Computational Complexity **(Due: Flexible)**

Assignment #3 (Arora Barak Ch - 3)

Instructor: Shreesh Maharaj *Name:* Jimut Bahan Pal

Q. 1. Show that **SPACE(n)** $\neq$ **NP**

Soln. In order to prove **SPACE(n)** $\neq$ **NP**, we need to identify a language in **SPACE(n)** which is not in **NP**. Not every language in **SPACE(n)** would work, for example [1] regular language is **SPACE(n)** and all of these are also in **NP**.

If some machine in **SPACE(n)** does run in exponential time, it could be that there could be another machine accepting the same language that runs in polynomial time especially if we are allowed to use non-determinism. As an example, it is suspected that **SAT** which is in **SPACE(n)** requires exponential time but it also does belong to **NP**.

We need not show **NP** $\subsetneq$ **SPACE(n)** or **SPACE(n)** $\subsetneq$ **NP**, since they are all open questions so far as we know. All we need to do is to prove that these two sets are not equal, i.e., unequal.

A log space reduction from language L_1 to language L_2 is a function R which can be computed by a deterministic log space TM [2] such that x $\in L_1$ iff R(x) $\in L_2$. A complexity class C is closed under log-space reduction if for any log space reduction R from L_1 to L_2, $L_1 \in C$ iff $L_2 \in C$. We need to show first that NP is closed under log space reduction. Then we need to show that SPACE(n) is not closed under log space reduction by space hierarchy theorem $L_1 \in SPACE(n^2)$ but $L_1 \notin SPACE(n)$.

For any log space reduction R from L_1 to L_2, which is in NP, we may execute the log space Turing machine R and a non deterministic polynomial time TM that decides L_2. The execution time of the first part is a polynomial of the input length since **SPACE(logn)**$\subset$ **P** and its output length is also bounded by the same polynomial. Hence the execution time of the second part is bounded by the composition of two polynomials, which is in turn polynomial of the original input length. Therefore, $L_1 \in NP$ and NP is closed under log space reduction.

We proceed to show that **SPACE(n)** is not closed under log space reduction. For any language $L_1 \in SPACE(n^2)$, we define a new language L_2. For any x $\in L_1$ whose length is n, x\$...\$ (there are $n^2 - n$\$'s) where \$ is a symbol outside the alphabet L_1. Let R be a TM (a reduction) which pads $n^2 - n$ \$'s after the input, whose length is n. Clearly, R runs in polynomial time (log space). We modify the original quadratic space Turing Machine for L_1 to ignore \$. The new Turing Machine checks:

- The length of the input x is a square number, say n^2
- The first n symbols are from the alphabet of L_1
- The following $n^2 - n$ symbols are all \$'s.

Then the new TM simulates the original TM on the first n symbols. This new TM is linear-space TM because the counting and checking phase only requires $O(\log|x|)$ space and the simulation phase requires $O(n^2) = O(|x|)$ space.

Note: x is the padded string and not the original input string. Hence the language L_2 decided by this new TM, we have $L_2 \in SPACE(n)$. Now pick a language $L_1 \in SPACE(n^2)$ but $L_1 \notin SPACE(n)$. By the padding argument, there exist a language $L_2 \in SPACE(n)$ and a logspace reduction R from L_1 to L_2. Thus $SPACE(n)$ is not closed under logspace reduction.

Q. 2 (a). Prove that the function H defined in the proof of Theorem 3.3 is computable in polynomial time.

Soln. Many problems that were in NP turned out to be NP-complete. This gave rise to the conjecture [3] that the class NP is the union of two disjoint classes P and NP-complete. If P turns out to be equal to NP, then the conjecture is true, but if it turns out that the conjecture is false then P $\notin$ NP, i.e., P $\notin$ NP, then $\exists$ a language L $\in$ NP $\setminus$ P that is not NP-complete.

From Ladner's theorem we know that suppose that P $\neq$ NP, then $\exists$ a language L $\in$ NP $\setminus$ P that is not NP-complete.

If P $\notin$ NP, then $\exists$ at least a language SAT $\in$ NP $\setminus$ P. Consider $SAT_H = \{<\phi\ 1^{n^{H(n)}}>|\ \phi \in$ SAT and $|\phi| =$ n $\}$. Where, H:$\mathbb{N}\rightarrow\mathbb{N}$ is a poly-time computable function. We need to consider two cases:

- H(n) = c, a constant $\forall$ n. So, SAT_H is simply SAT with a polynomial amount of padding. Thus, SAT_H is also NP-complete and is not in P assuming P $\neq$ NP.
- H(n) tends to ∞ with n, thus the padding is of polynomial size. Our claim is that SAT_H is not NP-complete.

