80240233: Computational Complexity | **Lecture 3**
ITCS, Tsinghua Univesity, Fall 2007 | 16 October 2007

Instructor: Andrej Bogdanov | **Notes by:** Jialin Zhang and Pinyan Lu

In this lecture, we introduce the complexity class coNP, the *Polynomial Hierarchy* and the notion of *oracle*.

1 The class coNP

The complexity class NP contains decision problems asking this kind of question:

$$\text{On input } x, \quad \exists y : |y| = p(|x|) \text{ s.t.} V(x,y),$$

where p is a polynomial and V is a relation which can be computed by a polynomial time Turing Machine (TM).

Some examples:

SAT: given boolean formula ϕ, does ϕ have a satisfiable assignment?

MATCHING: given a graph G, does it have a perfect matching?

Now, consider the opposite problems of SAT and MATCHING.

UNSAT: given boolean formula ϕ, does ϕ have no satisfiable assignment?

UNMATCHING: given a graph G, does it have no perfect matching?

This kind of problems are called coNP problems. Formally we have the following definition.

Definition 1. *For every $L \subseteq \{0,1\}^*$, we say that $L \in \text{coNP}$ if and only if $\overline{L} \in \text{NP}$, i.e. $L \in \text{coNP}$ iff there exist a polynomial-time TM A and a polynomial p, such that*

$$x \in L \;\Leftrightarrow\; \forall y, |y| = p(|x|)\ A(x,y) \textit{ accepts.}$$

Then the natural question is how P, NP and coNP relate. First we have the following trivial relationships.

Theorem 2. $\text{P} \subseteq \text{NP}$, $\text{P} \subseteq \text{coNP}$

For instance, MATCHING $\in$ coNP. And we already know that SAT $\in$ NP. Is SAT $\in$ coNP? If it is true, then we will have NP = coNP. This is the following theorem.

Theorem 3. *If* SAT $\in$ coNP*, then* NP = coNP.

Proof. Take any $L \in \text{NP}$, we can reduce L to SAT. Since SAT $\in$ coNP, so there exists an coNP algorithm for SAT. Therefore, there exists an coNP algorithm for L. So $L \in \text{coNP}$. So we proved that NP $\subseteq$ coNP.

For the other side, we have

$$L \in \text{coNP} \Rightarrow \overline{L} \in \text{NP} \Rightarrow \overline{L} \in \text{coNP} \Rightarrow L \in \text{NP}.$$

The first and last steps use the definition of coNP and the middle step use the result NP $\subseteq$ coNP, which is proved above.

To sum up, we complete the proof. □

2 Polynomial Hierarchy

Here we consider a new problem MIN-EQUIV. Given a boolean formula ϕ, is ϕ the smallest formula that computes the function ϕ? Formally,

$$\phi \in \text{MIN-EQUIV} \Leftrightarrow \forall \phi' < \phi,\ \exists x : \phi'(x) \neq \phi(x).$$

The opposite problem of MIN-EQUIV is $\overline{\text{MIN-EQUIV}}$. Given a boolean formula ϕ, is there a smaller formula that compute the function ϕ?

There is no obvious notion of a certificate of membership. It seems that the way to capture such languages is to allow not only an "exists" quantifier (as in the definition of NP) or only a "for all" quantifier (as in the definition of coNP). This motivates the following definition:

Definition 4. Σ_2 *is defined to be the class of decision problems for which there exists a polynomial-time TM A and a polynomial p such that* $x \in L \Leftrightarrow \exists y_1 \forall y_2\ A(x, y_1, y_2)$ *accepts, where* $|y_1| = p(|x|), |y_2| = p(|x|)$.

Π_2 *is defined to be the class of decision problems for which there exists a polynomial-time TM A and a polynomial p such that* $x \in L \Leftrightarrow \forall y_1 \exists y_2\ A(x, y_1, y_2)$ *accepts, where* $|y_1| = p(|x|), |y_2| = p(|x|)$.

