CSE200 Lecture Notes – Polynomial Hierarchy

Lecture by Russell Impagliazzo
Notes by Jiawei Gao

February 18, 2016

1 Polynomial Hierarchy

Recall from last class that for language L, we defined

- P^L is the class of problems poly-time Turing reducible to L.
- NP^L is the class of problems with witnesses verifiable in P^L.

In other words, these problems can be considered as computed by deterministic or nondeterministic TMs that can get access to an oracle machine for L.

The polynomial hierarchy (or polynomial-time hierarchy) can be defined by a hierarchy of problems that have oracle access to the lower level problems.

Definition 1 (Oracle definition of PH)**.**

Define classes

- $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$.
- For $i \geq 1$, $\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$.

Symmetrically, define classes

- $\Pi_1^{\mathsf{P}} = \text{co-}\mathsf{NP}$.
- For $i \geq 1$, $\Pi_{i+1}^{\mathsf{P}} = \text{co-}\mathsf{NP}^{\Pi_i^{\mathsf{P}}}$.

The Polynomial Hierarchy (PH) is defined as $\mathsf{PH} = \bigcup_i \Sigma_i^{\mathsf{P}} = \bigcup_i \Pi_i^{\mathsf{P}}$.

1.1 If $\mathsf{P} = \mathsf{NP}$, then PH collapses to P

Theorem 2. If $\mathsf{P} = \mathsf{NP}$, then $\mathsf{P} = \Sigma_i^{\mathsf{P}} \forall i$.

Proof. By induction on i.

Base case: For $i = 1$, $\Sigma_i^{\mathsf{P}} = \mathsf{NP} = \mathsf{P}$ by definition.

Inductive step: Assume $\mathsf{P} = \mathsf{NP}$ and $\Sigma_i^{\mathsf{P}} = \mathsf{P}$. Then $\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}} = \mathsf{NP}^{\mathsf{P}} = \mathsf{P}$. □

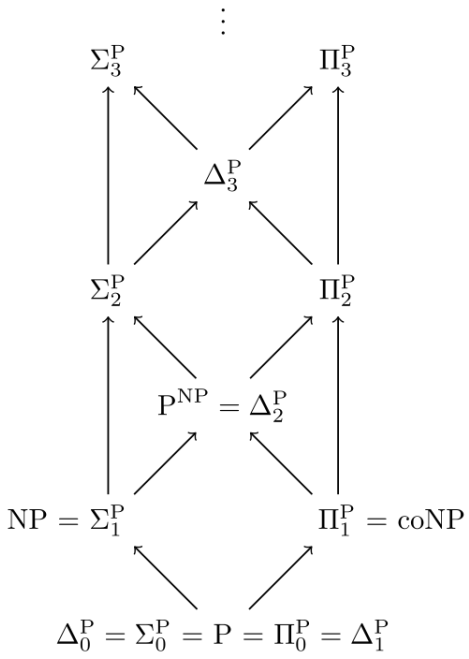

Figure 1: An overview of classes in PH. The arrows denote inclusion. Here $\Delta_{i+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$. (Image source: Wikipedia.)

Similarly, if any two different levels in PH turn out to be the equal, then PH collapses to the lower of the two levels.

Theorem 3. If $\Sigma_i^{\mathsf{P}} = \Sigma_{i+1}^{\mathsf{P}}$, then $\Sigma_j^{\mathsf{P}} = \Sigma_i^{\mathsf{P}}$ for all $j \geq i$.

1.2 Quantifier definition of PH

Theorem 4 (Quantifier definition of PH)**.**

- A language $L \in \Sigma_i^{\mathsf{P}}$ iff there is some relation $R(x, y_1, \ldots, y_i) \in \mathsf{P}$, polymomial q so that

$$x \in L \iff \exists y_1, \forall y_2 \ldots Q_i y_i \, R(x, y_1, \ldots, y_i)$$

  For all $j \in \{1, \ldots, i\}$, $|y_j| \leq q(|x|)$. If i is odd, then $Q_i = \exists$. Otherwise $Q_i = \forall$.
- A language $L \in \Pi_i^{\mathsf{P}}$ iff there is some relation $R(x, y_1, \ldots, y_i) \in \mathsf{P}$, polymomial q so that

$$x \in L \iff \forall y_1, \exists y_2 \ldots Q_i y_i \, R(x, y_1, \ldots, y_i)$$

  For all $j \in \{1, \ldots, i\}$, $|y_j| \leq q(|x|)$. If i is even, then $Q_i = \exists$. Otherwise $Q_i = \forall$.