For a contradiction, assume that SAT_H is NP-complete. So, $SAT \leq_P SAT_H$. So, instances of SAT_H should be of length $O(n^i)$. So, $|\phi|$ + $|\phi|^{H(|\phi|)} = O(n^i)$. Hence, $|\phi|$ = o(n).

The above implies poly-time reduction from a SAT instance [4, 5] of length $O(n)$ to a SAT instance of length $o(n)$, which in turns implies SAT can be solved in poly-time. This contradicts P $\neq$ NP. So, H has to be designed properly so it grows tending towards ∞.

$H(n)$ is the smallest number i< $\log\log n$ such that for every x$\in$$\{0,1\}^*$ with |x| $\leq$ logn, M_i halts on x within $i|x|^i$ steps and M_i outputs 1 iff x $\in SAT_H$, where M_i is the machine represented by the binary expansion of i according to the representation scheme of UTM $\mathcal{U}$. If there is no such i, then let $H(n) = \log\log n$. $H(n)$ is well defined, to compute H(n), we need to compute $H(k)$ for every k $\leq \log n$. Simulate at most $\log\log n$ machines for every input of length at most $\log n$ for $\log\log n(\log n)^{\log\log n} = o(n)$ steps. Compute SAT on all inputs of length at most $\log n$. So, $H(n)$ can be computed in polynomial time $O(n^3)$.

Q. 2 (b). Let SAT_H be defined as in the proof of Theorem 3.3 for a polynomial-time computable function H : $\mathbb{N} \rightarrow \mathbb{N}$ such that $\lim_{n\rightarrow\infty} H(n) = \infty$. Prove that if SAT_H is **NP**-complete, then SAT is in **P**.

Soln. $H(n)$ is the smallest $\alpha < \log\log n$ such that for every $x \in \{0,1\}^*$, and $|x| \leq \log n$, M_α outputs $SAT_H(x)$ in time $\alpha|x|^\alpha$. If such an α doesnot exists, then $H(n) = \log\log n$. $H(n)$ has been defined in such a way that if $SAT_H \in P$, then $H(n) = O(1)$, and if $H(n) < C$ for some constant C, for infinitely many values of n, then $SAT_H \in P$. This fact implies that if $SAT_H \notin P$, then $H(n) \rightarrow \infty$ as $n \rightarrow \infty$. Suppose SAT_H is NP-complete, then we must have a polynomial time reduction from SAT to SAT_H. $SAT \in NP$ and every problem can be reduced to an NP-complete problem.

$\phi \xrightarrow[poly-time\ reduction]{f} \phi o 1^{n^{H(n)}}$, $|\phi| = m$ and $|\psi| = n$, such that, ϕ is satisfiable iff ψ is satisfiable. Therefore, $|\psi o 1^{n^{H(n)}}| = n + 1 + n^{H(n)} \leq p(m)$ where $p(m)$ is a polynomial time function in m. Because, $SAT_H \notin P$, $H(n)$ tends to a very large value for n. Let's assume that $H(n) > 0$, and $p(m) = m^5$, then $n + 1 + n^{H(n)} \leq m^5 \implies n \leq \sqrt{m}$.

Thus, the larger the values of $H(n)$, and proportional to m (the length of the formula ϕ), the mapping will reduce ϕ to ψ where the length of ψ would be smaller by some fixed polynomial factor ($\sqrt{m}$ as shown above). Thus, performing the same reduction again on ψ and so on, we will get a simple polynomial time algorithm that would decide SAT, which cannot be possible. Hence, a contradiction.

Q. 3. Show that there is an oracle A and a language L $\in$ **NP**^{A} such that L is not polynomial-time reducible to 3SAT even when the machine computing the reduction is allowed access to A.

Answer by SHREESH MAHARAJ [6]

Soln. From the Baker Gill Solovay (BGS) theorem, they give a language **B** and U_B such that $U_B \in NP^B$ and $U_B \notin P^B$, thus proving that there are oracles **B** for which $P^B \neq NP^B$. We shall modify the U_B and B to $U_{B'}$ and B' such that we get a new language that cannot be reduced to 3SAT even if there is availability of B' as an oracle.

