IITM-CS6840: Advanced Complexity Theory January 28, 2012

Lecture No. 13 : Self Reducibility of SAT, Complete problem for Σ_k^{P}

Lecturer: Jayalal Sarma M.N. *Scribe: Sajin Koroth*

THEME: Polynomial Hierarchy and its relation to $\mathsf{P/poly}$
LECTURE PLAN:Recall that we introduced the advice based class $\mathsf{P/poly}$ in the last lecture. We also saw that $\mathsf{BPP} \subsetneq \mathsf{P/poly}$, and by definition $\mathsf{P} \subsetneq \mathsf{P/poly}$. But we don't know whether $\mathsf{NP} \subsetneq \mathsf{P/poly}$ or not. Hence if we could prove that $\mathsf{NP} \not\subset \mathsf{P/poly}$ then we would essentially be separating P from NP. The reason why most of the complexity theorists believe $\mathsf{NP} \not\subset \mathsf{P/poly}$ is, if $\mathsf{NP} \subset \mathsf{P/poly}$ then we would be able to prove that $\mathsf{PH} = \Sigma_2^{\mathsf{P}}$, contrary to the common belief that PH does not collapse. In today's lecture we will detail two key ingredients needed for showing the above mentioned conditional collapse of PH, **a complete problem for** Σ_k^{P} and **self reducibility property of** SAT

1 Complete problem for the hierarchy

We will first show a complete problem for the kth level of polynomial hierarchy. We will do this by generalizing SAT to SAT with quantified expressions. Later on we will use the self-reducibility nature of this problem to show the conditional collapse mentioned in the beginning. We know that a language $L \in \mathsf{NP}$ if and only if there is a deterministic polynomial time Turing machine M, such that $x \in L \iff \exists y \in \{0,1\}^{p(n)}, M(x,y) = 1$. Cook-Levin theorem guarantees that this machine M can be converted into a formula $\phi(x,y)$ on variables x_i, y_i such that ϕ correctly simulates M on the non-deterministic branch y on input x by $\phi(x,y) = 1$ if and only if $M(x,y)$ accepts. Hence all of NP can be captured by $\exists - \mathsf{SAT}$ which is defined as $L \in \exists - \mathsf{SAT}$ if there exists a polynomial p and a formula ϕ such that $x \in L$ if and only if the quantified expression $\exists y \in \{0,1\}^{p(n)} \phi(x,y)$ is true.

Note that the above definition can be generalized to arbitrary number of quantifiers. Recall that a language L is said to be in Σ_k^{P} if there exists polynomials $p_1, \ldots, p_k$ and a machine M running in deterministic polynomial time such that

$$x \in L \iff \exists y_1 \forall y_2 \exists y_3 \ldots Q_k y_k \left[M(x, y_1, y_2, y_3, \ldots, y_k) = 1\right], \forall i, |y_i| \leq p_i(|x|)$$

Cook-Levin theorem guarantees that machine M on input x can be converted into formula ϕ_x in polynomial time on variables $y_1, y_2, \ldots, y_k$ such that $\phi_x(y_1, y_2, y_3, \ldots, y_k)$ is satisfiable if and only if $M(x, y_1, y_2, y_3, \ldots, y_k)$ accepts. Hence we can say that the following problem is complete for Σ_k^{P},

Definition 1 ($\Sigma_k - \mathsf{SAT}$)**.** $\Sigma_k - \mathsf{SAT}$ is the set of all quantified Boolean formulas with at most k alternations (starting with an existential quantifier) which are true. That is

$$\Sigma_k - \mathsf{SAT} = \{\exists y_1 \forall y_2 \exists y_3 \dots Q_k y_k \phi(y_1, \dots, y_k) \mid \exists y_1 \forall y_2 \exists y_3 \dots Q_k y_k \phi(y_1, \dots, y_k) \text{ is true}\}$$

The above problem is clearly in Σ_k^{P} as you can in polynomial time construct from a formula, a machine in P for checking if the formula is satisfiable or not given an assignment of all the variables as input. The problem is Σ_k^{P} hard because of Cook-Levin reduction from any machine in P to an equivalent formula.

