_i, z)$. Since the last quantifier was already an $\exists$ quantifier we can append z to it and simulate the formula.

If the last quantifier for the complexity class is a $\forall$ quantifier then we use the same internal variables z except that the formula for R is now $\forall z(\varphi'(x, y_1, \ldots, y_i, z) \rightarrow \varphi''(z))$ where φ' ensures that z is a correct computation on input $(x, y_1, \ldots, y_i)$ and φ'' ensures that z is accepting. In this case the $\forall z$ merges with the last $\forall$ in the alternation. □

2.2.2 Hierarchy Collapse

Lemma 2.6. *If* $\Sigma_k\mathsf{P} = \Pi_k\mathsf{P}$ *then* $\mathsf{PH} = \Sigma_k\mathsf{P} \cap \Pi_k\mathsf{P}$

Proof. We'll show that assumption implies that $\Sigma_{k+1}\mathsf{P} = \Sigma_k\mathsf{P}$ which will suffice. Let $A \in \Sigma_{k+1}\mathsf{P}$. Therefore, there is some polynomial-time R and polynomial p such that

$$A = \{x|\ \exists^{p(|x|)} y_1 \quad \underbrace{\forall^{p(|x|)} y_2 \cdots Q y_{k+1}.\ (x, y_1, y_2, \ldots, y_{k+1}) \in R}_{\in \Pi_k\mathsf{P} = \Sigma_k\mathsf{P}} \quad \}.$$

Therefore there is some polynomial-time relation R' and polynomial p' such that A can be expressed as

$$A = \{x| \quad \underbrace{\exists^{p(|x|)} y_1 \exists^{p'(|x|)} y_1'}_{\exists^{p''(|x|)}(y_1, y_1')} \quad \forall^{p'(|x|)} y_2' \cdots Q^{p'(|)} y_k'.\ (x, y_1, y_1', y_2', \ldots, y_k') \in R'\}$$

From which it follows that $A \in \Sigma_k\mathsf{P}$. □

If this happens we say that PH collapses to the k-th level.

2.2.3 A Natural Problem in $\Pi_2\mathsf{P}$

Definition 2.5. Let MINCIRCUIT $= \{\langle C\rangle|\ C$ is a circuit that is not equivalent to any smaller circuit$\}$.

Note that $\langle C\rangle \in$ MINCIRCUIT $\Leftrightarrow \forall\langle D\rangle, size(D) < size(C), \exists y$ s.t. $D(y) \neq C(y)$ Thus MINCIRCUIT is in $\Pi_2\mathsf{P}$. It is still open if it is in $\Sigma_2\mathsf{P}$ or if it is $\Pi_2\mathsf{P}$-complete.

2.3 Non-uniform Complexity

The definitions of Turing machines yield finite descriptions of infinite languages. These definitions are *uniform* in that they are fixed for all input sizes. We now consider some definitions of *non-uniform* complexity classes.

2.3.1 Circuit Complexity

Definition 2.6. Let $\mathbb{B}_n = \{f|\ f : \{0,1\}^n \rightarrow \{0,1\}\}$. A *basis* Ω is defined as a finite subset of $\bigcup_n \mathbb{B}_n$.

Definition 2.7. A *Boolean circuit over basis* Ω is a finite directed acyclic graph C each of whose nodes is

- a source node labelled by either an *input variable* in $\{x_1, x_2, \ldots\}$ or *constant* $\in \{0,1\}$, or

- a node of in-degree $d > 0$ labeled by a *gate* function $g \in \mathbb{B}_d \cap \Omega$,

and which has one designated *output node* (gate). A circuit C has two natural measures of complexity, its *size*, $size(C)$, which is the number of gates in C and its *depth*, $depth(C)$ which is the length of the longest path from an input (source) to the output node of C.

Typically the elements of Ω we use are symmetric and unless otherwise specified we will assume the so-called De Morgan basis $\Omega = \{\wedge, \vee, \neg\} \subseteq \mathbb{B}_1 \cup \mathbb{B}_2$.

Definition 2.8. C is *defined on* $\{x_1, x_2, \ldots, x_n\}$ if its input variables are contained in $\{x_1, x_2, \ldots, x_n\}$. C defined on $\{x_1, x_2, \ldots, x_n\}$ computes a function $f \in \mathbb{B}_n$ in the obvious way.

Definition 2.9. A *circuit family* C is an infinite sequence of circuits $\{C_n\}_{n=0}^{\infty}$ such that C_n is defined on $\{x_1, x_2, \ldots, x_n\}$.

Circuit family C has size $S(n)$, depth $d(n)$, iff for each n

$$\begin{aligned} size(C_n) &\leq S(n) \\ depth(C_n) &\leq d(n). \end{aligned}$$

Circuit family C *decides* or *computes* a language $A \subseteq \{0,1\}^*$ iff for every input $x \in \{0,1\}^*$, $C_{|}x|(x) = 1 \Leftrightarrow x \in A$.

Definition 2.10. We say $A \in \mathsf{SIZE}(S(n))$ if there exists a circuit family over the De Morgan basis of size $S(n)$ that computes A. Similarly we define $A \in \mathsf{DEPTH}(d(n))$.

$$\mathsf{POLYSIZE} = \bigcup_k \mathsf{SIZE}(n^k + k)$$

Our definitions of size and depth complexity classes somewhat arbitrarily care about constant factors. For Turing machines we are stuck with them because alphabet sizes are variables but for circuits the complexity does not involve such natural constant factors. (We may regret decision this later!)

Remark. $\exists A \in \mathsf{POLYSIZE}$ such that A is not decidable.