Example 1.1. The decision problem "Is C a minimal circuit?" is in Π_2^{P}. Because to decide if C is a minimal circuit, we can decide whether

$$\forall C', |C'| \leq |C|, \exists x (C(x) \neq C'(x))$$

is true.

Next we prove the oracle definition and the quantifier definition are equivalent.

Quantifier definition $\implies$ oracle definition

Claim 1. If L can be written with i quantifiers beginning with $\exists$, then $L \in \Sigma_i^{\mathsf{P}}$.

Proof. By induction.

Base case: for $i = 1$, if $x \in L \iff \exists y_1 R(x, y_1)$, then $L \in \mathsf{NP} = \Sigma_1^{\mathsf{P}}$.

Inductive step: Assume any i-quantifier language $\in \Sigma_i^{\mathsf{P}}$.

Assume $x \in L$. $\exists y_1(\forall y_2 \exists y_3 \dots)\, R(x_1, y_1, \dots y_{i+1})$.

Define language L' such that $(x, y_1) \in L' \iff \forall y_2 \exists y_3 \dots R(x, y_1, \dots, y_{i+1})$.

Let $\overline{L'}$ be the complement of L'. Thus $(x, y_1) \in \overline{L'} \iff \exists y_2 \forall y_3 \dots R(x, y_1, \dots, y_{i+1})$.

$x \in L \iff \exists y_1 (x, y_1) \notin \overline{L'}$.

For a witness: y_1, we can ask the oracle for $\overline{L'}$: "Is $(x, y_1) \in \overline{L'}$?" If the oracle says no, then accept. If yes, then reject.

By induction hypothesis, the language $\overline{L'}$ is in $\in \Sigma_i^{\mathsf{P}}$. Thus, $x \in L$ iff there is a witness y_1 verifiable in Σ_i^{P}. Thus $L \in \mathsf{NP}^{\overline{L'}} \subseteq \Sigma_{i+1}^{\mathsf{P}}$.

□

Claim 2. If L can be written with i quantifiers beginning with $\forall$, then $L \in \Pi_i^{\mathsf{P}}$.

It can be proved symmetrically.

Oracle definition $\implies$ quantifier definition

The proof is based on [4].

Proof. By induction.

Base case: for $i = 1$, we already proved $L \in \Sigma_1^{\mathsf{P}} = \mathsf{NP}$ iff $x \in L \iff \exists y_1 R(x, y_1)$, where $|y_1| = \mathsf{poly}(|x|)$ and $R \in \mathsf{P}$.

Inductive step: Assume every $L' \in \Sigma_i^{\mathsf{P}}$ can be written with i-quantifier. Let M_L be the oracle machine for L.

$$M_{L'}(x) = 1 \iff \exists y_1, \forall y_2 \dots Q_i y_i \; R'(x, y_1, \dots, y_i) = 0.$$

Negating both sides,

$$M_{L'}(x) = 0 \iff \forall y_1, \exists y_2 \dots Q_i y_i \; R'(x, y_1, \dots, y_i) = 0.$$

Let $L \in \Sigma_{i+1}^{\mathsf{P}}$. Assume M is a TM in NP that makes m queries to oracle machine M' which is in Σ_i^{P}. Let y be the witness string of M, and let $(q_1, a_1), \dots (q_m, a_m)$ be the m queries and their

answers. Then, an input x in L iff

$$M(x)=1 \iff \exists(q_1,\ldots,q_m,a_1,\ldots,a_m,y) \\ [(M'(q_1)=a_1)\wedge\cdots\wedge(M'(q_m)=a_m)\wedge R(x,a_1,\ldots,a_m,y)]. \quad (1)$$

If $a_j=1$, then $M'(q_j)=1$, again by induction assumption, M' is in Σ_i^{P}, so

$$\exists y_{j,1}\forall y_{j,2}\ldots Q_i y_{j,i}\, R'(q_j,y_1,y_2,\ldots,y_i)=1.$$

