CS 6810: Theory of Computing **Fall 2023**

Lecture 4: Aug 31, 2023

Lecturer: Eshan Chattopadhyay *Scribe: Tenghao Li*

In this lecture, we proved the Ladner's Theorem. The only assumption we used in this theorem and its proof is that $\mathbf{NP} \neq \mathbf{P}$. We finished the proof by explicitly constructing a language SAT_H.

1 Ladner's Theorem

Theorem 1.1 (Ladner's Theorem)**.** *Given that $\boldsymbol{NP} \neq \boldsymbol{P}$, there exists a language $L \in \boldsymbol{NP} \setminus \boldsymbol{P}$ such that L is not $\boldsymbol{NP}$-Complete.*

To prove the theorem, we aim to explicitly construct the language L by "padding" the SAT language. Recall that SAT (Boolean Satisfiability Problem) is the problem of determining if there exists a True/False assignment that satisfies the given boolean fomula.

Example 1.2.

$$SAT' = \{\phi \circ 1^{2^n} : |\phi| = n, \phi \in SAT\}$$

$\text{SAT}' \in \mathbf{P}$ because the TM can brute force all possible boolean assignments in $O(n2^n)$ time. Since the size of the input is $N = 2^n$, the running time is polynomial with repect to the input size N, namely $O(N \log N)$. Intuitively, SAT' is an example of padding too much.

Example 1.3.

$$SAT'' = \{\phi \circ 1^{n^c} : |\phi| = n, \phi \in SAT\}$$

where c is a constant.

SAT'' is **NP**-complete, because $\text{SAT} \leq_p \text{SAT}''$ by simply mapping ϕ to $\phi \circ 1^{n^c}$. Since c is a constant, it is a poly-time reduction. Intuitively, SAT'' is an example of padding too little.To find the right number of padded 1s, we define the following language SAT_H and function $H(n)$.

Definition 1.4.

$$SAT_H = \{\phi \circ 1^{H(n)} : |\phi| = n, \phi \in SAT\}$$

Definition 1.5. *$H(n)$ is the smallest natural number $i \leq \log\log n$ such that the TM M_i correctly solves $x \in SAT_H$ within $i|x|^i$ time, for all input $x \in \{0,1\}^*$ with $|x| \leq \log n$. If no such i exists, then $H(n) := \log\log n$.*

The definition of H is self-referencing H itself. However, this definition is not cyclic. To compute $H(n)$, we only consider the input x with size no greater than $\log n$. Therefore, $H(n)$ is recursively defined by values of function H on smaller integers. To intuitively understand these definitions, consider a list of Turing Machines $M_1, M_2, ..., M_i, ..., M_{\log\log n}$. M_1 allows linear running time, M_2 allows quadratic running time, and so on. We run these TMs on the input set $S = \{x \in \{0,1\}^* : |x| \leq \log n\}$ until we find the smallest integer i such that M_i can correctly determine if $x \in \text{SAT}_H$ for all $x \in S$ in polytime.

Lemma 1.6. *If $SAT_H \in \boldsymbol{P}$ then $H(n) \leq c$ for all $n \in \mathbb{N}$ where c is a constant.*

Proof. Since $\text{SAT}_H \in \mathbf{P}$, there exists a polytime TM M that solves SAT_H. Denote $\beta = \lfloor M \rfloor$. Since we can always padding the TM (e.g adding some useless states), assume β is large enough such that the running time of TM is bounded by βn^β.
If $n \leq 2^{2^\beta}$, then by definition $H(n)$ is no greater than $\log\log n$.

$$H(n) \leq \log\log n \leq \beta$$

If $n > 2^{2^\beta}$, then obeserve that $M = M_\beta$. Since M_β always solves SAT_H within βn^β steps, the smallest idex of TM is no greater than β. By definition of $H(n)$, we know that $H(n) \leq \beta$. □

Lemma 1.7. *If $H(n) = \beta$ for infinitely many n's, then $SAT_H \in \boldsymbol{P}$.*

Proof. We aim to show that M_β solves SAT_H in βn^β steps. For the sake of contradiction, suppose there exists an input x with $|x| = n$ such that M_β makes an mistake on x or does not halt in βn^β steps. Then for all $n > 2^n$, $H(n) \neq \beta$, which contradicts with the assumption that $H(n) = \beta$ for infinitely many n's. □

Corollary 1.8. *If $SAT_H \notin \boldsymbol{P}$, then $\lim_{n\to\infty} H(n) = \infty$.*

Proof. This is easy to derive from Lemma 1.7. □

Now we are ready to prove the Ladner's Theorem.

Proof of Ladner's Theorem. We aim to show that $\text{SAT}_H \in \mathbf{NP} \setminus \mathbf{P}$ and SAT_H is **NP**-complete.

First, SAT_H is clearly in **NP**. The certificate is a satisfying boolean assignment. The verifier will first evaluate the formula on the given assignment, and then verify if the input contains the right number of 1's. The verifier runs in polynomial time.

Second, we claim that $\text{SAT}_H \notin \mathbf{P}$. For the sake of contradiction, assume $\text{SAT}_H \in \mathbf{P}$. Then by Lemma 1.6, $H(n) \leq c$ for all $n \in \mathbb{N}$ for some constant c. Therefore, this is exactly the same situation of Example 1.3. We can constuct a polynomial reduction by mapping ϕ to $\phi \circ 1^{n^{H(n)}}$. Thus,

$$\text{SAT} \leq_p \text{SAT}_H$$

Hence, $\text{SAT} \in \mathbf{P}$ which is a contradiction to our only assumption that $\mathbf{NP} \neq \mathbf{P}$.

Third, we now show that SAT_H is not **NP**-complete. (More details on next lecture)

□