Lecture 5
Computational Complexity Theory

Abhishek Shetty
Undergraduate Department
Indian Institute of Science

Raghav Malhotra
Undergraduate Department
Indian Institute of Sciece

Instructor
Chandan Saha
Computer Science and Automation
Indian Institute of Science

August 20 2015

1 Ladner's Theorem

Definition 1.1. $\mathsf{NPC} = \{L \in \mathsf{NP} \mid \forall A \in \mathsf{NP} : A \leq_p L\}$

Definition 1.2. $\mathsf{NP} - Intermediate = \mathsf{NP} \setminus (\mathsf{P} \bigcup \mathsf{NPC})$

Definition 1.3. Let $B : \mathbb{N} \to \mathbb{N}$. $SAT_B = \{\psi 01^{n^{B(n)}} \in \{0,1\}^* \mid \psi \in SAT \text{ and } |\psi| = n\}$

Lemma 1.1. *$\exists H : \mathbb{N} \to \mathbb{N}$ such that*

- *$H(m)$ is computable in time $\mathsf{O}(m^3) : \forall m \in \mathbb{N}$*
- *$SAT_H \in \mathsf{P} \iff \exists c$ such that $\forall m \in \mathbb{N} : H(m) < c$*
- *$SAT_H \notin \mathsf{P} \implies H(m) \to \infty$ as $m \to \infty$*

Proof. We define $H(n)$ to the be the smallest number $k < \log\log n$ such that $\forall x \in \{0,1\}^*$ with $|x| < \log n$, M_k decides whether $x \in SAT_H$ within $k|x|^k$ steps. In the absence of such a k, we define $H(n) = \log\log(n)$. H is well-defined $H(n)$ depends on strings of atmost size $\log(n)$ for which SAT_H is well defined in terms of $H(i)$ where $i < n$. Notice that H is a monotically increasing function.

Assume that $SAT_H \in P$. There exists a machine M that decides whether $x \in SAT_H$ in $c|x|^c$ steps. We can find i such that $M_i = M$ and $i > c$. Now we look at n such that $\log\log n > i$. From the definition of H, we have $H(n) < i$. For $\log\log(n) < i$, $H(n) < i$ by definition. Therefore, we have $\forall n : H(n) < i$.

Assume that $\exists c : H(n) < c$. Since H is bounded, its range is finite and as its domain is infinite, by pigeonhole principle we have $\exists i : H(n) = i$ for infinitely many n. From the definition of H, we have M_i decides SAT_H in in^i steps, else if we have x such that M_i does not decide x in the bound then $\forall n : \log(n) > |x|$, $H(n) > i$, which is a contradiction.

$SAT_H \notin P \implies H$ is unbounded. As H is monotonically increasing and unbounded, we have $\lim_{n\to\infty} H(n) = \infty$. □

Remark. $\lim_{n\to\infty} f(n) = \infty \iff \forall M \in \mathbb{N}\ \exists N_0 \in \mathbb{N} : \forall n > N_0,\ f(n) > M$

Theorem 1.2 (Ladner)**.** $\mathsf{P} \neq \mathsf{NP} \implies \exists L \in \mathsf{NP} - Intermediate.$

Proof. It can be shown that if $\mathsf{P} \neq \mathsf{NP}$, SAT_H as defined above is in $\mathsf{NP} - Intermediate$. The proof of the above claim was completed in Lecture 4. □

2 Oracle Turing Machines

Diagonalization is an extremely powerful proof technique in complexity theory. For clarity, we look at the two characterizing qualities of a diagonalization argument,

a. Given a string, there is a Turing Machine corresponding to the string and given a Turing Machine, there exist infinitely many strings representing the Turing Machine.

b. There exists a Universal Turing Machine that can simulate any Turing Machine with only a small overhead.

To look at the limits of the diagonalization argument, we look at a construction of a new type of Turing Machine called the Oracle Turing Machine which satisfy the above properties. The Oracle Turing Machine is given access to an oracle to a language that can be decided within one computation step.

Definition 2.1. A Determinisitic Oracle Turing Machine M with access to an oracle for the language $O \subset \{0,1\}^*$ denoted as M^O is defined by the following properties

- The machine has three special states. q_{query}, q_{yes}, q_{no}, and a special read write tape called the oracle tape.
- Whenever the machine enters the state q_{query} , the query x written on the oracle tape is asked to the oracle which decides O and moves to q_{yes} if $x \in O$ and to q_{no} if $x \notin O$.
- Regardless of O, the membership query counts as a single step of computation for the oracle machine.

Definition 2.2. Let $O \subset \{0,1\}^*$. P^O is the set of all languages which can be decided by a poly-time TM with access to an oracle that decides O. NP^O is the set of all languages decided by similar non deterministic Turing machines.

Lemma 2.1. *Let $\overline{SAT}$ be the set of all unsatisfiable boolean formulae. Then $\overline{SAT} \in \mathsf{P}^{SAT}$.*

Proof. If the machine wants to check whether a formula $\phi \in \overline{SAT}$, it queries the oracle and outputs the opposite of the oracle's reply. As the only computational steps are writing down the query on the oracle tape, querying the oracle and output the answer, the machine runs in polynomial time. □

Corollary 2.1.1. $\mathsf{NP} \cup Co\text{–}\mathsf{NP} \subset \mathsf{P}^{SAT}$.

Lemma 2.2. $O \in \mathsf{P} \iff \mathsf{P}^O = \mathsf{P}$.

