Lecture 1

Polynomial Time Hierarchy

April 1, 2008
Lecturer: Paul Beame
Notes:

1.1 Polynomial Time Hierarchy

We first define the classes in the polynomial-time hierarchy.

Definition 1.1 For each integer i, define the complexity class Σ_i^p to be the set of all languages L such that there is a polynomial time Turing machine M and a polynomial q such that

$$x \in L \Leftrightarrow \exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \cdots Q_i y_i \in \{0,1\}^{q(|x|)}.M(x, y_1, \ldots, y_i) = 1$$

where

$$Q_i = \begin{cases} \forall & \text{if } i \text{ is even} \\ \exists & \text{if } i \text{ is odd} \end{cases}$$

,
and define the complexity class Π_i^p to be the set of all languages L such that there is a polynomial time Turing machine M and a polynomial q such that

$$x \in L \Leftrightarrow \forall y_1 \in \{0,1\}^{q(|x|)} \exists y_2 \in \{0,1\}^{q(|x|)} \cdots Q_i y_i \in \{0,1\}^{q(|x|)}.M(x, y_1, \ldots, y_i) = 1$$

where

$$Q_i = \begin{cases} \exists & \text{if } i \text{ is even} \\ \forall & \text{if } i \text{ is odd} \end{cases}.$$

(It is probably more consistent with notations for other complexity classes to use the notation $\Sigma_i\mathsf{P}$ and $\Pi_i\mathsf{P}$ for the classes Σ_i^p and Π_i^p but the latter is more standard notation.)

The *polynomial-time hierarchy* is $\mathsf{PH} = \bigcup_k \Sigma_k^p = \bigcup_k \Pi_k^p$.

Observe that $\Sigma_0^p = \Pi_0^p = \mathsf{P}$, $\Sigma_1^p = \mathsf{NP}$ and $\Pi_1^p = \mathsf{coNP}$. Here are some natural problems in higher complexity classes.

$$\textsc{Exact-Clique} = \{\langle G, k\rangle \mid \text{the largest clique in } G \text{ has size } k\} \in \Sigma_2^p \cap \Pi_2^p$$

since TM M can check one of its certificates is a k-clique in G and the other is not a $k+1$-clique in G.

$$\textsc{MinCircuit} = \{\langle C\rangle \mid \text{ C is a circuit that is not equivalent to any smaller circuit}\} \in \Pi_2^p$$

since

$$\langle C\rangle \in \textsc{MinCircuit} \Leftrightarrow \forall\langle C'\rangle\exists y\ s.t.\ (size(C') \geq size(C) \vee C'(y) \neq C(y)).$$

It is still open if $\textsc{MinCircuit}$ is in Σ_2^p or if it is Π_2^p-complete However, Umans [1] has shown that the analogous problem MinDNF is Π_2^p-complete (under polynomial-time reductions).

Define

$$\Sigma_i\mathrm{SAT} = \{\langle\varphi\rangle \mid \varphi \text{ is a Boolean formula s.t. } \exists y_1 \in \{0,1\}^n \forall y_2 \in \{0,1\}^n \cdots Q_i y_i \in \{0,1\}^n \varphi(y_1,\ldots,y_i) \text{ is true}\}.$$

and define Π_iSAT similarly. Theorem we can convert the Turing machine computation into a Booleam formula and show that Σ_iSAT is Σ_i-complete and Π_iSAT is Π_i-complete.

It is generally conjectured that $\forall i, \mathsf{PH} \neq \Sigma_i^p$.

Lemma 1.2 $\Pi_i^p \subseteq \Sigma_i^p$ implies that $\mathsf{PH} = \Sigma_i^p = \Pi_i^p$.

1.1.1 Alternative definition in terms of oracle TMs

Definition 1.3 An oracle TM $M^?$ is a Turing machine with a separate oracle input tape, oracle query state q_{query}, and two oracle answer states, q_{yes} and q_{no}. The content of the oracle tape at the time that q_{query} is entered is given as a query to the oracle. The cost for an oracle query is a single time step. If answers to oracle queries are given by membership in a language A, then we refer to the instantiated machine as M^A.

Definition 1.4 Let $\mathsf{P}^A = \{L(M^A) \mid M^? \text{ is a polynomial-time oracle TM}\}$, let $\mathsf{NP}^A = \{L(M^A) \mid M^? \text{ is a polynomial-time oracle NTM}\}$, and $\mathsf{coNP}^A = \{\overline{L} \mid L \in \mathsf{NP}^A\}$.

Theorem 1.5 For $i \geq 0$, $\Sigma_{i+1}^p = \mathsf{NP}^{\Pi_i^p}$ $(= \mathsf{NP}^{\Sigma_i^p})$.

Proof $\Sigma_{i+1}^p \subseteq \mathsf{NP}^{\Pi_i^p}$: The oracle NTM simply guesses y_1 and asks (x, y_1) for the Π_i^p oracle for $\forall y_2 \in \{0,1\}^{q(|x|)} \ldots Q_{i+1} y_{i+1} \in \{0,1\}^{q(|x|)}.M(x, y_1, y_2, \ldots, y_{i+1}) = 1$.

$\mathsf{NP}^{\Pi_i^p} \subseteq \Sigma_{i+1}^p$: Given a polynomial-time oracle NTM $M^?$ and a Π_i^p language A then $x \in L = L(M^A)$ if and only if there is an accepting path of M^A on input x.