The polynomial hierarchy generalizes the definitions of NP, coNP, Σ_2, Π_2.

Definition 5. Σ_k *is defined to be the class of decision problems for which there exists a Polynomial-time TM A and a polynomial p such that* $x \in L \Leftrightarrow \exists y_1 \forall y_2 \cdots \exists/\forall y_k\ A(x, y_1, y_2, \cdots, y_k)$ *accepts, where* $|y_i| = p(|x|), i = 1, \cdots k$.

Π_k *is defined to be the class of decision problems for which there exists a Polynomial-time TM A and a polynomial p such that* $x \in L \Leftrightarrow \forall y_1 \exists y_2 \cdots \exists/\forall y_k\ A(x, y_1, y_2, \cdots, y_k)$ *accepts, where* $|y_i| = p(|x|), i = 1, \cdots k$.

The polynomial hierarchy is the class $\text{PH} = \cup_i \Sigma_i$.

There are some basic observations about polynomial hierarchy:

1. $P = \Sigma_0 = \Pi_0$;
2. $\text{NP} = \Sigma_1$, $\text{coNP} = \Pi_1$;
3. $P \subseteq \text{NP} \cap \text{coNP} \subseteq \Sigma_2, \Pi_2 \subseteq \Sigma_3, \Pi_3 \subseteq \cdots$;
4. $L \in \text{NP} \Leftrightarrow \overline{L} \in \text{coNP}$;
5. $L \in \Sigma_k \Leftrightarrow \overline{L} \in \Pi_k$.
6. $\forall k, \Sigma_k, \Pi_k \subseteq EXP$.

We believe – but don't know how to prove – that $\Sigma_k \neq \Sigma_{k+1}$, $\Pi_k \neq \Pi_{k+1}$, $\Sigma_k \neq \Pi_k$ and $\text{EXP} \neq \Sigma_k, \Pi_k$ for all k.

There is a survey by Schaeffer and Umans[1, 2], which gives several nature complete problems for $\Sigma_2, \Sigma_3, \Pi_2, \Pi_3$.

3 Oracle

Oracle is equivalent of subroutine in complexity theory. An oracle Turing Machine can be executed with access to a special tape, where they can make queries of the form "is $q \in L$" for some language L and get the answer in one step. That is, oracle gives us functionality what we do not know how to implement efficiently.

Definition 6. *P^A is defined to be all decision problems decided by Polynomial-time oracle TM given access to oracle A.*

Definition 7. NP^A *is defined bo be all decision problems decided by Nondeterministic Polynomial-time oracle TM given access to oracle A.*

We already saw a special kind of oracle computation, namely a reduction. In a reduction the oracle is asked only one question, and the answer to this question is the output of the algorithm. In particular, if decision problem A reduces to B, then $A \in \text{P}^B$.

Let's play with oracles for a bit to get a feel about what they do:

1. What is $\text{P}^{\text{MATCHING}}$? This is just P, since any call to the oracle can be simulated by the polynomial-time machine making the call. For the same reason $\text{NP}^{\text{MATCHING}} = \text{NP}$.
2. How about P^{SAT}? A polynomial-time machine with a SAT oracle can solve any NP question by Cook's theorem – first reduce to SAT then ask the question. But it can also solve any coNP question – again, reduce to SAT, ask the question, then output the opposite answer. So we have $\text{P}^{\text{SAT}} \supseteq \text{NP}, \text{coNP}$.
3. Since P^{SAT} is more powerful than both NP and coNP, how does it relate to Σ_2 and Π_2? The following theorem, implies that $\text{P}^{\text{SAT}} \subseteq \Sigma_2 \cap \Pi_2$.

Theorem 8. $\mathrm{NP}^{\mathrm{SAT}} = \Sigma_2$

Proof. Step 1: prove $\Sigma_2 \subseteq \mathrm{NP}^{\mathrm{SAT}}$. For any $L \in \Sigma_2$, there exists a polynomial time TM V such that $x \in L \Leftrightarrow \exists y \forall z\ V(x, y, z)$ accepts.