First assume that we can pad every 3SAT boolean instance ϕ to ϕ' with some additional dummy 3CNF expressions such that $|\phi'|$ is odd and they are equivalent, i.e., ϕ is satisfiable iff ϕ' is satisfiable. We can do it in $n + O(1)$ time with $O(1)$ padding, but even if it takes polynomial time and extra polynomial padding, it doesnot matter.

Now we need to combine the B and 3SAT to B' somehow so that BGS theorem still holds but additionally 3SAT $\in P^{B'}$. So, we do something like the following:

$U_{B'} = \{1^n | \exists x \in B$, such that $|x| = 1^{2n}\}$ and
$B' = B'_{constructed} \cup \{\phi | \phi \in 3SAT \textit{ and } |\phi| \textit{ is odd}\}$

Now we shall construct $B'_{constructed}$ according to the theorem such that if the deterministic machine $M_i^{B'}$ for input 1^n (n is determined as in theorem) asks the oracle B' a query of odd length we check if it is 3SAT and answer correctly but if it asks a query of even length, we proceed according to the construction, that is, answering correctly if it is already in the otherwise answer no everytime. Then since we are running for 1^n we flip the answers at 2n length so that $M_i^{B'}$ doesnot decide $U_{B'}$. We can prove similarly as in the BGS theorem that for this B' and $U_{B'}$ too, we have $U_{B'} \in NP^{B'}$ and $U_{B'} \notin P^{B'}$. $U_{B'} \in NP^{B'}$ is easy to prove. We construct a non-deterministic TM which for input 1^n creates non-deterministic branches that runs for 2n steps to generate a different 2n length string and then asks oracle B' if the 2n-length string is in B', and if the answer is yes it accepts 1^n else it rejects 1^n. This construction shows that $U_{B'} \in NP^{B'}$. $U_{B'} \notin P^{B'}$ can be proved with the help of diagonalization argument. Basically it is different from every $L(M_i^{B'})$ for every oracle Turing Machine that have B' as an oracle. This is because of how we construct $B'_{constructed}$. Now we shall prove by contradiction that there doesnot exist a reduction from $U_{B'}$ to 3SAT even with the availability of oracle B'. Assume there is a reduction using oracle B', i.e., $U_{B'} \leq_P^{B'} 3SAT$. That means we can reduce a string of the form 1^n to a 3SAT instance ϕ using a polynomial-time deterministic machine which uses B' as oracle. We can now describe a deterministic TM $M^{B'}$ which will decide strings $U_{B'}$ in polynomial time using B' as an oracle. First this machine reduces the input 1^n to a 3SAT-instance ϕ using B' as an oracle. This can be done because we have the reduction above. Then if ϕ is not odd length $M^{B'}$ will pad it to make ϕ' which is odd length. Next, it will give this ϕ' to oracle B', and get the answer yes/no. It will accept if the answer is yes and reject if the answer is no.

This machine is deterministically polynomial and uses oracle B'. Thus we have proved that $U_{B'} \in P^{B'}$, a **contradiction**. Therefore $U_{B'} \not\leq_P^{B'} 3SAT$.

Disclaimer: All the answers are collected, shamelessly copied, plagiarized from the internet, and are not the author's creation.

References

[1] Yuval Filmus. Show that np is not equal to space(n), cs stackexchange question, 2018. https://cs.stackexchange.com/questions/93434/show-that-np-is-not-equal-to-spacen last accessed June 18, 2020.

[2] Yuh Duah Lyuu. Theory of computation, sol.to h/w - 2, 2010. https://www.csie.ntu.edu.tw/~lyuu/complexity/2010/20101026s.pdf last accessed June 18, 2020.

[3] Arijit Bishnu. Lecture 7: Diagonalization, 2010. https://www.isical.ac.in/~arijit/courses/spring2010/slides/complexitylec7.pdf last accessed June 18, 2020.

[4] Chandan Saha. E0 224: Computational complexity theory, 2014. https://www.csa.iisc.ac.in/~chandan/courses/complexity14/notes/lec7.pdf last accessed June 18, 2020.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, USA, 1st edition, 2009.

[6] Shreesh Maharaj. How can i show that the cook-levin theorem does not relativize?, 2016. https://cs.stackexchange.com/questions/53428/how-can-i-show-that-the-cook-levin-theorem-does-not-relativize last accessed June 18, 2020.