2 Self reducibility of SAT

Suppose we are given that $\mathsf{NP} \subset \mathsf{P}/\operatorname{poly}$ then we know that there is a polynomial time deterministic Turing machine and a polynomial length advice string for each input length such that the machine decides a given language in NP. We will sketch how this can cause a collapse in the Polynomial Hierarchy, without giving the details but exposing some difficulties which we have to overcome before getting to the proof.

To prove that PH collapses to Σ_2^{P} it suffices to show that $\Sigma_3^{\mathsf{P}} = \Sigma_2^{\mathsf{P}}$. Recall that Σ_3^{P} is the set of true quantified Boolean formulas which are of the form $\exists y_1 \forall y_2 \exists y_3 M(x, y_1, y_2, y_3)$, and Σ_2^{P} are true quantified Boolean formulas which are of the form $\exists y_1 \forall y_2 M(x, y_1, y_2)$. Also we are given that $\mathsf{NP} \subset \mathsf{P}/\operatorname{poly}$ hence for any $L \in \mathsf{NP}$ there exists $h : N \to \{0,1\}^*$ and an $M \in \mathsf{P}$ such that $x \in L$ if and only if $(x, h(|x|))$ is accepted by M. The idea to place Σ_3^{P} in Σ_2^{P} is the following, the third there exists y_3 and $M(x, y_1, y_2, y_3)$ can be combined to a machine in NP, where it first guesses a string y_3 of size $p_3(|y_3|)$ and then runs M on (x, y_1, y_2, y_3). We have assumed that equivalent to this NP machine there is a $\mathsf{P}/\operatorname{poly}$ machine, and even though we don't know the advice string we know there exists a good advice string, and given the advice string the last "there exists" quantifier in Σ_3^{P} can be eliminated by replacing it with the polynomial time machine which is given the advice string, hence we would a get a language in Σ_2^{P}. But unfortunately we don't know the advice string, hence the next best thing to do is to guess the advice string using the first "there exists" quantifier in Σ_2^{P}.

Even though we are guaranteed that at least one guess is the correct advice string, there is a catch here, we could have guessed the advice string incorrectly in some branch which in turn could have led the machine M to accept incorrectly thus falsely accepting a string outside the language L in Σ_3^{P}. To get around this problem we will use the first part to reduce the problem in Σ_3^{P} to $\Sigma_3 - \mathsf{SAT}$ and then use an algorithm for SAT which given a sub-routine which tells a formula is satisfiable or not, which uses a crucial property of the SAT problem, self-reducibility to construct a satisfying assignment for the given formula. And irrespective of whether the advice string is good or bad, we would not be able to construct a satisfying assignment for an unsatisfiable formula. This guarantees the correctness of the computation captured by the SAT formula.

Self reducibility of SAT refers to the property of the SAT problem that checking a formula on n variables is satisfiable reduces to checking the satisfiability of two formulas on $n-1$ variables. This property leads to a polynomial time algorithm for constructing a satisfying assignment given a polynomial time sub-routine deciding the decision version of SAT problem correctly. Algorithm 2 constructs a satisfying assignment given a sub-routine which correctly solves SAT instances of up to n variables. Notice that one important property of Algorithm 2 is that even if the sub-routine which checks the satisfiability of a formula is wrong, the algorithm would not be accepting an un-satisfiable formula as a satisfiable formula. Because at the end of the algorithm we are checking whether the assignment constructed by the algorithm is satisfiable or not, so even if the sub-routine for SAT, **SATISFIABLE** is erroneous we would not be able to construct a satisfying assignment for an un-satisfiable formula. But it might fail to construct a satisfying assignment for a satisfiable formula if the sub-routine is erroneous.

SATISFYING-ASSIGNMENT($\phi(x_1, \ldots, x_n)$)

```
 1  ψ(x_1, ..., x_n) ← φ(x_1, ..., x_n)
 2  for i ← 1 to n
 3        do
 4           ψ' ← ψ(x_i = 0)
 5           ψ'' ← ψ(x_i = 1)
 6           if (SATISFIABLE(ψ'))
 7              then a_i ← 0
 8                   ψ ← ψ(0, x_2, ..., x_n)
 9           elseif (SATISFIABLE(ψ''))
10              then a_i ← 1
11                   ψ ← ψ(1, x_2, ..., x_n)
12           else
13                   return IMPOSSIBLE
14           if φ(a_1, ..., a_n) = 1
15              then return (a_1, ..., a_n)
16              else  return IMPOSSIBLE
```