To describe this accepting path we need to include a string y consisting of

- the polynomial length sequence of nondeterministic moves of $M^?$,
- the answers $b_1, \ldots, b_m$ to each of the oracle queries during the computation,
- the queries $z_1, \ldots, z_m$ given to A during the computation,

(Note that each of $z_1, \ldots, z_m$ is actually determined by a deterministic polynomial time computation given the nondeterministic guesses and prior oracle answers so this can be checked at the end.) However, we need to ensure that each oracle answer b_i is actually the answer that the oracle A would return on inputs z_i.

If all the answers b_i were *yes* answers then after an existential quantifier for $y_1 = y$ we could simply check that $(z_1, \ldots, z_m)$ are the correct queries by checking that they are in A^m which is in Π_i^p since $A \in \Pi_i^p$.

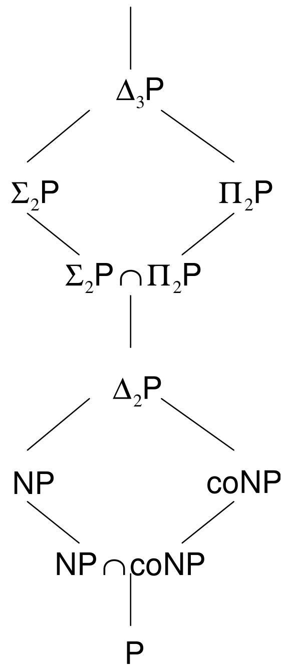

Figure 1.1: The First Levels of the Polynomial-Time Hierarchy

The difficulty is that verifying the *no* answers is a Σ_i^p problem (which likely can't be expressed in Π_i^p). The trick to handle this is that since $\overline{A} \in \Sigma_i^p$, there is some $B \in \Pi_{i-1}^p \subseteq \Pi_i^p$ and polynomial p such that $z_j \notin A$ iff $\exists y'_j \in \{0,1\}^{p(|x|)}.(z_j, y'_j) \in B$.

Therefore, to express L using a existentiallly quantified variable y_1 that includes y as well as all y'_j such that the query answer b_j is *no*. It follows that $x \in L$ iff $\exists y_1, (x, y_1) \in A'$ for some Π_i^p set A' and thus $L \in \Sigma_{i+1}^p$. $\square$

It follows also that $\Pi_{i+1}^p = \mathsf{coNP}^{\Sigma_i^p}$ for $i \geq 0$. This naturally also suggests the definition:

$$\begin{aligned} \Delta_0^p &= \mathsf{P} \\ \Delta_{i+1}^p &= \mathsf{P}^{\Sigma_i^p} \qquad \text{for } i \geq 0. \end{aligned}$$

Observe that $\Delta_i^p \subseteq \Sigma_i^p \cap \Pi_i^p$ and

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

Also, observe that in fact ExactClique is in $\Delta_2^p = \mathsf{P}^{\mathsf{NP}}$ by querying Clique on $\langle G, k\rangle$ and $\langle G, k+1\rangle$.

1.2 Non-uniform Complexity

1.2.1 Circuit Complexity

Let $\mathbb{B}_n = \{f \mid f : \{0,1\}^n \to \{0,1\}\}$. A basis Ω is a subset of $\bigcup_n \mathbb{B}_n$.

Definition 1.6 A Boolean circuit over basis Ω is a finite directed acyclic graph C each of whose nodes is either

1. a source node labelled by either an input variable in $\{x_1, x_2, \ldots\}$ or constant $\in \{0,1\}$, or
2. a node of in-degree $d > 0$ called a *gate*, labelled by a function $g \in \mathbb{B}_d \cap \Omega$.

There is a sequence of designated output gates (nodes). Typically there will just be one output node. Circuits can also be defined as straight-line programs with a variable for each gate, by taking a topological sort of the graph and having each line describes how the value of each variable depends on its predecessors using the associated function.

Say that Circuit C is defined on $\{x_1, x_2, \ldots, x_n\}$ if its input variables $\subseteq \{x_1, x_2, \ldots, x_n\}$. C defined on $\{x_1, x_2, \ldots, x_n\}$ computes a function in the obvious way, producing an output bit vector (or just a single bit) in the order of the output gate sequence.

Typically the elements of Ω we use are symmetric. Unless otherwise specified $\Omega = \{\wedge, \vee, \neg\} \subseteq \mathbb{B}_1 \cup \mathbb{B}_2$.

Definition 1.7 A circuit family C is an infinite sequence of circuits $\{C_n\}_{n=0}^{\infty}$ such that C_n is defined on $\{x_1, x_2, \ldots, x_n\}$

$size(C_n)$ = number of nodes in C_n.

$depth(C_n)$ =length of the longest path from input to output.

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

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

We say that $A \in \mathsf{SIZE}_\Omega(S(n))$ if there exists a circuit family of size $S(n)$ that computes A. Similarly we define $A \in \mathsf{DEPTH}_\Omega(d(n))$. When we have the De Morgan basis we drop the subscxript Ω. Note that if another (complete) basis Ω is finite then it can only impact the size of circuits by a constant factor since any gate with fan-in d can be simulated by a CNF formula of size $d2^d$. We write $\mathsf{POLYSIZE} = \bigcup_k \mathsf{SIZE}(n^k + k)$.

There are undecidable problems in $\mathsf{POLYSIZE}$. In particular

$$\{1^n \mid \text{ Turing machine } M_n \text{ accepts } \langle M_n \rangle\} \in \mathsf{SIZE}(1)$$

as is any unary language.

Next time we will prove the following theorem due to Karp and Liption:

Theorem 1.8 (Karp-Lipton) If $\mathsf{NP} \subseteq \mathsf{POLYSIZE}$ then $\mathsf{PH} = \Sigma_2^p \cap \Pi_2^p$.

References

[1] C. Umans. The minimum equivalant dnf problem and shortest implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001.