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

Lecture 5: September 5, 2023

Lecturer: Eshan Chattopadhyay *Scribe: Evan Williams*

1 Ladner's Theorem, Continued

We now show that SAT_H is not NP-complete. Suppose for sake of contradiction that SAT_H is NP-complete. Then, there must exist a polynomial time reduction from SAT to SAT_H by Lemma 1.6 from the previous lecture. This reduction maps a boolean formula ψ of size n to an instance of SAT_H of length n^c for some constant c. This instance would be of the form $\psi \circ 01^{k^{H(k)}}$ where $k = |\psi|$ and $n^c = k + k^{H(k)}$. Observe that because $SAT_H \notin P$, $\lim_{n\to\infty} H(n) = \infty$ (by Corollary 1.8 from the previous lecture). So $\lim_{n\to\infty} |\psi|/n = 0$. Observe that we have effectively reduced the SAT instance ψ to an instance ϕ where $\frac{|\phi|}{|\psi|} = o(|\psi|)$. We can apply this reduction repeatedly to obtain a SAT instance of constant size. **1)** Compute $H(k)$ for every $k \leq \log n$, **2)** simulate at most $\log\log n$ machines for every input of length at most $\log n \log\log n(\log n)^{\log\log n} = o(n)$ steps, and **3)** compute SAT on all inputs of length at most $\log n$. We effectively obtain a SAT instance of constant size, which implies that $SAT \in P$, contradicting the supposition that $P \neq NP$. ■

2 Oracle Turing Machines and Relativization

Definition 2.1. *An* **oracle** *is a language* $O \subseteq \{0,1\}^*$ *and a* **query** *is a string* $x \in \{0,1\}^*$

The oracle is able to answer queries about a particular function. In other words, given a function $f : \{0,1\}^* \to \{0,1\}^*$ and x, the oracle answers $f(x)$.

Definition 2.2. *Given an oracle O, an* **oracle Turing Machine** M^O *is a multi-tape Turing Machine with the following:*

- *An oracle tape*
- *Three additional states,* q_{query}, q_{yes}, q_{no}

The oracle Turing Machine is able to write a string x on its oracle tape and then transition into q_{query}. If the oracle says yes, then the machine transitions to state q_{yes}, otherwise, it transitions to state q_{no}. We define the complexity class $DTIME^O(T(n))$ as the set of languages the oracle Turing Machine M^O can compute in $O(T(n))$ time. Thus, we have analogous complexity classes for P and NP - P^O and NP^O.

Question 1. *Is Co-NP* $\subseteq P^{SAT}$*?*

Yes. Construct an oracle Turing Machine $M^{S\bar{A}T}$ that takes in a string x and writes it on its oracle tape to run a query. Then, it simply returns the opposite of whatever the oracle returns. So the oracle Turing Machine $M^{S\bar{A}T}$ will always correctly compute SAT with linear overhead.

Theorem 2.3. *There exist oracles A, B such that* $P^A = NP^A$*, but* $P^B \neq NP^B$

This idea was presented by Baker-Gill-Solovay. A proof **relativizes** if you enumerate over Turing Machines and use a Universal Turing Machine to simulate other Turing Machines. We observe that any diagonalization proof must relativize.

Now we prove the theorem. Let $A = \{< M, x, 1^n >$: M accepts x in2^nsteps$\}$. Recall that $EXP = \bigcup_{c \in \mathbb{N}} DTIME(2^{(}n^c))$. We claim that $P^A = NP^A = EXP$. Obviously $P^A \subseteq NP^A$, so it suffices to show that $EXP \subseteq P^A$ and $NP^A \subseteq EXP$.

The remainder of the proof will be detailed in the next lecture.