Proof. Clearly $\mathsf{P} \subset \mathsf{P}^O$. However if O is polynomial we can replace the oracle O with a poly-time computation of O. Thus, $\mathsf{P}^O \subset \mathsf{P} \implies \mathsf{P}^O = \mathsf{P}$. Conversely, assume $O \notin \mathsf{P}$. A machine with access to O can decide O in polynomial time. This implies $O \in \mathsf{P}^O$. Thus, $\mathsf{P} \neq \mathsf{P}^O$. □

Regardless of what the oracle O is, the set of all Turing machines with access to O satisfy properties required for diagonlization. This is because we can represent any Turing machine with an oracle as a string and therefore simulate them on an universal Turing machine. Thus, any result on complexity and Turing machines which only uses properties a and b mentioned above is true for Turing machines with access to an arbitrary oracle. Such results are called *relativizing* results.

Theorem 2.3 (Gill-Solovay-Baker)**.** *$\exists A, B \subset \{0,1\}^*$ such that $\mathsf{NP}^A = \mathsf{P}^A$ and $\mathsf{NP}^B \neq \mathsf{P}^B$.*

We prove the Theorem through a set of Lemmas.

Lemma 2.4. *Let $A \subset \{0,1\}^*$ be defined as*
$A = EXPCOM = \{(M, x, 1^n) : DTM\ M \text{ accepts string } x \text{ within } 2^n \text{ steps}\}$. Then $\mathsf{P}^A = \mathsf{NP}^A$.

Proof. From the definition of the language, it is clear that any language $L \in \mathsf{EXP}$ can be reduced to A in polynomial time through the reduction $x \in L \to (M, x, 1^{n^c})$ where n^c is such that M decides L in 2^{n^c} steps. Existence of M and c is clear from the fact that $L \in \mathsf{EXP}$. Consider $L \in \mathsf{EXP}$. Consider the polytime Turing Machine S that reduces L to A. Now providing the turing machine with an Oracle to the language A, S^A can decide L while still functioning in polytime. Thus, $L \in \mathsf{P}^A$ which implies that $\mathsf{EXP} \subset \mathsf{P}^A$.

Given $L \in \mathsf{NP}^A$, we can simulate its actions though an exponential time machine by enumerating its non deterministic choices and simulating all its oracles calls as $A \in \mathsf{EXP}$. Therefore, $\mathsf{NP}^A \subset \mathsf{EXP}$. Trivially, we have $\mathsf{P}^A \subset \mathsf{NP}^A$ and thus we have $\mathsf{EXP} \subset \mathsf{P}^A \subset \mathsf{NP}^A \subset \mathsf{EXP} \implies \mathsf{P}^A = \mathsf{NP}^A$. □

Definition 2.3. Given $B \subset \{0,1\}^*$, we define $U_B = \{1^n : \text{There is a string of size } n \text{ in } B\}$.

Lemma 2.5. $\forall B \subset \{0,1\}^*$, $U_B \in \mathsf{NP}^B$.

Proof. A string of size n in B is given as a certificate to the machine which tests authenticity through queries to the oracle that decides B. Equivalently, we nondeterministically guess a string of size n and query its presence in B using the oracle. □

Lemma 2.6. $\exists B \subset \{0,1\}^*$ *such that* $U_B \notin \mathsf{P}^B$.

Proof. We construct the language B in stages such that in each stage only the fate of only finitely many strings is decided. We look at the *ith* stage of construction and thus inductively build up the language. We start with the empty language. Let B_k denote the set of strings whose fate has been decided by the end of stage k.

Stage i - As only finitely many strings have been decided by this stage, we choose n such that $\forall x \in B_{i-1} : n > |x|$. Let M_i be the Turing Machine described by the binary expansion of i. Now we run M_i^B on 1^n for $2^n/10$ steps. Whenever M_i queries the oracle for string $y \in B_{i-1}$. Whenever M_i queries on a string $z \notin B_{i-1}$, we then decide that the string is not in B. After $2^n/10$ computational steps if M_i^B accepts 1^n then we declare that no string of size n is in B. Else, we choose a string of size n whose fate is undecided and add it to B. The existence of such a string is guarenteed as M_i^B can decide atmost $2^n/10$ strings of size n and n was chosen such that no string of size n was decided in an earlier stage. Thus, the language B is constructed inductively by following through on all stages.

Now we prove that $U_B \notin P^B$. Assume there exists an oracle Turing machine M^B that runs in time $n^c : c \in \mathbb{N}$ that decides U_B. Since $\forall c : n^c = o(2^n/10)$, there exist N such that $\forall n > N : n^c < 2^n/10$. Since there are infinitely many strings corresponding to a Turing Machine M, we chose an $i > N$ such that $M_i = M$. Since by construction, $\forall i : M_i^B$ answers incorrectly on 1^n which contradicts our premise. Thus, $U_B \notin P^B$. □

Proof of Theorem 2.3. From Lemma 2.4, we have a language A such that $\mathsf{P}^A = \mathsf{NP}^A$. From Lemmas 2.5 and 2.6, we have a language B such that $\mathsf{P}^B \neq \mathsf{NP}^B$, which proves the theorem. □

The above theorem shows that any proof or disproof of $\mathsf{P} = \mathsf{NP}$ cannot be based just on properties a and b, as any such result would relativize contradicting the above theorem. Thus, the above theorem gives an indication that such a proof must depend on details of Turing machines that fail in the presence of oracles.