If $a_j=0$, then $M'(q_j)=0$, by induction assumption, (note that this time we index the i quantifiers from 2 to $i+1$)

$$\forall y'_{j,2}\exists y'_{j,3}\ldots Q'_{i+1}y'_{j,i+1}\, R'(q_j,y'_2,y'_3,\ldots,y'_{i+1})=0.$$

By merging $\forall y_{j,2}$ with $\forall y'_{j,2}$, $\exists y_{j,3}$ with $\exists y'_{j,3}$, ... up to $Q_i y_{j,i}$ with $Q'_i y'_{j,i}$, the two formulas above become

$$\exists y_{j,1}\,\forall(y_{j,2},y'_{j,2})\exists(y_{j,3},y'_{j,3})\ldots Q_i(y_{j,i},y'_{j,i})Q_{i+1}y'_{i+1} \\ [(a_j=1\wedge R'(q_j,y_1,y_2,\ldots,y_i)=1)\vee(a_j=0\wedge R'(q_j,y'_2,y'_3,\ldots,y'_{i+1})=0)]. \quad (2)$$

Combining formulas (1) and (2) and taking each $\forall$ and $\exists$ quantifier over all $j\in\{1,\ldots,m\}$, we get

$$\begin{aligned} M(x)=1 \iff & \exists(q_1,\ldots,q_m,a_1,\ldots,a_m,y,y_{1,1},\ldots,y_{m,1}) \\ & \forall(y_{1,2},\ldots,y_{m,2},y'_{1,2},\ldots,y'_{m,2}) \\ & \exists(y_{1,3},\ldots,y_{m,3},y'_{1,3},\ldots,y'_{m,3}) \\ & \ldots \\ & Q_i(y_{1,i},\ldots,y_{m,i},y'_{1,i},\ldots,y'_{m,i}) \\ & Q_{i+1}(y'_{1,i+1},\ldots,y_{m,i+1}) \\ & \bigwedge_{j=1}^{m}[(a_j=1\wedge R'(q_j,y_1,y_2,\ldots,y_i)=1)\vee(a_j=0\wedge R'(q_j,y'_2,y'_3,\ldots,y'_{i+1})=0)] \\ & \wedge R(x,a_1,\ldots,a_m,y). \end{aligned}$$

□

2 More consequences of $\mathsf{P}=\mathsf{NP}$

2.1 Learning

See notes from previous classes.

2.2 Approximate counting

The class #P is analogous to NP. Instead of asking "are there any witnesses (or accepting paths of NTM)?", #P problems ask "how many witnesses (or accepting paths)?".

Definition 5 (#P)**.** A language is in class #P if its objective is to compute $f(x)$, where f is the number of polynomial length witnesses y satisfying $R(x, y)$, and R is verifiable in polynomial time.

NP $\subseteq$ #P in a straightforward way.

Toda's theorem shows that any problem in PH has a polynomial-time Turing reduction to a counting problem. Thus #P is more powerful then NP.

Theorem 6 (Toda's Theorem)**.** $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Corollary 7. If $\#\mathsf{P} = \Sigma_i^{\mathsf{P}}$, then $\mathsf{PH} = \Sigma_{i+1}^{\mathsf{P}}$.

Stockmeyer's Theorem: for any function $F \in \#\mathsf{P}$, we can approximate F in $\mathsf{P}^{\Sigma_2^{\mathsf{P}}}$. (i.e. approximate algorithm A satisfies $A(x)/(1+\epsilon) \leq F(x) \leq A(x)(1+\epsilon)$). Thus if $\mathsf{P} = \mathsf{NP}$, we can do approximate counting in polynomial time.

References

[1] Polynomial hierarchy - wikipedia. `https://en.wikipedia.org/wiki/Polynomial_hierarchy`.

[2] Sharp-p - wikipedia. `https://en.wikipedia.org/wiki/Sharp-P`.

[3] Toda's theorem - wikipedia. `https://en.wikipedia.org/wiki/Toda%27s_theorem`.

[4] Jonathan Katz. Notes on complexity theory. `http://www.cs.umd.edu/~jkatz/complexity/f11/lecture9.pdf`.