We can think "$\exists y$" part to be the nondeterministic tape of a NTM N. Once N guesses y, it has to determine whether $\forall z : V(x, y, z)$ accepts. We can ask SAToracle here, "does there exists $z, s.t. V(x, y, z)$ rejects?" and output the opposite answer.

Step 2: prove $\mathrm{NP}^{\mathrm{SAT}} \subseteq \Sigma_2$. For any $L \in \mathrm{NP}^{\mathrm{SAT}}$, we are given an oracle Polynomial-time NTM N. We need to simulate N^{SAT} by $\exists y, \forall z, V(x, y, z)$ accepts.

Nondeterministic tape of N is part of y in "$\exists y$" of V. When N makes an oracle call Φ_i, V keeps trace of Φ_i, and guesses an answer a_i to Φ_i . In the end, V will check "Yes" answers($\phi_i \in$ SAT) and "No" answers ($\phi_i \in$ UNSAT).

To sum up, we can define the Σ_2 language as following:

$$\exists y \exists a_1, a_2, \cdots, a_k \exists v_1, v_2, \cdots v_k \forall w_1, w_2, \cdots, w_k\ V(x, y, a, v, w) \text{accepts},$$

where $V(x, y, a, v, w)$ accepts if and only if the following two conditions satisfy: (1) given input x, nondeterministic tape y and the oracle's answer a, $N(x, y, a)$ accepts; (2) a_i is a correct answer of Φ_i, which means that for every $i, 1 \leq i \leq k$, either $a_i =$ "Yes" and $\Phi_i(v_i) =$ "Yes" or $a_i =$ "No" and $\Phi_i(w_i) =$ "No". □

This argument gives an alternate characterization of the polynomial hierarchy using oracle machines, and it can be extended to higher levels of the hierarchy too:

1. $\mathrm{coNP}^{\mathrm{SAT}} = \Pi_2$;
2. $\Sigma_{k+1} = \mathrm{NP}^{\Sigma_k-\text{complete problem}}$.

Since "Σ_kSAT" is complete for class Σ_k, this means $\mathrm{NP}^{\Sigma_k \mathrm{SAT}} = \Sigma_{k+1}$.

The following table summarizes some conjectures people believe and the fact people prove about polynomial hierarchy.

	P, NP, coNP	PH
Believe	$\mathrm{P} \neq \mathrm{NP}$	$\Sigma_k \neq \Sigma_{k+1}$ for all k
	$\mathrm{NP} \neq \mathrm{coNP}$	$\Sigma_k \neq \Pi_k$ for all k
Fact	$\mathrm{P} = \mathrm{NP} \Leftrightarrow \mathrm{P} = \mathrm{coNP}$	$\Sigma_k = \Sigma_{k+1} \Rightarrow \Pi_{k+1} = \Sigma_{k+1} = \Pi_k = \Sigma_k$

Theorem 9. $\mathrm{P} = \mathrm{NP} \Rightarrow \Sigma_2 = \mathrm{P}$

Proof. Take any $L \in \Sigma_2$. This means $x \in L \Leftrightarrow \exists y \forall z, V(x, y, z)$ Accepts. We define another language $L' : (x, y) \in L' \Rightarrow \forall z', V(x, y, z')$ accepts. So $L' \in$ coNP. By the assumption P = NP, we have $L' \in$ P. This means there exists Polynomial-time TM V such that $(x, y) \in L' \Leftrightarrow V'(x, y)$ accepts. Then, $x \in L \Rightarrow \exists y \; V'(x, y)$ accepts. This means $L \in$ NP = P. $\square$

References

[1] M. Schaeffer and C. Umans. Completeness in the Polynomial-Time Hierarchy: a compendium. SIGACT News. guest Complexity Theory column. September 2002.

[2] M. Schaeffer and C. Umans. Completeness in the Polynomial-Time Hierarchy: Part II. SIGACT News. guest Complexity Theory column. December 2002.